AXION QUANTUM BLOCKCHAIN

AXN Genesis Configuration Technical Whitepaper

Steer Technologies | axion-steer-technologies-mainnet-1

Genesis Date: March 14, 2026 | Protocol Version: 1

Confidential — Engineering Reference Document

1. Executive Summary

The Axion Quantum Blockchain (AXN) is a purpose-built distributed ledger platform developed by Steer Technologies. Its genesis configuration — the immutable foundation from which every subsequent network state derives — is encoded in a Go module (genesis.go) that embeds all economic, consensus, and market-stability parameters in a single, deterministic, cryptographically sealed structure.

This whitepaper provides a comprehensive technical analysis of the genesis layer: its architecture, tokenomics, reward model, fee mechanics, market-stabilisation mechanisms, and the engineering disciplines applied to guarantee node consensus and prevent numeric instability.

Key Design Principle

Every parameter in the genesis configuration is immutable after creation. All nodes executing the same genesis produce identical results — a hard requirement for Byzantine-fault-tolerant consensus. No floating-point arithmetic is used anywhere in economic logic; all values are expressed in fixed-point integers with 1 AXN = 10^8 smallest units.

2. Blockchain Identity & Genesis Parameters

2.1 Chain Identification

The network is uniquely identified by the following immutable parameters:

Parameter	Value
Chain ID	axion-steer-technologies-mainnet-1
Protocol Version	1
Genesis Timestamp	2026-03-14T00:00:00Z (Unix: 1773446400)
Block Time	30 seconds
Previous Hash	0x000...000 (64 zeros)
Genesis Validator	genesis_validator

2.2 Genesis Addresses

Three canonical addresses are hard-coded in the genesis block. These addresses are immutable by design and will be finalised before mainnet deployment:

Role	Address
Liquidity Pool	AQi01670ab0a97edc27badb1974b69d77cdde95b17f0
Treasury	AQi0d7e3af7a1aa478c206d653f686c3886aad4b2ea6
Genesis Validator	AQi0649e55e17a6904419118fc10755b2c579fb02e48

2.3 ConfigHash — Deterministic Identity

The ConfigHash is a SHA-256 digest computed by serialising every genesis parameter in a fixed binary order (using big-endian encoding). It is NOT derived from JSON — this guarantees identical output regardless of platform or library version.

The hash serves three network-critical purposes: fork detection, network identity verification, and consensus validation. Any single-byte change to any parameter produces a completely different hash.

```
ConfigHash = SHA256( chainID || protocolVersion || totalSupply || preminedAmount || ... )
```

3. Tokenomics

3.1 Supply Architecture

AXN uses a fixed-point representation to eliminate floating-point non-determinism. One AXN equals exactly 100,000,000 (10^8) smallest units, providing eight decimal places of economic precision.

Metric	Value
Total Supply	100,000,000 AXN (100 million)
Pre-mined Amount	10,000,000 AXN (10% of total)
Smallest Unit	1 = 0.00000001 AXN (10^-8)
Precision	AXN_PRECISION = 100,000,000
Fixed-Point Bits	32 bits fractional (~0.0000000002 precision)

3.2 Initial Allocation

The pre-mined 10,000,000 AXN is allocated across three genesis addresses at block 0:

Recipient	Amount (AXN)	Purpose

Liquidity Pool	9,000,000 (90%)	Market liquidity & trading depth
Stabiliser Treasury	990,000 (9.9%)	Price stabilisation capital
Genesis Validator	10,000 (0.1%)	Bootstrap validator operations

Note: The genesis validator allocation carries the inscription 'MPVCx2 | In Hoc Signo Vinces', embedding a historical reference into the immutable genesis record. The total allocated pre-mine sums exactly to 10,000,000 AXN; any excess against total supply would be caught by runtime validation.

4. Block Reward Model

4.1 Phased Reward Structure

Block rewards follow a four-phase declining schedule, transitioning to an eternal logarithmically-decaying formula at block 50,000. This provides high early incentives for network bootstrap while ensuring perpetual validator income.

