Riscy Whitepaper

version 0.1

A Turing-complete RISC-V Sidechain on eCash

May 2026

0xShomari

https://riscy.network

Table of Contents

1. Abstract

Riscy is a Turing-complete RISC-V sidechain built on a fork of the Nexa blockchain and secured by Bitcoin miners via the Drivechain mechanism (BIP-300/301) on the eCash network. By combining Nexa's high-throughput UTXO architecture with a general-purpose RISC-V virtual machine, Riscy enables smart contract programmability comparable to EVM-based chains while preserving the parallelism, scalability, and sound-money properties of the UTXO model.

This paper describes the architecture, RISC-V execution environment, Drivechain integration, scalability model, developer toolkit, and roadmap of the Riscy network. It is written for developers, protocol researchers, miners, and builders who want to understand the technical foundations of Riscy before deploying contracts or participating in the network.

2. Introduction

Bitcoin's scripting system is intentionally limited. This constraint preserves security and predictability but prevents the deployment of complex smart contract logic directly on the Bitcoin network. Sidechains have long been proposed as the solution: independent chains that inherit Bitcoin's security while offering expanded functionality.

Paul Sztorc's Drivechain proposal (BIP-300/301) provides a concrete mechanism for Bitcoin-secured sidechains. The eCash hard fork, scheduled for August 21, 2026, activates Drivechains on a Bitcoin fork, enabling any number of purpose-built sidechains to launch with Bitcoin-class Proof-of-Work security from day one.

Riscy is one of the seven sidechains launching with eCash. It is purpose-built for general-purpose smart contract execution. While other sidechains in the eCash ecosystem focus on payments (Thunder), privacy (zSide), identity (Bitnames), assets (BitAssets), prediction markets (Truthcoin), or interoperability (Coinshift), Riscy provides the programmability layer: a Turing-complete execution environment where developers can deploy arbitrary logic.

The base layer is a fork of Nexa, a UTXO blockchain built by Bitcoin Unlimited that has demonstrated >100,000 TPS capacity with native token support and sub-second confirmation times. Riscy extends Nexa's architecture with a RISC-V virtual machine, enabling contracts written in Rust, C, Zig, or any language that compiles to the RISC-V instruction set.

The result is a sidechain that combines Bitcoin's security model, Nexa's throughput, and general-purpose programmability — without the global state bottleneck of account-based EVM chains.

3. Architecture Overview

3.1 Nexa-Derived UTXO Layer

Riscy's base layer is forked from the Nexa blockchain, inheriting its UTXO transaction model, block structure, and Merkle tree commitment scheme. The UTXO model is fundamental to Riscy's scalability: because each transaction output is an independent unit of state, transaction validation can be parallelized across CPU cores without contention on shared global state.

Key properties inherited from Nexa:

- UTXO-based transaction model with native token support
- Merkle tree block commitments for efficient SPV proofs
- Sub-second block propagation via compact block relay
- Native script engine for simple payment conditions
- Proven capacity exceeding 100,000 transactions per second

3.2 RISC-V Virtual Machine

Riscy adds a RISC-V virtual machine alongside Nexa's native script engine. The RISC-V VM executes smart contracts stored as ELF binaries in transaction outputs. When a transaction references a RISC-V contract, the VM loads the binary, executes it with the transaction context as input, and produces a deterministic result that the consensus layer validates.

The choice of RISC-V over EVM, WASM, or a custom bytecode is deliberate:

- Open standard: RISC-V is a free, open instruction set architecture maintained by RISC-V International. No licensing, no vendor lock-in.
- Mature toolchain: GCC, LLVM/Clang, and Rust all have production-grade RISC-V backends. Developers use their existing tools.
- Language agnostic: Any language that compiles to RISC-V works — Rust, C, C++, Zig, Go (via TinyGo), and more.
- Hardware path: RISC-V is a real CPU architecture. Contract execution can be hardware-accelerated on RISC-V silicon in the future.
- Deterministic execution: The base integer ISA (RV32I/RV64I) has no floating-point or non-deterministic instructions, making consensus-safe execution straightforward.

