REMZAR

An Energy-Efficient, Single-Binary, Post-Quantum Layer-1 Blockchain

Built as a sovereign base layer for verified data.

Whitepaper

Protocol Version 1.0.0

Mainnet start date: 2026-06-26

Author: Ronald Delamotte

Remzar Blockchain

www.remzarchain.com

2026

Document Status and Scope

Remzar is presented as a working Rust-based Layer-1 blockchain implementation, not as a purely theoretical proposal. This whitepaper focuses on the base protocol: node operation, storage, networking, post-quantum cryptography, Proof of Registry consensus, economics, security boundaries, and the operational roadmap.

Abstract

Remzar is a single-binary, post-quantum-oriented Layer-1 blockchain implemented in Rust. It is designed to make a public blockchain network practical to operate from one executable while preserving bounded validation rules, deterministic block production, local wallet ownership, and built-in auditability. A Remzar node combines wallet generation, encrypted key storage, database initialization, peer-to-peer networking, validator registration, block production, transfers, chain inspection, diagnostics, and audit export through a guided command-line workflow.

Remzar replaces energy-race mining with Proof of Registry (PoR), a deterministic validator-participation model built around an explicit on-chain registry, validator warmup, quarantine, heartbeat renewal, lease expiry, dead-peer eviction, and deterministic leader selection. Blocks are organized around 30-second slots. A short 2-second local pacing puzzle acts as rate control, not as competitive Proof-of-Work. Within each slot, deterministic failover rounds allow the network to move past unavailable leaders without changing the outer block cadence.

Remzar's cryptographic model is centered on post-quantum primitives. Block batches are committed through Merkle roots and attested with one ML-DSA-65 signature per block, making signature verification an O(1) block-level operation instead of an O(N) transaction-level bottleneck. Remzar uses BLAKE3-style hashing and 64-byte commitments for fast integrity checks, while wallet storage is protected with passphrase hardening, authenticated encryption, and memory-hygiene discipline.

The native asset, REMZAR, has a total maximum supply of 200 million coins. Issuance comes from validator block rewards only. The reward schedule steps down through fixed 500,000-block eras and then enters a 1 REMZAR-per-block stabilized tail until the supply cap is exhausted around the year 2172. The design goal is a chain that ordinary operators can run locally, organizations can audit directly, and future cryptographic upgrades can be introduced deterministically without abandoning protocol simplicity.

1. Introduction

Most Layer-1 blockchain systems impose a gap between the ideal of self-sovereign participation and the practical work required to run infrastructure. Nodes may require multiple services, complicated configuration, external indexers, hosted APIs, or specialized DevOps knowledge. As a result, users and organizations often depend on third-party infrastructure even when the protocol itself is public.

Remzar is built around the opposite assumption: a base-layer blockchain should be operable by a normal user or organization from a single program. The node should create its own local storage layout, guide the operator through wallet creation, connect to peers, produce and verify blocks, expose balances, send transfers, export logs, and generate audit reports without requiring hidden services or fragile scripts.

The Remzar design is intentionally conservative at the base layer. The chain uses fixed protocol constants, bounded input sizes, deterministic timing, explicit validator membership, and strict storage boundaries. Rather than pushing all complexity into the base protocol, Remzar keeps the Layer-1 readable and auditable while leaving higher-frequency or experimental functionality to optional utilities, mini-chains, sidechains, or future research tracks.

Remzar's thesis is simple: a modern public blockchain can be post-quantum-oriented, energy-efficient, locally operable, auditable, and economically transparent without requiring a Proof-of-Work race or a heavyweight Proof-of-Stake governance system.

2. Design Goals

2.1 Single-Binary Operation

Remzar is designed to run as one compiled Rust executable. The operator does not need to assemble a separate wallet application, database service, audit exporter, miner, networking daemon, or debugging tool for normal operation. The same binary exposes node runtime and operator workflows.

- **Full lifecycle from one program:** database setup, wallet generation, validator registration, node start, peer connectivity, transfers, balances, logs, and audit exports.
- **Guided command-line workflow:** the menu-driven interface reduces hidden configuration and makes node operation repeatable.
- **Local-first ownership:** wallets, chain data, logs, and reports are stored locally under deterministic directories.

2.2 Energy-Efficient Consensus

Remzar does not use open-ended hash competition. Its Proof of Registry model selects validators from an explicit registry and uses deterministic slot timing and failover. The local puzzle is a pacing guard, not an energy race. Security is based on identity, eligibility, bounded validation, and reproducible leader selection rather than maximum electricity consumption.

2.3 Post-Quantum-Oriented Security

The current Remzar design centers the protocol around ML-DSA-65 for post-quantum digital signatures. Remzar's dependency stack also includes ML-KEM-768 for post-quantum key encapsulation, supporting a roadmap and implementation model for post-quantum session establishment and transport protection where Remzar-specific secure channels use it. The paper describes Remzar as post-quantum-oriented because the base security model is designed around standardized post-quantum primitives, while still recognizing that external networking components and future interoperability layers must be audited as part of release hardening.

2.4 Bounded Validation and Predictable Limits

Remzar uses constant-defined limits to reduce denial-of-service surface area and to make resource usage predictable. Blocks, transaction buffers, transaction counts, private-key input lengths, timestamp skew, serialized batches, and network participation rules are all bounded by explicit constants. This makes validation easier to audit and easier to reason about under load.

2.5 Auditability by Default

Remzar treats auditability as part of the product, not as an afterthought. Logs, error records, blocks, transaction batches, rewards, and chain ranges are stored and exported through built-in mechanisms. Operators can produce structured JSON and PDF-style audit artifacts, verify cryptographic fingerprints, and inspect chain state without requiring custom forensic tooling.

3. Protocol Facts at a Glance

Category	Current Remzar Value
Project name / ticker name	Remzar “ZAR”
Protocol version	1.0.0 / VERSION = 1
Native asset	REMZAR
Official start date	2026-06-26
Default node port	36213
Implementation language	Rust, edition 2024
Deployment model	Single Rust binary; Docker-compatible release path
Consensus	Proof of Registry (PoR) with deterministic leader rotation
Block interval	30 seconds
Local pacing puzzle	2 seconds minimum proposer pacing
Failover window	12 seconds: 2s puzzle + 5s build slack + 5s leader grace
Failover proposal deadline	24 seconds into the slot, preserving a 6-second gossip tail
Maximum failover rounds	2 deterministic rounds per 30-second slot
Max block size	2 MiB
Block overhead reserve	16 KiB
Transaction buffer limit	2 MiB disk-backed buffer
Max transactions per block	8,192 active hard cap
Signature model	One ML-DSA-65 post-quantum signature per block batch
Commitment model	Merkle root over transaction IDs
Hashing	BLAKE3 / RemzarHash-style 64-byte commitments
Wallet protection	Argon2id passphrase hardening + AES-GCM authenticated encryption + zeroization discipline
Max supply	200,000,000 ZAR
Issuance source	Validator block rewards only
Genesis reward	0 ZAR
Rewardless prefix	1 block
Reward ladder	50, 25, 10, 5, 4, 3, 2, 1 ZAR per block, each for 500,000 blocks
Stabilized tail	1 ZAR per block until the cap is exhausted
License	MIT OR Apache-2.0

4. System Overview

4.1 Core Node Architecture

A Remzar node is a cohesive Layer-1 application rather than a loose collection of services. The node includes the operator interface, wallet tooling, peer-to-peer networking, RocksDB-backed storage, chain validation, validator registry logic, block production, transaction processing, rewards, diagnostics, and audit exports.

- **Operator Interface:** menu-driven CLI for setup, node start, wallets, transfers, balances, chain views, logs, audit exports, and utilities.
- **Storage Layer:** RocksDB-backed databases segmented by function so chain state, registry state, accounts, logs, peers, and sidechain data remain organized.
- **Network Layer:** libp2p-based peer discovery, gossip, request/response exchange, peer identity, and transport multiplexing.
- **Cryptography and Integrity:** post-quantum signatures, Merkle commitments, RemzarHash/BLAKE3-style hashing, password-hardening, and authenticated wallet encryption.
- **Consensus and Registry:** Proof of Registry validator membership, leader scheduling, activation delay, quarantine, heartbeat renewal, lease expiry, and deterministic failover.

4.2 Why Rust

Rust fits Remzar's goals because it provides memory safety without a garbage collector, strong concurrency rules, predictable performance, and a disciplined Cargo-based build system. These properties are useful for a blockchain node because consensus software must handle untrusted inputs, networking, storage, cryptography, and long-running process behavior without allowing avoidable memory-unsafe failure classes.

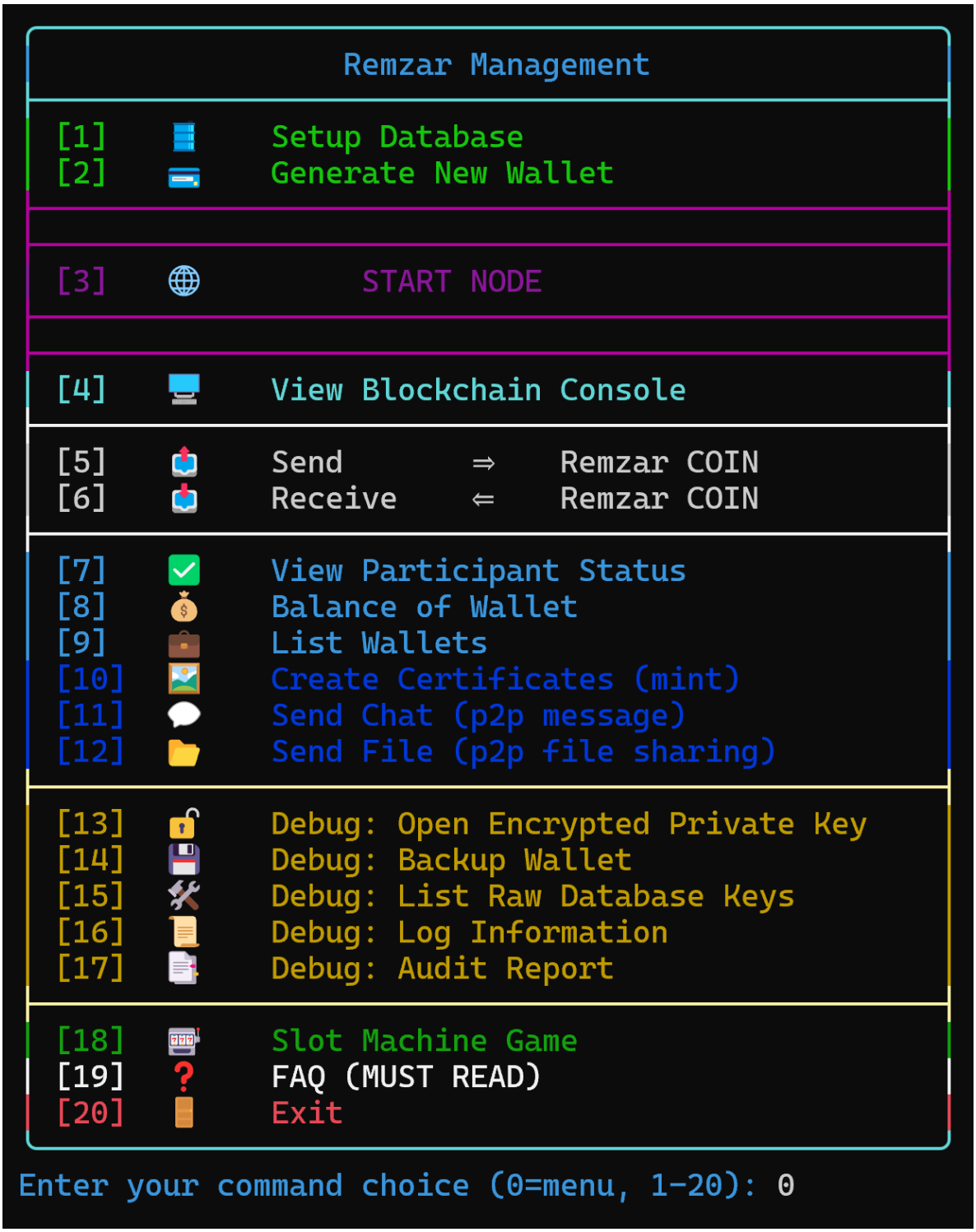
4.3 Local Directory Architecture

On setup, Remzar creates a deterministic local directory layout. This makes backups, diagnostics, audits, migration, and operator support easier because each subsystem has a predictable storage location.

Directory	Purpose
000.wallets	Encrypted wallet files and wallet-related local data
001.database_db	General CLI and node-state database
002.blockchain_db	Main blockchain storage
003.blockchain_db_console	Viewing blockchain blocks (temporary folder, not core DB)
003.registry_db	Validator/participant registry storage
004.log_db	Structured logs, error records, and diagnostics
005.audit_reports	Exported audit reports
006.accountmodel_db	Account model and state-tree related storage
007.peerlist	Peer list and bootstrap state
008.sidechain_db	Optional sidechain and mini-chain storage

4.4 Menu Map: 20 Operator Operations

Remzar's menu map is part of the protocol's operational design. It gives the operator one consistent surface for the full node lifecycle.



#	Operation	Subsystem / Data Flow
1	Setup Database	Initialize directories and RocksDB databases
2	Generate Wallet	Create keys, encrypt wallet, write under 000.wallets
3	Start Node	P2P bootstrap, chain initialization, synchronization, runtime loop
4	Blockchain Console	Real-time chain inspection tools
5	Send Coins	Unlock wallet, build transaction, sign, propagate
6	Receive Coins	Read-only inbound wallet view
7	Participant Status	Registry and leader schedule view
8	Check Balance	State read and balance computation
9	List Wallets	Filesystem discovery under wallet directory
10	Certificate / NFT	File hash, submit record, write receipt, create RWA certificate
11	Send Chat	Signed peer message over P2P
12	Send File	Chunked transfer, receiver merge, integrity verification
13	Wallet Utilities	Guarded maintenance and recovery tools
14	Backup Wallet	Copy encrypted wallet file
15	Debug Wallet Keys	Validate decryptability and print safe metadata
16	Export Logs	Bounded export from log database
17	Audit Report	Export a chain range as JSON/PDF-style report
18	Games	Optional module outside the base consensus core
19	FAQ	Built-in operator manual
20	Exit	Safe shutdown flow

5. Cryptographic Stack

5.1 Post-Quantum Digital Signatures: ML-DSA-65

Remzar's updated cryptographic identity model centers block attestation and validator authentication around ML-DSA-65, the Dilithium-derived digital signature scheme standardized through FIPS 204. In the Remzar code configuration, ML-DSA-65 secret-key and signature sizes are treated as explicit constants, including the MLDSA65 secret-key byte length and Guardian signature length.

The most important scaling decision is that Remzar does not require one post-quantum signature verification per transaction inside a block. Instead, transactions are hashed, committed into a Merkle root, and the root is signed once for the full block batch. This keeps post-quantum signature verification at block scope.

5.2 Post-Quantum Key Encapsulation: ML-KEM-768

Remzar's dependency stack includes ML-KEM-768, the Kyber-derived key encapsulation mechanism standardized through FIPS 203. ML-KEM-768 is the intended primitive for post-quantum key establishment in Remzar-specific secure transport/session flows. In a production security review, every transport path should be

checked to confirm whether it is classical, post-quantum, or hybrid, because third-party P2P frameworks may still contain classical components unless explicitly wrapped or replaced.

5.3 Hashing and RemzarHash

Remzar uses fast cryptographic hashing for transaction IDs, block headers, Merkle leaves, audit fingerprints, and deterministic genesis construction. The configuration refers to 64-byte genesis hashes and Merkle roots, and the benchmark notes measure BLAKE3 style 64-byte output. This gives Remzar high throughput for hashing-heavy paths while preserving deterministic commitments for audit and verification.

5.4 Merkle Commitments and One Signature per Block

Each block batch follows a simple commitment pipeline:

1. Build transactions.
2. Serialize transactions deterministically.
3. Hash transactions into 64-byte transaction IDs.
4. Compute a 64-byte Merkle root over the transaction IDs.
5. Sign the Merkle root once with ML-DSA-65.
6. Verify the block using one ML-DSA-65 verification plus Merkle inclusion checks where needed.

This model gives Remzar a clean verification story: the block signature authenticates the batch commitment, and transaction inclusion is proven through the Merkle path. Signature cost is O(1) per block and amortized as O(1/N) across included transactions.

5.5 Wallet Protection

Remzar wallets are designed around local ownership and encrypted storage. Private key material is not meant to be stored in plaintext. Wallet protection combines passphrase hardening, authenticated encryption, and best-effort memory hygiene.

- **Argon2id:** hardens passphrases before key derivation.
- **AES-GCM:** provides authenticated encryption for wallet files.
- **Zeroization discipline:** reduces the lifetime of sensitive buffers in memory where practical.
- **Guarded CLI flows:** backup, debugging, and diagnostics are explicit operator actions rather than hidden behavior.

5.6 Cryptographic Agility

Post-quantum cryptography is a moving field. Remzar therefore treats cryptographic suite identifiers, deterministic versioning, and activation heights as important protocol design tools. Future changes to parameter sets, hybrid modes, or transport encapsulation can be introduced through versioned rules instead of emergency rewrites.

6. Block Model

6.1 Slot Timing

Remzar targets a 30-second block interval. One block height receives one full 30-second slot. The current local pacing puzzle is 2 seconds, intentionally short so that block production remains responsive while still preventing tight-loop proposer spam.

Timing Constant	Value	Meaning
BLOCK_CREATION_INTERVAL_SECS	30 seconds	Outer chain cadence and full block slot length
PUZZLE_CREATION_INTERVAL_SECS	2 seconds	Minimum local proposer pacing time
FAILOVER_BUILD_SLACK_SECS	5 seconds	Reserved time for assembly, signing, commit, and publish after the puzzle
FAILOVER_LEADER_GRACE_SECS	5 seconds	Additional publication grace before deterministic rotation
FAILOVER_WINDOW_SECS	12 seconds	Puzzle + build slack + leader grace
SLOT_GOSSIP_BUFFER_SECS	6 seconds	Tail of slot reserved for propagation and drift
FAILOVER_PROPOSAL_DEADLINE_SECS	24 seconds	Latest time in slot to begin/publish a useful proposal
FAILOVER_MAX_ROUNDS	2	Maximum deterministic leader rounds inside one 30-second slot
SLOT_GATE_DRIFT_SECS	2 seconds	Tight drift bound for slot/round gating

6.2 Size and Count Bounds

Remzar's block and buffer limits are part of the security model. They bound memory pressure, serialization costs, and network propagation costs.

Constant	Current Value	Security / Operational Purpose
MAX_BLOCK_SIZE	2 MiB	Hard upper bound on block size
BLOCK_OVERHEAD_RESERVE	16 KiB	Conservative reserve for metadata and signatures
TRANSACTION_BUFFER_LIMIT	2 MiB	Disk-backed mempool-like buffer limit; reject new items when full
MAX_TXS_PER_BLOCK	8,192	Hard cap preventing excessive numbers of tiny transactions
MAX_BATCH_SERIALIZED_OVERHEAD	2,048 bytes	Allowance for serialized batch framing above payload
MAX_FUTURE_SKEW_SECS	2 hours	Coarse timestamp safety bound

6.3 Throughput Interpretation

The current active protocol constants should be distinguished from public performance targets. With a 30-second interval and an 8,192-transaction hard cap, the explicit count limit alone equals about 273 transactions per second if every transaction is counted one-for-one. The benchmark notes also model a 2 MiB practical block target with approximately 15,000 compact transactions per block, which corresponds to about 500 transactions per second. Before mainnet performance is advertised, the release constants and benchmark assumptions should be aligned so that the published TPS number and active hard caps match exactly.

The important architectural result remains: the CPU-side block assembly and Merkle pipeline benchmarks are far above the conservative public target, and ML-DSA-65 signing is not the active bottleneck because one post-quantum signature is used per block rather than per transaction.

7. Consensus: Proof of Registry

7.1 Overview

Proof of Registry (PoR) is Remzar's energy-efficient validator participation model. Instead of open mining or stake-weighted governance, PoR uses explicit registry membership, deterministic eligibility, and verifiable leader selection. A validator must be known to the chain before it can propose blocks and earn rewards.

7.2 Registry and Admission

Remzar uses a persistent registry that maps node identities to wallet addresses and validator status. The registry is auditable and forms the basis of the leader committee. During early network growth, Remzar can use a founder-approved or allowlisted validator admission process to prevent spam and stabilize the bootstrap network. This is a bootstrap safety policy, not a claim that long-term decentralization is finished. The roadmap should progressively move admission toward transparent, rule-driven onboarding as the network matures.

7.3 Eligibility, Warmup, Quarantine, and Heartbeats

Validator eligibility is controlled by deterministic time and height gates.

Rule	Current Value	Meaning
Activation warmup	30 seconds	Minimum time after registration before a validator can propose
Validator activation delay	ceil(30s / 30s) = 1 block	Derived block delay before eligibility
Quarantine	4 blocks	Rejoining validators must observe a period before eligibility
Canonical renewal interval	10 blocks / 5 minutes	A live validator should renew its on-chain record periodically
Canonical lease	10 blocks / 5 minutes	Validator eligibility expires if renewal is missed
Dead-peer eviction	2 blocks / 60 seconds	Runtime live set removes disconnected or unreachable peers quickly
Heartbeat grace	0 seconds	No extra grace for known-dead runtime peers

7.4 Deterministic Leader Selection

For each 30-second slot, every node computes the eligible validator set and selects the leader deterministically from shared chain state. This avoids dependency on external randomness. Because the selection input is derived from the chain and the registry snapshot, honest nodes can independently verify whether a proposed block was produced by the eligible leader for that slot and failover round.

7.5 Failover Rounds

Unavailable leaders should not freeze the network. Remzar therefore fits deterministic failover rounds inside a slot. Each round uses the same canonical committee snapshot, but selects a different leader according to the deterministic schedule. With the current constants, a 30-second slot contains a 24-second proposal window and a 12-second failover window, allowing two proposer rounds before the final 6 seconds are reserved for gossip and settlement.

7.6 Rewards

The selected minter for a valid block receives the block reward, subject to the reward schedule, reward delay, rewardless prefix, and maximum supply cap. After the cap is reached, block rewards stop permanently. The chain may continue operating on whatever non-inflationary fee or policy mechanism is adopted in later versions.

8. Transactions, Accounts, and State

8.1 Account Model

Remzar uses a local account/state model backed by RocksDB. Balances, blocks, reward entries, transaction batches, and canonical chain views are stored in dedicated column families. The account model is designed for deterministic balance reads and verifiable state updates.

8.2 Transaction Lifecycle

1. The operator unlocks an encrypted wallet through the CLI.
2. The wallet constructs a transaction with bounded amount and canonical fields.
3. The transaction is signed or authorized according to the current Remzar cryptographic suite.
4. The node validates formatting, bounds, replay protection, and spend rules.
5. Valid transactions enter the disk-backed transaction buffer.
6. A selected block producer batches transactions, commits them through the Merkle root, signs the block commitment, and broadcasts the block.
7. Peers verify the leader, slot, block bounds, Merkle commitment, signature, and state transition before accepting the block.

8.3 Defensive Transaction Rules

- **Maximum single transfer:** 100,000,000 REMZAR in base units, preventing abnormal giant-send edge cases.
- **Replay protection:** transactions must not be reusable across incompatible contexts.
- **Double-spend prevention:** state transition logic rejects conflicting spends.
- **Strict serialization bounds:** oversized or malformed payloads fail closed.
- **Batch limits:** transaction count and byte-size limits are enforced before inclusion.

8.4 Data Anchoring and Certificates

Remzar includes a Certificate/NFT-style utility path in the CLI. The base idea is to hash a file or data object, submit a record, and produce receipts that can later verify that the data existed in a particular committed chain context. The base-chain commitment should store hashes and receipts rather than forcing large files directly into the L1 block body.

9. Peer-to-Peer Networking

Remzar uses a libp2p-based networking layer with TCP, DNS, Noise, ping, Gossipsub, Kademlia, identify, Yamux, macros, Tokio integration, and request-response features. The network layer is responsible for peer discovery, liveness, message exchange, block propagation, transaction propagation, peer identity mapping, and operator connectivity.

The default node port is 36213. Operators should open this port for Remzar node communication unless a deployment explicitly uses a different configured port.

9.1 Network Message Classes

- **Block announcements:** broadcast signed block commitments and metadata.
- **Transaction gossip:** propagate valid pending transactions through bounded channels.
- **Registry messages:** share validator identity, registration, renewal, and participation state.
- **Request-response sync:** allow peers to request missing blocks, metadata, and state-related records.
- **Operator utilities:** support signed chat, file transfer, peer diagnostics, and audit-oriented workflows without mixing these features into the core consensus rule set.

9.2 Transport Security and PQ Review Boundary

Remzar's protocol design includes post-quantum key encapsulation through ML-KEM-768. However, because practical P2P stacks often include classical transport components, the release security process should classify every connection path as classical, post-quantum, or hybrid. A professional mainnet security statement should only call a path fully post-quantum when the implementation has been reviewed end-to-end, including handshake, identity binding, session keys, replay protection, downgrade resistance, and dependency behavior.

10. Storage, Logs, and Audit Exports

10.1 RocksDB Column Families

Remzar segments storage into 19 active column families. This separation helps keep the chain auditable and prevents unrelated data from being mixed into a single opaque store.

Column	Name	Purpose
0	meta_data	Headers and blockchain metadata
1	global_metadata	Global configuration data
2	wallet_accounts	Account and wallet data
3	network_data	Peer nodes, validators, and P2P state
4	sidechain_data	Sidechain, mini-chain, bridge, and related commitments
5	state_data	Balances and account model state
6	transaction_data	Individual transaction records and buffer entries
7	transaction_batch_data	Finalized transaction batches
8	reward_data	Individual reward transactions
9	reward_batch_data	Finalized reward batches
10	blockmint_data	Finalized blockmint data
11	logs	Events, debugging, and errors
12	block_to_hash	Block hash to serialized block mapping
13	tx_to_hash	Transaction hash to serialized transaction mapping
14	identity	Node identity to wallet address mapping
15	block_meta_by_hash	Block metadata by hash
16	batch_by_block_hash	Transaction batch by block hash
17	canonical_height_to_hash	Canonical height to block hash view
18	canonical_chain_view	Tip hash, tip height, and canonical chain view

10.2 Structured Logs and Detection

Remzar's detection and error-handling architecture is intended to make failures visible and bounded. Protocol guards cover liveness, replay rejection, double-spend rejection, Sybil resistance, strict size/format limits, dead-peer eviction, and serialization safety. Errors should be structured, serializable, and exportable rather than hidden behind panics or ambiguous runtime messages.

10.3 Audit Reports

Operators can export chain ranges and recent logs in bounded formats. Audit exports should include the relevant blocks, hashes, Merkle roots, validator identifiers, reward information, transaction summaries, and cryptographic fingerprints. This design allows merchants, exchanges, organizations, and independent operators to verify state and activity without relying on a centralized explorer.

11. Economics and Tokenomics

11.1 Native Asset

REMZAR is the native asset of the Remzar Layer-1 blockchain. The on-chain implementation uses integer base units rather than floating-point arithmetic. The current configuration comments indicate an 8-decimal style.

11.2 Supply and Issuance

Parameter	Value
Native asset	REMZAR
Start date	2026-06-26
Maximum supply	200,000,000 ZAR
Reward supply	200,000,000 ZAR
Other issuance buckets	None in current configuration: no staking supply and no gaming supply
Genesis reward	0 ZAR
Rewardless prefix	1 block
Block interval	30 seconds
Initial block reward	50 ZAR
Stabilized block reward	1 ZAR
Reward step interval	500,000 blocks

There is no protocol-level premine, founder allocation, development reserve, treasury mint, staking bucket, or gaming bucket in the current configuration. All supply enters circulation through validator rewards until the maximum supply is reached.

11.3 Reward Schedule

Phase	Reward per Block	Duration	Approximate Date Range
Genesis	0 REMZAR	1 block	2026-06-26
Step 1	50 REMZAR	500,000 blocks	2026-06-26 to 2026-12-16
Step 2	25 REMZAR	500,000 blocks	2026-12-16 to 2027-06-08
Step 3	10 REMZAR	500,000 blocks	2027-06-08 to 2027-11-28
Step 4	5 REMZAR	500,000 blocks	2027-11-28 to 2028-05-20
Step 5	4 REMZAR	500,000 blocks	2028-05-20 to 2028-11-10
Step 6	3 REMZAR	500,000 blocks	2028-11-10 to 2029-05-02
Step 7	2 REMZAR	500,000 blocks	2029-05-02 to 2029-10-23
Step 8	1 REMZAR	500,000 blocks	2029-10-23 to 2030-04-14
Stabilized tail	1 REMZAR	~150,000,050 blocks	2030-04-14 to ~2172-11-19

The final dates are approximate because they assume an uninterrupted 30-second block cadence from 2026-06-26 00:00 UTC. Under that assumption, the reduction ladder ends around 2030-04-14, and the reward supply is exhausted around 2172-11-19. After the cap is reached, inflation from block rewards becomes 0% permanently.

11.4 Stabilized Issuance

At 30-second blocks, a 1 REMZAR-per-block tail produces about 1,051,200 REMZAR per year. Relative to a 200,000,000 REMZAR maximum supply, that is about 0.526% of the maximum supply per year before trending toward 0% as the cap is approached. This stabilized issuance is not perpetual inflation; it stops when MAX_SUPPLY is exhausted.

11.5 Plain-English Tokenomics Summary

- One native asset: REMZAR.
- One issuance mechanism: validator block rewards under Proof of Registry.
- Maximum supply: 200,000,000 REMZAR.
- No configured staking, gaming, treasury, or founder mint bucket.
- Genesis block is rewardless.
- Rewards step down through the ladder and then stabilize at 1 REMZAR per block.
- After the cap is reached, block rewards stop permanently.

12. Genesis and Initial Distribution

Every Remzar network begins from a deterministic genesis configuration loaded by nodes during first initialization. The genesis block anchors the chain identity, initial hash state, and initial monetary state. In the current configuration, the genesis block has a zero reward and uses fixed 64-byte previous-hash, Merkle-root, and genesis-hash values.

Genesis Field	Current Value / Description
Genesis previous hash	64 bytes of zero value, displayed as 128 hex characters
Genesis Merkle root	Fixed 64-byte configured value
Genesis hash	Fixed 64-byte canonical RemzarHash value
Genesis nonce	299,792,458
Genesis validator	Canonical r-prefixed 129-character wallet-format address
Genesis reward	0 REMZAR
Genesis JSON path	blockchain/genesis.json

The Remzar monetary state begins without a protocol-level premine. The chain's supply grows only when valid reward transactions are produced under the reward schedule. This makes the distribution model simple to audit: supply starts at zero and moves upward through visible validator rewards until the hard cap is reached.

13. Performance and Benchmarks

Remzar's benchmark notes show that the local CPU-side hash, Merkle, state-apply, block encode/decode, and block assembly paths run far above the conservative public throughput target. The benchmark suite completed with 18 default tests passed, 1 release Merkle sweep test passed, and 0 failed tests in the provided notes.

13.1 Batch-Signed Performance Model

The central benchmark conclusion is that post-quantum signatures are not the primary TPS bottleneck in the Remzar block model because one ML-DSA-65 signature is used per block batch. A block has many transactions, but only the Merkle root of the batch is signed. Verification requires one post-quantum verification per block, plus ordinary hashing and Merkle inclusion logic.

Measured Path	Benchmark Note
Block tx serialize + hash	10,000 transactions in 0.059s; about 168,870 tx/s
Block Merkle root	10,000 txids in 0.006s; about 1,683,162 tx/s
One block signature	1 ML-DSA-65 sign in about 0.007s
One block verification	1 ML-DSA-65 verify in about 0.005s
Effective block assembly	10,000 tx in 0.065s; about 153,472 tx/s before fixed one-signature cost
ML-DSA-65 primitive sign path	About 45 signatures/s; about 1,350 signatures per 30s
ML-DSA-65 primitive verify path	About 136 verifications/s; about 4,080 verifications per 30s

13.2 Public TPS Statement

A professional Remzar TPS statement should be conservative. The safe wording is: Remzar's local CPU-side pipeline benchmarks above the 500+ TPS target, but production TPS depends on the final release constants, transaction size, RocksDB write path, network propagation, validator distribution, mempool policy, block-size policy, and consensus timing. The active 8,192 transaction-per-block cap should be reconciled with any 15,000 transaction-per-block public target before mainnet marketing material is finalized.

14. Security Considerations

14.1 Threat Model

Remzar assumes the network will receive malformed payloads, oversized batches, invalid signatures, stale blocks, replay attempts, double-spend attempts, dead peers, malicious peers, and validators that may become unavailable. The design response is to fail closed, reject invalid data early, maintain deterministic eligibility and block timing, and keep all resource usage bounded by constants.

14.2 Replay and Double-Spend Protection

Transactions must not be valid indefinitely across incompatible contexts. Remzar's validation layer should reject reused authorizations, conflicting spends, and state transitions that violate account balances or canonical chain rules. Replay protection is especially important for signed messages, wallet-to-wallet tools, certificate records, and any cross-chain or sidechain feature.

14.3 Sybil and Validator Abuse Controls

Proof of Registry reduces open-participation spam by requiring validators to register before eligibility. Warmup, quarantine, heartbeat renewal, lease expiry, and admission policy further reduce join-and-mint abuse, dead-validator stalls, and uncontrolled validator churn. During early growth, the allowlisted admission model provides operational safety, but long-term decentralization requires transparent onboarding rules and monitoring.

14.4 Denial-of-Service Bounds

- 2 MiB maximum block size.
- 2 MiB disk-backed transaction buffer.
- 8,192 transaction cap per block.
- 2,048-byte serialized overhead allowance.
- Explicit private-key input length limits for ML-DSA-65 secret material.
- Timestamp skew and slot-gating drift bounds.
- Quarantine and eviction windows for unstable peers and validators.

14.5 Implementation Safety

Remzar's Rust codebase should continue to enforce a secure-by-construction policy: no unsafe code in consensus-critical paths, no intentional panics in production logic, structured errors instead of unwrap/expect behavior, deterministic serialization, and reproducible release builds. These engineering rules matter because blockchain security depends on the implementation as much as on the protocol diagram.

15. Roadmap

15.1 Phase 1: Adoption-Ready UX

- Improve wallet backup, watch-only mode, QR export, address book support, and safer recovery flows.
- Build a lightweight explorer and read-only API for blocks, transactions, account state, and audit proofs.
- Create merchant workflows: invoice-like transfers, confirmations, exportable receipts, and audit-friendly settlement records.
- Standardize wallet-to-wallet signed messages, delivery receipts, replay protection, and optional encryption.

15.2 Phase 2: Mini-Chains and Sidechains

Remzar should preserve base-layer simplicity by moving high-frequency or experimental workloads into optional mini-chains and sidechains. A mini-chain can act as a bounded local micro-ledger whose state roots periodically anchor to Remzar. A sidechain framework can define how state roots, data availability, dispute windows, and bridge assumptions work without bloating the base L1.

15.3 Phase 3: Cryptographic Evolution

Because Remzar is already designed around post-quantum primitives, the next cryptographic phase is disciplined cryptographic agility: versioned signature suites, versioned KEM/session suites, parameter updates, hybrid compatibility where necessary, key rotation tooling, and deterministic activation heights for any consensus-affecting change.

15.4 Research Track: AI Brain Blocks and Agent Commitments

AI Brain Blocks should be treated as research, not as a base-layer promise. The practical idea is to anchor auditable agent statements, hashes of prompts or inputs, model identifiers, output commitments, and verification records. Any AI-related data must remain bounded, deterministic, and inspectable. Remzar should not turn the base chain into an opaque computation engine.

16. Conclusion

Remzar is a Rust-based Layer-1 blockchain designed around a clear operating principle: public blockchain infrastructure should be simple to operate, bounded to validate, locally auditable, energy-efficient, and prepared for post-quantum cryptographic requirements. The protocol combines a single-binary node architecture with deterministic Proof of Registry consensus, 30-second block slots, post-quantum block attestation using ML-DSA-65, Merkle-root batch commitments, DB-backed local storage, and built-in audit export tooling.

This architecture avoids open-ended mining competition while preserving deterministic validation rules and transparent operational limits. Core protocol behavior is defined through explicit constants, including block timing, transaction bounds, validator eligibility rules, reward issuance, and maximum supply. Remzar's 200 million REMZAR capped supply, reward-only issuance model, rewardless genesis block, and deterministic reward ladder provide a monetary policy that is straightforward to inspect, reproduce, and verify.

By signing transaction batches at the block level rather than requiring post-quantum verification for every transaction, Remzar provides a practical path toward high-throughput validation while keeping post-quantum signature costs bounded. Its design emphasizes local ownership, predictable resource use, auditable state transitions, and reproducible node operation from a single executable.

Remzar's primary contribution is not a claim of unlimited scalability or completed decentralization. It is a disciplined base-layer design: one program, one chain, bounded validation, deterministic consensus, post-quantum-oriented cryptography, transparent issuance, and repeatable auditability. The protocol's remaining work is to continue hardening validator admission, network security, fee policy, cryptographic agility, and long-term decentralization while preserving the simplicity of the base layer.

The remaining work is to continue forever with simplicity.

References

- *Satoshi Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008*.
- *NIST FIPS 203, Module-Lattice-Based Key-Encapsulation Mechanism Standard*.
- *NIST FIPS 204, Module-Lattice-Based Digital Signature Standard*.
- *Rust programming language documentation and Cargo package-management documentation*.