Phase	Block Range	Reward (AXN)	Duration (approx.)
Phase 1	1 – 100	50 AXN	~50 minutes
Phase 2	101 – 1,000	20 AXN	~7.5 hours
Phase 3	1,001 – 10,000	10 AXN	~3.1 days
Phase 4	10,001 – 50,000	5 AXN	~13.9 days
Eternal	50,001+	≤5 AXN (log decay)	Perpetual

4.2 Eternal Reward Formula

From block 50,000 onwards, the reward decays according to a logarithmic function. This ensures validators always earn a non-zero reward (minimum 1 smallest unit), while emission rate decreases predictably with network age.

```
reward(h) = BaseReward / (1 + log2(h / EternalRewardStartHeight))
```

All computations use 32-bit fixed-point arithmetic with big.Int overflow protection. The log2 function is implemented via integer bit-length detection plus linear interpolation between powers of two, yielding ±0.0000000002 accuracy — sufficient for economic calculations where sub-atomic precision is irrelevant.

Consensus Safety

GetBlockReward() never panics. If a validation gap were somehow present (a logic bug), the

function returns MinBlockReward (1 smallest unit) rather than crashing. This ensures all nodes continue to produce identical deterministic results even under error conditions.

5. Transaction Fee Model

5.1 Fee Formula

Transaction fees are computed from two components: a base fee that decreases with network maturity, and a per-byte fee that scales with transaction size. The effective fee is the greater of the two, with a hard floor of 1 smallest unit.

```
effectiveFee = max( baseFee(h), txSizeBytes × ByteFee )
baseFee(h) = BaseFee / (1 + blockHeight / BaseFeeDenominatorStartHeight)
```

Parameter	Value / Rationale
BaseFee	0.01 AXN (1,000,000 smallest units)
ByteFee	1,977 smallest units / byte
ByteFee Rationale	Steer Technologies founded in 1977; 1 KB tx ≈ 0.01977 AXN
Denominator Start	Block 100,000
Minimum Fee	1 smallest unit (0.00000001 AXN)

5.2 Gas Distribution

All collected fees are split 50/50 between the Liquidity Pool and the Treasury. The Treasury portion is further divided equally between a USD-pegged reserve and an AXN reserve. When the total is odd, the extra unit goes to the AXN reserve — a documented, deterministic rounding rule applied identically on all nodes.

Recipient	Share	Purpose
Liquidity Pool	50% (5,000 bps)	Market depth
Treasury — USD	25% (~2,500 bps)	Fiat-pegged reserve
Treasury — AXN	25%+ (2,500+ bps)	AXN reserve (+ odd unit)

6. Market Stabilisation System

AXN incorporates a protocol-level market stabilisation layer — an automated intervention engine that activates when price deviates beyond defined thresholds. This is a distinctive feature that differentiates AXN from conventional blockchains.

6.1 Economic Parameters

Parameter	Value & Meaning
StuckFiatFactor	17% — weight of fiat signal in 'stuck' price detection
StuckAXNBaseFactor	11% — weight of AXN base signal in stuck detection
FairPriceDays	30 days — rolling window for fair price calculation

6.2 Intervention Parameters

The intervention system uses a four-zone price band to determine action intensity. Zones are expressed as percentages of the fair price reference (100 = fair value):

Zone	Threshold	Action
Buy Strong	≤ 50% of fair price	Maximum buy intervention
Buy Moderate	51%–75% of fair price	Moderate buy intervention
Neutral	76%–124% of fair price	No intervention
Sell Moderate	125%–150% of fair price	Moderate sell intervention
Sell Strong	> 150% of fair price	Maximum sell intervention

6.3 Intervention Constraints

To prevent the stabilisation mechanism from itself becoming a source of volatility, the following limits apply:

- Maximum Intervention: 20% of the treasury balance per intervention event
- Minimum Balance: 10,000 AXN must remain in treasury before any intervention
- Minimum Interval: 100 blocks must pass between consecutive interventions
- Flash Crash Protection: If price drops more than 15% within 10 blocks, emergency buy-side intervention is triggered automatically

7. Engineering Architecture & Safety Properties

7.1 No Floating-Point Arithmetic

All economic computations — rewards, fees, distributions, and economic parameters — exclusively use integer arithmetic. The fixed-point representation (32-bit fractional, 4,294,967,296 units = 1.0) provides sufficient precision while being perfectly deterministic across all hardware architectures, operating systems, and Go versions.

The log2 implementation uses bit-shifting and linear interpolation, achieving ±0.0000000002 precision without any floating-point operations.

7.2 Overflow Protection

All multiplication operations that could overflow uint64 are routed through mulDivSafe(), which internally uses math/big.Int arithmetic. Division by zero is guarded with a dedicated ErrDivisionByZero sentinel error. Neither panics nor silent integer overflows are possible in the business logic path.

7.3 Immutability Pattern

GenesisConfig fields are unexported (lowercase in Go). The only construction path is NewGenesisConfig(), which performs comprehensive validation before returning. All slice accessors (RewardPhases(), InitialAllocations()) return defensive copies to prevent external mutation.