3.3 Block Structure

Each Riscy block contains a standard UTXO transaction set plus a contract execution segment. The block header commits to both via separate Merkle roots:

- Transaction Merkle Root: Commits to all UTXO transactions in the block (payments, token transfers, contract deployments).
- State Merkle Root: Commits to the post-execution state of all RISC-V contract storage after processing the block's contract calls.

This dual-root structure allows light clients to verify payment transactions without downloading contract state, and contract-aware clients to verify execution results independently.

4. Drivechain Integration

4.1 BIP-300 Hashrate Escrow

Riscy is secured by Bitcoin miners via BIP-300 (Hashrate Escrow), activated on the eCash network. BIP-300 allows miners to enforce sidechain peg-in and peg-out operations by voting on withdrawal bundles over an extended period (~13,150 blocks, approximately 3 months). This slow withdrawal cadence ensures that any attempted theft requires sustained majority hashrate collusion over months, making attacks economically irrational.

Key properties of BIP-300 security:

- No separate validator set: Riscy does not have its own stakers or validators. Security comes from eCash (Bitcoin fork) miners.
- No separate token staking: There is no Riscy-specific token used for consensus. The native unit is eCash.
- Permissionless participation: Any eCash miner automatically participates in Riscy's security by including sidechain commitments in their blocks.

4.2 BIP-301 Blind Merged Mining

Block production on Riscy uses BIP-301 (Blind Merged Mining). Riscy block producers create sidechain blocks and submit compact block header commitments to the eCash mainchain. eCash miners include these commitments in their coinbase transactions without needing to run Riscy node software.

This design has critical advantages:

- Zero marginal cost for miners: eCash miners secure Riscy by including a small data commitment. No additional hardware, bandwidth, or software required.
- Full eCash hashrate: Because the cost to miners is negligible, rational miners will include Riscy commitments, giving the sidechain access to the full eCash security budget.
- Sidechain sovereignty: Riscy sets its own block size, block time, fee market, and consensus rules independently of the eCash mainchain.

4.3 Peg-In / Peg-Out Mechanics

Users move eCash into Riscy by sending a peg-in transaction on the eCash mainchain to the Riscy escrow address. After sufficient confirmations, the equivalent amount becomes spendable on the Riscy sidechain.

To move funds back to eCash, users initiate a withdrawal on Riscy. The withdrawal is bundled with other pending withdrawals into a Withdrawal Bundle, which eCash miners vote on over the BIP-300 escrow period. Once approved, the funds are released on the eCash mainchain.

- Peg-in: eCash mainchain Riscy sidechain (~10 confirmations, minutes)
- Peg-out: Riscy sidechain eCash mainchain (~13,150 miner votes, months)

The asymmetric timing is inherent to the Drivechain security model: fast entry, slow exit. This ensures miners have ample time to reject fraudulent withdrawal attempts.

5. RISC-V Contract Execution

5.1 Instruction Set

Riscy's VM implements the RV64I base integer instruction set with the M (integer multiplication/division) and A (atomic operations) standard extensions. This minimal but complete ISA provides:

- 64-bit integer arithmetic and logic
- Load/store memory access
- Conditional branching and jumps
- Integer multiply, divide, remainder
- Atomic compare-and-swap (for concurrent data structures)

Floating-point extensions (F, D) are intentionally excluded to ensure fully deterministic execution across all validator nodes.

5.2 Gas Metering

Every RISC-V instruction consumes a fixed amount of gas (called "fuel" in Riscy). The fuel cost table is calibrated to real execution time on reference hardware:

- Base arithmetic/logic (ADD, SUB, AND, OR, XOR, shifts): 1 fuel
- Multiply/divide (MUL, DIV, REM): 3 fuel
- Memory load/store (LW, SW, LD, SD): 2 fuel
- Branch/jump (BEQ, BNE, JAL, JALR): 2 fuel
- System calls (host functions for storage, crypto, I/O): 10–100 fuel depending on operation

Contracts specify a fuel limit at invocation. If execution exceeds the limit, the transaction reverts. Fuel is paid in eCash (the native sidechain unit) and goes to the block producer.

5.3 Supported Languages

Any language with a RISC-V compilation target can produce Riscy contracts:

• Rust: First-class support. A Riscy contract SDK (riscy-sdk) provides host function bindings, storage macros, and a build toolchain that produces optimized RV64 ELF binaries.

• C / C++: Supported via riscv64-unknown-elf-gcc or Clang with the RISC-V backend. A C header library provides host function declarations.

• Zig: Native RISC-V backend in the Zig compiler. Minimal runtime overhead makes Zig ideal for gas-sensitive contracts.

• Others: Any language targeting RISC-V via LLVM (Go via TinyGo, Swift, etc.) can produce valid contract binaries, though SDK support may be community-contributed.

5.4 Contract Deployment

Contracts are deployed by including the compiled RV64 ELF binary in a special deployment transaction output. The binary is stored on-chain and referenced by its content hash. Subsequent transactions invoke the contract by referencing this hash and providing call data.

Contract storage uses a key-value model where each contract has an isolated storage namespace. Storage reads and writes are performed via host system calls and are committed to the State Merkle Root at block finalization.

6. Consensus Mechanism

Riscy's consensus is a hybrid of two layers:

Layer 1 (eCash mainchain): Proof-of-Work mining on SHA-256d. Miners produce eCash blocks and include Riscy sidechain commitments via BIP-301 Blind Merged Mining. This layer provides the ultimate security guarantee: reverting Riscy's history requires attacking eCash's hashrate.

Layer 2 (Riscy sidechain): Riscy block producers assemble sidechain blocks, execute RISC-V contracts, compute state roots, and submit block header commitments to eCash miners. Block production is permissionless: anyone running a Riscy full node can produce blocks.

Block Parameters:

• Target block time: 2 seconds

• Maximum block size: 32 MB (inherited from Nexa's scalable block architecture)

• Difficulty adjustment: Per-block DAA derived from Nexa's algorithm

• Confirmation depth for probabilistic finality: 6 blocks (~12 seconds)

• Confirmation depth for bridge operations: 30 blocks (~60 seconds)

Block rewards on the sidechain are funded by transaction fees and fuel payments. There is no block subsidy — the native unit is eCash, pegged 1:1 from the mainchain via Drivechain.

7. Scalability

7.1 UTXO Parallelism

The UTXO model is Riscy's primary scalability advantage over account-based chains. In an account model (Ethereum, Solana), transactions that touch the same account must be serialized. In a UTXO model, transactions that consume different outputs can be validated in parallel with zero coordination.

Riscy inherits Nexa's parallel validation pipeline:

- Signature verification is parallelized across all CPU cores using batch Schnorr validation.
- UTXO lookups use a concurrent hash map that supports lock-free reads.
- RISC-V contract executions that operate on disjoint storage keys are scheduled in parallel.

This architecture scales linearly with hardware: doubling CPU cores approximately doubles validation throughput.

7.2 Throughput Targets

Nexa has demonstrated >100,000 TPS in controlled testing. Riscy targets the same order of magnitude for simple UTXO transactions, with contract-heavy workloads achieving lower but still substantial throughput depending on contract complexity:

- Simple transfers (no contract): >100,000 TPS
- Token transfers (native UTXO tokens): >80,000 TPS
- Light contract calls (<1,000 fuel): >30,000 TPS
- Heavy contract calls (10,000+ fuel): >5,000 TPS

These are conservative estimates based on Nexa's benchmarks, adjusted for RISC-V VM overhead. Actual throughput will depend on block size, hardware, and workload mix.

7.3 State Growth Management

UTXO state grows with the number of unspent outputs, not the total transaction history. Spent outputs can be pruned. Contract storage is managed via a sparse Merkle trie that supports efficient pruning of unused storage keys. Riscy nodes can operate in pruned mode, retaining only the current UTXO set and contract state without full historical blocks.

8. Developer Toolkit

Riscy provides a comprehensive builder toolkit designed to minimize friction from idea to deployed contract:

• riscy-sdk (Rust): The primary contract development kit. Provides host function bindings (#[riscy::contract] attribute macro), storage abstractions, ABI generation, and a cargo subcommand (cargo riscy build) that produces deployment-ready ELF binaries.

• riscy-cc (C/C++): A header-only library with host function declarations, storage helpers, and a CMake toolchain file for cross-compiling to riscv64-unknown-elf.