7.4 Singleton with sync.Once

The global genesis configuration is initialised exactly once via sync.Once, making it safe for concurrent access without locking after the first call. GetGlobalGenesisConfig() returns an error; MustGetGlobalGenesisConfig() panics — the latter is only appropriate during process startup.

7.5 Validation Chain

Creation of a GenesisConfig triggers eight distinct validators in sequence:

- Supply bounds (total supply > 0, pre-mine ≤ total)
- Block time range (1–3,600 seconds)
- Reward phase contiguity and non-overlap
- No coverage gap between last phase and eternal reward start
- Eternal reward formula validity
- Fee formula validity
- Gas distribution share sum = exactly 10,000 basis points
- Allocation sum ≤ total supply, all allocations non-zero
- Timestamp string and Unix integer consistency

Error Handling Policy

The codebase defines 12 typed sentinel errors (e.g., ErrInvalidTotalSupply, ErrGapInRewardCoverage). Functions return errors up the call stack — they never panic in business logic. Panics are reserved exclusively for developer-mode initialisation helpers (mulDivSafePanic, MustGetGlobalGenesisConfig) where a bug must be caught loudly at startup.

7.6 Builder Pattern for Genesis Block

The genesis block is assembled by GenesisBlockBuilder, which converts the validated configuration into genesis allocation transactions. Each transaction uses a deterministic hash derived from the allocation index and address (SHA-256 of 'genesis_alloc_{i}_{address}'), is sourced from the zero address, and carries the special type 'GENESIS_ALLOCATION' with zero fee.

8. Observed Risks & Recommendations

The following observations are provided based on code analysis. They do not indicate production-blocking defects but represent areas for consideration prior to mainnet deployment.

8.1 Reward Coverage Gap

The last fixed reward phase ends at block 50,000, and the eternal reward formula starts at EternalRewardStartHeight = 50,000. The validator validateNoRewardGap() checks that the last phase's EndBlock is at least EternalRewardStartHeight - 1 = 49,999. The current configuration sets the last phase to EndBlock: 50,000, which satisfies the validator. However, block 50,000 is claimed by both Phase 4 and the eternal formula. GetBlockReward() iterates fixed phases first, so Phase 4 wins for block 50,000 — this is functionally correct but should be documented explicitly.

8.2 Genesis Address Finalisation

The source code explicitly comments that the three genesis addresses 'will be updated before final deployment.' These addresses must be audited and verified before mainnet launch; the ConfigHash will change as a result.

8.3 log2 Linear Approximation

The log2FixedPrecise() implementation uses linear interpolation between powers of two rather than a higher-order approximation (e.g., CORDIC or polynomial). For the reward formula this is acceptable, but the approximation error grows as x approaches the upper bound of an interval. Consider documenting the worst-case error bound and verifying it is economically insignificant.

8.4 mulDivSafe Truncation

mulDivSafe uses integer division (truncation toward zero), not rounding. Results are deterministic and consistent across nodes, but may accumulate small systematic bias in long-running economic simulations. The fixedToUint64 helper does apply rounding at the final conversion step, which mitigates this concern at the output boundary.

9. Summary

The Axion Quantum Blockchain genesis configuration represents a disciplined approach to blockchain initialisation. Key engineering achievements include:

- Complete elimination of floating-point arithmetic from all economic logic
- Comprehensive overflow protection through math/big.Int
- Immutable configuration sealed at creation with defensive copying
- Deterministic SHA-256 binary ConfigHash for network identity
- Nine-stage validation pipeline preventing invalid genesis states
- Perpetual validator incentives via logarithmic eternal reward formula
- Protocol-native market stabilisation with anti-manipulation constraints
- Builder pattern for safe genesis block construction

The 90/9.9/0.1 pre-mine allocation (Liquidity Pool / Treasury / Validator) reflects a market-first philosophy: the overwhelming majority of pre-mined tokens are deployed directly into liquidity infrastructure rather than retained by founders or reserved for future issuance.

The ByteFee value of 1,977 — embedding Steer Technologies' founding year into the protocol — is a deliberate act of institutional memory encoded in the immutable genesis record, serving both an economic function (spam prevention at 0.01977 AXN per KB) and a symbolic one.

Axion Quantum Blockchain | Steer Technologies

axion-steer-technologies-mainnet-1 | Protocol v1 | Genesis: 2026-03-14

This document is derived from analysis of genesis.go. Addresses subject to change before mainnet.