• riscy-cli: Command-line tool for deploying contracts, invoking functions, querying state, and managing wallets. Connects to any Riscy full node via JSON-RPC.

• TypeScript SDK: A JavaScript/TypeScript library for building dApps that interact with Riscy contracts. Supports wallet management, transaction construction, contract invocation, and event subscription.

• Block Explorer: Open-source block explorer with contract state inspection, transaction tracing, and fuel profiling.

• Testnet Faucet: Free testnet coins for development and testing, rate-limited by address.

• Contract Templates: A curated library of audited, parameterized contract templates (fungible tokens, NFTs, multi-sig wallets, escrow, AMM pools) available as cargo riscy new --template <name>.

• AI-Friendly APIs: All SDK functions use structured, typed interfaces optimized for LLM tool-calling. AI agents can deploy contracts, execute transactions, and query chain state through natural function calls.

9. Native Unit and Fee Model

Riscy does not have its own token. The native unit of account is eCash, pegged 1:1 from the eCash mainchain via the Drivechain peg mechanism described in Section 4.3.

This design eliminates the need for a separate token economy, avoids speculative tokenomics, and ensures that Riscy's value proposition is purely functional: developers and users pay for computation and storage in the same asset they bridge in.

Fee structure:

• Transaction fee: Market-determined, similar to Bitcoin's fee market. Users bid for block space. Minimum relay fee is 1 sat/byte.

• Fuel fee: Contracts consume fuel (gas) priced in eCash per fuel unit. The fuel price adjusts dynamically based on block utilization, similar to EIP-1559's base fee mechanism.

• Storage fee: A small per-byte fee for contract deployment (storing ELF binaries on-chain) and per-key fee for contract storage writes.

All fees are collected by the block producer. There is no fee burning — eCash supply is governed by the mainchain's emission schedule.

10. Roadmap

Q2 2026: Testnet launch, riscy-sdk alpha (Rust), riscy-cli, testnet faucet, block explorer

Q3 2026: eCash mainnet fork (August 21, 2026), Riscy sidechain genesis, Drivechain peg activation, riscy-sdk stable release

Q4 2026: riscy-cc (C/C++ support), TypeScript SDK, contract template library, ecosystem builder grants

Q1 2027: On-chain governance for sidechain parameters, advanced contract features (cross-contract calls, delegated execution)

Q2 2027: Hardware-accelerated RISC-V validation (FPGA reference implementation), state pruning optimizations

Q3 2027: Formal verification toolkit for RISC-V contracts, security audits, inter-sidechain messaging with other eCash sidechains

Q4 2027: Mobile SDK, light client protocol, decentralized contract registry

11. Conclusion

Riscy fills a specific and critical role in the eCash sidechain ecosystem: general-purpose smart contract execution secured by Bitcoin-class Proof-of-Work. By forking Nexa's battle-tested UTXO architecture and adding a RISC-V virtual machine, Riscy achieves a combination that no existing chain offers: UTXO parallelism, Turing-complete programmability, language-agnostic contract development, and Drivechain security — all without introducing a new token.

Developers write contracts in Rust, C, Zig, or any language they already know. Users pay fees in eCash, the same asset they bridge in. Miners secure the chain by doing what they already do. No new trust assumptions, no new tokens, no new consensus mechanisms.

Smart contracts on Bitcoin. Finally general-purpose.

12. References

[1] eCash Hard Fork — https://ecash.com

[2] Nexa Blockchain — https://nexa.org

[3] BIP-300: Hashrate Escrow — https://github.com/bitcoin/bips/blob/master/bip-0300.mediawiki

[4] BIP-301: Blind Merged Mining — https://github.com/bitcoin/bips/blob/master/bip-0301.mediawiki

[5] RISC-V Specification — https://riscv.org/technical/specifications/

[6] Nexa Blockchain Specification — https://spec.nexa.org

[7] Bitcoin Unlimited — https://www.bitcoinunlimited.info

[8] Paul Sztorc, Drivechain — https://www.drivechain.info

[9] aserti3-2d DAA — https://reference.cash/protocol/forks/2020-11-15-asert

[10] Schnorr Signatures (BIP-340) — https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki