

HELIXIUM (HLX) Technical Whitepaper v1.1

Proof-Native Layer-1 Blockchain for Portable Digital Verification

Version 1.1 · Pre-Mainnet Public Demo / Testnet Phase · Substrate Runtime Spec-121 · September 2026

Fixed Supply (Policy Target) 46,000,000 HLX · Mainnet monetary policy target · 4,600,000,000,000,000 nucleo maximum · Permanent enforcement pending mainnet implementation and independent verification	Base Unit 1 HLX = 100,000,000 nucleo · 1 nucleo = 10^{-8} HLX · Eight-decimal precision · No inflationary issuance policy	Runtime Polkadot SDK (Substrate) · Deployed pallets include: HelixProof, BlockHelix, HelixValidatorRegistry, HlxTransfer, Aura, Grandpa, Balances, TransactionPayment, Sudo, Assets, Session · Forkless upgrade path
Proof Lifecycle A-T-C-G · Authorization → Transfer → Certification → Guard · ProofLifecycleFinalized = proof lifecycle closure, NOT block-consensus finality · Lifecycle model, not consensus mechanism	Current Status Pre-mainnet public demo / testnet (Spec-121) · Aura block production · GRANDPA finality · Sudo/Root authority retained · Not yet production-decentralised	

Document Classification

Technical Whitepaper v1.1 · September 2026 · Informational Distribution Only

Intended audiences: infrastructure investors, Web3 grant reviewers, independent validator operators, enterprise integration partners, and protocol engineers evaluating proof-native verification infrastructure.

This document describes protocol architecture, proof lifecycle design, validator mechanics, security boundaries, token economics, and roadmap. It is not a full implementation specification. Exact runtime call names, event names, and storage items are published through runtime metadata and technical release manifests.

Legal Disclaimer

This document is for informational purposes only. It does not constitute legal, financial, tax, or investment advice, an offer to sell securities, or a guarantee of returns. Token purchases carry risk, including the risk of total loss. Helixium is currently pre-mainnet — the public demo and testnet demonstrate technical direction only. The protocol-defined maximum supply of 46,000,000 HLX is a mainnet monetary policy target; permanent runtime enforcement is pending implementation and independent verification. The current testnet retains Sudo/Root and runtime upgrade authority. No exchange listing is guaranteed. No returns are guaranteed. Regulatory treatment of HLX may vary by jurisdiction. Participants should seek independent legal, tax, and financial advice before participating.

Executive Summary

Helixium (HLX) is a proof-native Layer-1 blockchain that introduces a four-stage, role-accountable proof lifecycle as a runtime primitive — not an application-layer add-on. Where conventional blockchains treat verification as a hash-anchoring exercise, Helixium models the full lifecycle of a proof object through a structured A–T–C–G model and a Nukleoid role architecture, making proof state visible, auditable, and portable across systems.

- **Native coin: HLX** — maximum-supply policy target of 46,000,000 HLX; base unit: nucleo (10^{-8} HLX, i.e. 1 HLX = 100,000,000 nucleo); no inflationary issuance policy; permanent supply enforcement is a mainnet requirement pending implementation and independent verification
- **Core innovation:** Proof objects are runtime-visible entities with lifecycle states (Authorization → Transfer → Certification → Guard / ProofLifecycleFinalized), not merely hash records
- **Deployed runtime:** Polkadot SDK (Substrate), Spec-121 testnet; deployed pallets include HelixProof, BlockHelix, HelixValidatorRegistry, HlxTransfer, Aura, Grandpa, Balances, TransactionPayment, Sudo, Assets, and Session. "ProofLifecycle," "NukleoidRoleRegistry," and "GuardChallenge" are logical protocol responsibilities, not necessarily separately deployed pallets under those exact names
- **Consensus:** Aura block production; GRANDPA finality. A–T–C–G is a proof lifecycle model, not a block consensus mechanism. ProofLifecycleFinalized denotes proof lifecycle closure, not block-level consensus finality
- **Role assignment:** Finalized-block-hash-derived assignment beacon, configured through a Root-authorized runtime call. Not described as autonomous BABE VRF. Role assignment sets, quorum parameters, and security profile are versioned runtime configuration under technical verification
- **Validator model:** Dynamic A/T/C/G Nukleoid role assignment per proof; validators are not permanently fixed to a single role; current testnet uses adaptive quorum appropriate for a small validator set
- **Portable proof bundles:** Verifiable via standardised endpoints (/proof/:id/bundle, /proof/:id/verify); offline SDK/CLI verification is a planned Phase 3 capability
- **Transaction fees:** Eligible signed transactions incur a base fee under runtime transaction-payment rules; observed live Spec-121 base fee for HlxTransfer.transfer_keep_alive is 100,000 nucleo (0.001 HLX) plus any optional tip; fees route to the deterministic fee/rewards vault (PalletId "hlx/rewd"); fees are not burned; reward distribution to validators is a separate specification
- **Current status:** Pre-mainnet public demo / testnet (Spec-121); Sudo/Root authority retained; not yet production-decentralised; no independent security audit completed
- **Early Supporter Round:** 920,000 HLX (2% of total supply, included within the 30% Community and Sale allocation) at \$0.25/HLX; implied FDV \$11.5M; eligibility, pricing, lock, and vesting terms will be published before any distribution; no specific schedule is committed in this whitepaper; no exchange listing guaranteed; no returns guaranteed
- **Target audiences:** Infrastructure investors, Web3 grant reviewers, validator operators, enterprise integration partners

Helixium does not claim mainnet decentralisation at this stage. The public demo and testnet prove product direction and technical feasibility. Mainnet requires broader validator participation, independent security audits, economic security mechanisms, governance maturity, and removal of Sudo/Root authority. The current testnet retains centralized operator control.

The Verification Problem

Across healthcare, financial services, legal, supply chain, and AI-generated content industries, verification is treated as a point-in-time event rather than a persistent, auditable lifecycle. The root cause is structural: there is no shared, runtime-native verification layer that makes proof state persistent, role-accountable, and portable across organisational boundaries. The absence of a shared verification layer forces every organisation to solve the same problem independently — resulting in duplicated infrastructure, inconsistent standards, and verification records that do not lend themselves to comparison or audit across organisational boundaries.

Structural Limitations of Conventional Verification



Fragmented Trust Anchors

Credentials issued by a hospital EHR, financial custodian, or logistics platform are not natively verifiable by a counterparty without a direct API integration or bilateral data-sharing agreement — each cross-system validation requires bespoke trust infrastructure negotiated bilaterally. This bilateral integration model creates compounding maintenance burden: when any party in a multi-party workflow upgrades its credentialing system, all bilateral integrations must be renegotiated or rebuilt, leaving verification infrastructure perpetually lagging behind operational reality, with stale integrations producing false-positive results for credentials that have since been superseded or revoked.



Hash Anchoring Limitations

A SHA-256 or Keccak hash anchored at block N proves the document existed in that form at that block — and nothing more. It carries no information about the authorising identity, the transfer chain, the certification criteria applied, or whether the proof has since been superseded, challenged, or invalidated. This gap is particularly acute in long-lived credentials: a professional licence anchored at block N may remain valid for years, during which the holder's status changes, the credential may be transferred, suspended, or revoked — none of which is reflected in the original hash record.



AI Provenance Gap

As large language models, image generators, and automated decision systems produce outputs at scale, regulated industries — financial advice, medical diagnosis, legal drafting — require an auditable chain from model version → inference run → output → downstream use. A single output may be produced by a chain of models — a retrieval system, a reasoning model, a formatting model, and a safety filter — each of which must be traceable for the provenance record to be complete. Existing general-purpose blockchains do not natively model multi-model provenance chains as a portable, independently verifiable runtime record.

The result is a verification landscape where trust is expensive to establish, fragile to maintain, and difficult to audit across organisational boundaries. Helixium addresses this structural gap at the runtime layer — not by replacing cryptographic hashing, but by extending it with a structured, role-accountable, challenge-aware proof lifecycle. The A–T–C–G model addresses each limitation directly: Authorization resolves identity accountability, Transfer resolves custody chain, Certification resolves integrity criteria, Guard resolves policy and challenge, and the portable proof bundle resolves portability.



No Lifecycle Visibility

Point-in-time verification records do not answer six critical questions: (1) which identity authorised the submission, (2) whether the proof was transferred to a new custodian, (3) which party certified its integrity and under what criteria, (4) whether it was challenged after issuance, (5) whether it was revoked or invalidated, and (6) what the current valid state is at the time of query. This asymmetry is exploited in fraud scenarios where a credential is presented after it has been internally revoked or superseded, but before the verifying party has been notified.



Centralised Revocation

X.509 certificate revocation lists (CRLs) and OCSP responders are single points of failure: if the issuing CA is compromised or its revocation infrastructure is unavailable, relying parties cannot determine whether a certificate remains valid. CRLs are typically published on a schedule (hourly, daily, or weekly), meaning a certificate revoked at 09:00 may still be accepted as valid by a relying party that cached the previous CRL at 08:00. OCSP stapling reduces but does not eliminate this window, and neither mechanism provides an on-chain, independently auditable revocation event.



Portability Deficit

Proof records stored in proprietary systems — a notarisation SaaS, a credentialing platform, or a supply chain TMS — require the verifying party to either trust the issuing platform's API response or obtain a raw export, neither of which is independently verifiable without access to the originating system's internal state. Several major credentialing and notarisation platforms have been acquired, pivoted, or shut down in the past five years, leaving customers with proof records that are no longer independently verifiable and requiring costly re-issuance that introduces gaps in the audit trail.

Helixium Overview

Helixium is a proof-native Layer-1 blockchain built on the Polkadot SDK, organised around three logical protocol responsibility domains — **Proof Lifecycle**, **Nukleoid Role Registry**, and **Guard/Challenge** — addressing proof state management, validator role accountability, and challenge resolution. The protocol uses Substrate's forkless runtime upgrade mechanism; the proof API, `/proof/:id/verify` endpoint, and A-T-C-G lifecycle state are live in the public demo environment. The deployed Spec-121 testnet runtime uses Aura for block production and GRANDPA for finality.

- ❗ The three logical domains — Proof Lifecycle, Nukleoid Role Registry, and Guard/Challenge — are conceptual groupings, not necessarily separately deployed pallets. Deployed Spec-121 runtime pallets include: HelixProof, BlockHelix, HelixValidatorRegistry, HlxTransfer, Aura, Grandpa, Balances, TransactionPayment, Sudo, Assets, Session. Exact mappings are published through runtime metadata and technical release manifests.

What Conventional Hash Anchoring Provides

- Cryptographic proof that a document with a given SHA-256 or Keccak hash existed at or before a specific block height, without reliance on a central timestamp authority.
- Modifications produce a different hash, making tampering detectable; the block height provides a consensus-backed timestamp that cannot be retroactively altered.

What Hash Anchoring Cannot Independently Describe

- The identity, public key, permission scope, and policy context of the submitting party — required to determine whether the submission was legitimate.
- The custody chain: which parties received, held, or forwarded the proof, and whether each transfer was authorised and integrity-verified.
- The Certification Nukleoid identity and attestation details.
- Whether the Guard stage evaluated the proof against runtime policy rules and what the decision outcome was (ACCEPT, REJECT, PENDING, or REVIEW).
- The full lifecycle state transition sequence — Pending → Authorized → Transferred → Certified → Finalized or Invalidated — with block references and role audit trail per transition.

The Helixium Extension

Helixium extends hash anchoring rather than replacing it: compact proof receipts and hashes are stored on-chain; raw data is never written to chain state. The **Proof Lifecycle** domain maintains proof state, role assignments, and block references per transition; the **Guard/Challenge** domain implements challenge resolution and quorum evaluation; the **Nukleoid Role Registry** domain records validator roles, capabilities, and per-proof epoch assignments.

Core Properties

- Each proof object is stored on-chain with a unique Proof ID; every A-T-C-G stage transition records the assigned validator's identity, role, attestation signature, and block reference — a per-proof, per-stage audit trail queryable on-chain.
- The `/proof/:id/bundle` endpoint returns a structured JSON object containing the Proof ID, compact receipt, lifecycle state, role audit trail, and block references; the `/proof/:id/verify` endpoint returns current state, validity flag, and role audit summary — both without requiring access to the originating application.
- Only the role-assigned validator for a given stage and epoch may submit an authoritative LifecycleStageSigned attestation; local evaluation produces a Local Verification Receipt, not a network state transition.

What Helixium Is Not

- A-T-C-G defines the four verification stages — Authorization, Transfer, Certification, Guard — at the application and runtime layer; it is not a replacement for the block consensus mechanism.
- ProofLifecycleFinalized denotes proof lifecycle closure, not block-consensus finality; GRANDPA determines block finality independently.
- Helixium is purpose-built for proof lifecycle management and does not provide a general-purpose smart contract execution environment.
- The originating document or payload is never written to chain state; only the compact proof receipt, lifecycle state, role assignments, and block references are persisted.
- The current network operates with a controlled validator set; mainnet decentralisation requires ≥ 5 independent external validators, a completed security audit, active StakeRegistry and slashing hooks, and operational on-chain governance.

Design Principles

Helixium's architectural principles are not design preferences — they are invariants enforced at the runtime layer: the **Proof Lifecycle logical domain** enforces lifecycle ordering, the **Nukleoid Role Registry logical domain** enforces role-scoped attestation authority, and the **Guard/Challenge logical domain** enforces challenge-window sequencing. Any deviation from these principles — such as allowing a validator to attest outside its assigned role, or allowing Guard finalization while a challenge is active — would require an explicit, governance-approved runtime upgrade to the relevant domain logic. These constraints are embedded in the runtime extrinsic logic and are designed to enforce correctness at the runtime layer; they cannot be overridden at the application or API layer.

Lifecycle Over Timestamp

A proof object is not a write-once record. It is a stateful entity that progresses through four ordered stages — Authorization, Transfer, Certification, Guard — with each transition recorded as a distinct runtime event (`LifecycleStageSigned`, `NukleoidRoleVerified`) and stored in the Proof Lifecycle logical domain's on-chain state map. The current lifecycle state is queryable at any time from runtime storage; historical transitions are available through block events. A proof is designed to enforce strict stage ordering at the extrinsic level — it cannot skip stages or transition backwards.

Role Accountability

Every lifecycle stage transition is associated with a specific Nukleoid role (A, T, C, or G) and a specific validator identity, recorded in the Nukleoid Role Registry logical domain. The assigned validator submits a `LifecycleStageSigned` attestation; upon successful stage verification, a `NukleoidRoleVerified` event is emitted. This creates a per-proof, per-stage audit trail: for any proof object, it is possible to query which validator acted at each stage, under which role assignment, at which block height, and with which attestation signature.

Engine Availability ≠ Decision Authority

Every validator in the network runs the complete A–T–C–G evaluation stack — Proof Lifecycle, Nukleoid Role Registry, and Guard/Challenge — locally against every proof. This local evaluation produces a **Local Verification Receipt**, which is a validator-internal artefact and does not constitute a network state transition. Role assignment is derived from a finalized-block-hash-based assignment beacon configured through a Root-authorized runtime call. Only the validator assigned to a specific role for a specific proof epoch may submit an authoritative `LifecycleStageSigned` attestation; submissions outside the assigned role are rejected by the runtime.

Compact On-Chain State

The Proof Lifecycle logical domain stores, per proof object: the Proof ID, compact receipt hash (SHA-256 or Keccak of the original payload), current lifecycle state, assigned Nukleoid role identities, and block references for each state transition. The originating document, payload, or data asset is never written to chain state. This design keeps the per-proof on-chain footprint to a fixed-size record regardless of the size of the underlying document, making the chain state lean, auditable, and scalable without storage bloat.

Portable Verification

Proof bundles are exported via two standardised endpoints: `/proof/:id/bundle` returns a structured JSON object containing the Proof ID, compact receipt, lifecycle state, role audit trail, and block references for each transition; `/proof/:id/verify` returns the current state, a validity flag, and a role audit summary. Both endpoints are callable by any third party without access to the originating application, its internal database, or its UI. A planned Phase 3 Verifier SDK will enable offline bundle verification against a known chain state snapshot, removing the dependency on a live API connection.

Deterministic Security

Guard decisions are produced by a deterministic, modular pipeline of twelve validation modules: Structure Validator, Authorization Verifier, Transfer Integrity Validator, Certification Validator, State Consistency Validator, Policy Engine, Evidence Validator, Challenge State Validator, Quorum Validator, Risk Evaluator, Decision Engine, and Decision Auditor. The Decision Engine aggregates all module outputs into a single deterministic outcome: ACCEPT, REJECT, PENDING, or REVIEW. External observations — from AI risk models, malware scanners, or anomaly detectors — may be submitted as signed evidence to the Guard Core but must not directly determine blockchain state; Guard applies its deterministic rules to those observations and produces the authoritative decision.

Honest Status Boundaries

The protocol distinguishes explicitly between four status categories applied throughout this document: **(A) Verified in the current testnet** — features live in the Spec-121 testnet runtime and exercised in the public demo environment; **(B) Implemented but not independently audited** — logic present in the deployed runtime but not yet subject to external security review; **(C) Planned for mainnet** — capabilities scoped and designed but not yet deployed; **(D) Policy proposed but not yet finalized** — governance frameworks or parameter regimes under active development. These labels are applied consistently to avoid conflating current capability with future intent.

These principles are architectural constraints enforced at the runtime layer. Governance-controlled runtime upgrades are planned as a Phase 4 capability — until that framework is active, any change to these constraints requires a coordinated runtime upgrade by the current network operators. All such changes will be recorded as on-chain events in the upgrade history.

Full Engine Availability and Decision Authority

A foundational invariant of the Helixium protocol is that engine availability does not equal decision authority: every validator in the network may run the complete A–T–C–G engine stack locally, but only the role-assigned validator may issue an authoritative attestation for a given proof and epoch. The assignment is recorded in the NukleoidRoleRegistry at the start of each proof epoch, binding a specific validator public key to a specific role (A, T, C, or G) for that proof's lifecycle. The runtime enforces this binding at the extrinsic level: when a validator submits a LifecycleStageSigned attestation, the runtime checks the NukleoidRoleRegistry to confirm that the submitting account is the assigned validator for that role and epoch — if not, the extrinsic is rejected with a role-mismatch error.

 The current testnet retains Sudo/Root authority. Runtime upgrades, including changes to role assignment logic, can be executed by the Root account without on-chain governance approval. This is a known centralization boundary that will be addressed in Phase 4 governance implementation.

The Engine–Role Distinction

Full Engine Availability	Role-Scoped Authority	Quorum and Configuration
Every validator node runs the complete ProofLifecycle, NukleoidRoleRegistry, and GuardChallenge evaluation stack locally against every proof object it observes. This local evaluation traverses all four A–T–C–G stages — Authorization, Transfer, Certification, Guard — and produces a Local Verification Receipt : a validator-internal data structure recording the local evaluation outcome, the proof state observed, and the validator's own assessment of each stage. The Local Verification Receipt is not broadcast to the network and does not constitute a state transition. It is used internally for validator monitoring, anomaly detection, and pre-attestation consistency checks.	At the start of each proof epoch, the NukleoidRoleRegistry assigns validators to A–T–C–G roles for that proof. Role assignment is derived from a finalized-block-hash-based assignment beacon, configured through a Root-authorized runtime call. This means the current assignment mechanism operates under an operator/Root trust assumption. The exact deployed beacon procedure is under technical verification. This mechanism should not be described as autonomous, unpredictable, unbiased, or immune to operator influence — those properties require a documented threat model and verification evidence that are not yet available. The assigned validator for a given role is the only account authorised to submit a LifecycleStageSigned attestation for that stage and epoch; any other account's submission is rejected with a NukleoidRoleNotAssigned error.	Role assignment sets, quorum requirements, and the overall security profile are versioned runtime configuration. The exact parameters — including the number of validators per role and the threshold of attestations required before a NukleoidRoleVerified event is emitted and proof state advances — are under technical verification and will be published through runtime metadata and technical release manifests. This document does not assert false precision about quorum sizes. If the required quorum is not reached within the configured epoch window, the proof transitions to PendingReview state rather than proceeding with insufficient attestations. Production mainnet quorum and staking requirements are planned for Phase 4 via the StakeRegistry pallet.

Why This Matters

- Full engine availability means that every validator independently computes the expected lifecycle state for every proof it observes; if a submitted attestation is inconsistent with the local evaluation, the validator can flag the discrepancy as a signed observation submitted to the Guard Core for review
- Role-scoped authority means that even a validator with full engine access and a correct local evaluation cannot advance network state without being the assigned validator for that role and epoch — preventing unilateral finalization regardless of local computation correctness
- The combination of full local evaluation and role-scoped attestation authority creates a two-layer integrity check: local evaluation detects inconsistencies before attestation, and runtime role enforcement rejects unauthorised attestations before they can affect chain state
- If the assigned validator for a role is unavailable — due to network partition, node failure, or liveness timeout — the proof epoch may be reassigned by the runtime after the configured timeout; if reassignment also fails to produce a quorum, the proof transitions to PendingReview, preserving the security threshold rather than accepting a reduced-quorum finalization

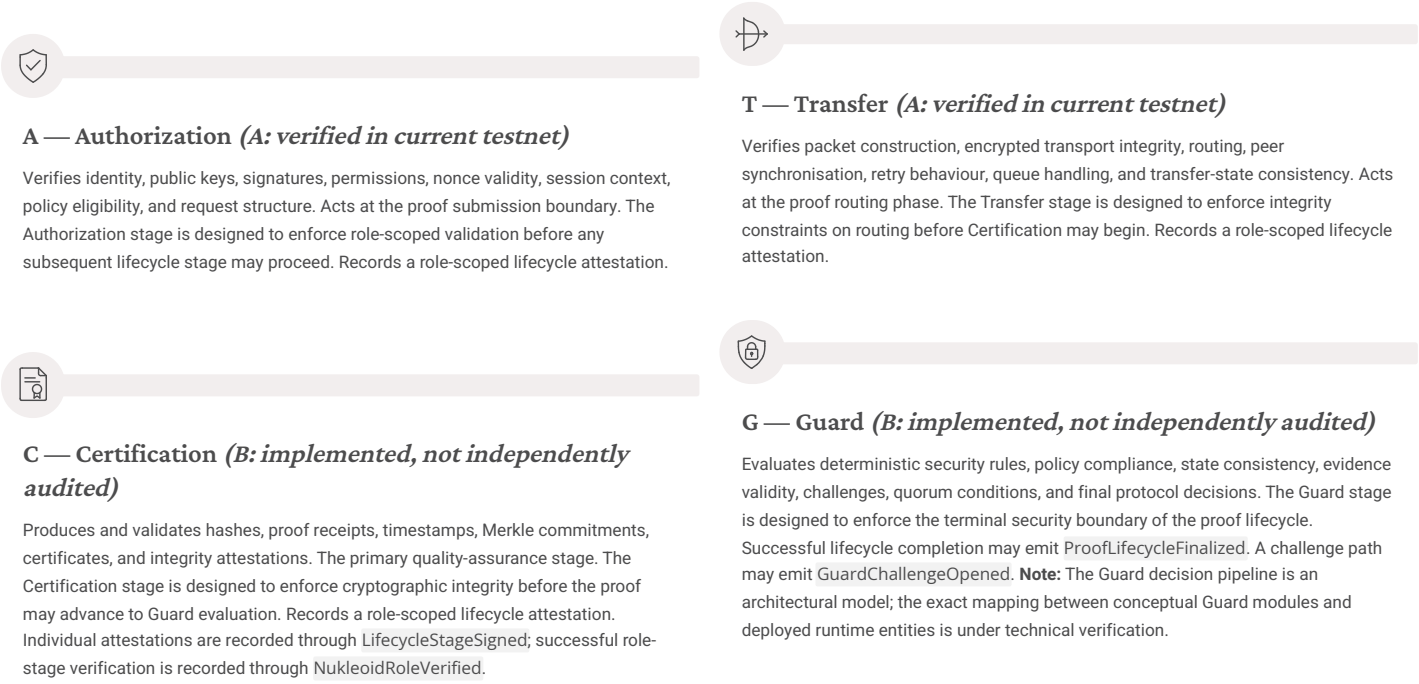
Runtime primitives may exist before their production policy, governance controls, and independent security review are complete. The engine–role distinction is enforced at the extrinsic level in the current runtime implementation; it is not a planned or aspirational constraint. Independent security audit of this enforcement boundary is a planned prerequisite for mainnet launch and is listed as a Phase 4 deliverable.

A-T-C-G Proof Lifecycle

The A-T-C-G model defines four sequential, role-accountable verification stages through which a proof object passes from submission to finalization or invalidation.

Architectural boundary: A-T-C-G is a proof lifecycle model at the application and runtime layer, not a block consensus mechanism. `ProofLifecycleFinalized` denotes proof lifecycle closure, not block-level consensus finality, which is determined by GRANDPA independently.

The Four Lifecycle Stages



Invalidation Path

- After Certification, a signed account with the required challenge bond may open a challenge, suspending Guard finalization until the quorum resolves it.
- If fraud evidence is submitted and validated by the required quorum, the proof transitions to Invalidated state.
- Invalidation is a terminal state; it does not alter prior block history but marks the proof as no longer valid for downstream use.
- Proof records are append-only; invalidation preserves the historical record rather than deleting it.
- Challenge and invalidation paths are implemented to the extent verified by the runtime and tests. Independent audit, production-grade policy, and slashing governance remain pending *(C: planned for mainnet)*.

Lifecycle State Reference

Pending Proof submitted; awaiting Authorization Terminal: No	Authorized Authorization stage complete Terminal: No	Transferred Transfer stage complete Terminal: No	Certified Certification stage complete Terminal: No
Finalized Guard stage complete; lifecycle closed Terminal: Yes	Invalidated Proof invalidated following Guard review Terminal: Yes	PendingReview Awaiting quorum or evidence resolution Terminal: No	

C = Certification throughout this document. Status labels: (A) verified in current testnet · (B) implemented, not independently audited · (C) planned for mainnet.

Validator and Role Architecture

Helixium's validator model is designed around the principle that verification quality requires role specialisation without role rigidity. Validators are participants who register Nukleoid capabilities and are assigned roles dynamically per proof and assignment epoch. The Nukleoid role architecture defines the functional capabilities of validators within the A–T–C–G proof lifecycle.

The Four Nukleoid Roles

A — Authorization Nukleoid

Responsible for verifying identity, public keys, signatures, permissions, nonce validity, session context, policy eligibility, and request structure. Acts at the proof submission boundary.

T — Transfer Nukleoid

Manages packet construction verification, encrypted transport integrity, routing, peer synchronisation, retry behaviour, queue handling, and transfer-state consistency.

C — Certification Nukleoid

Produces and validates hashes, proof receipts, timestamps, Merkle commitments, certificates, and integrity attestations. The primary quality-assurance role in the lifecycle.

G — Guard Nukleoid

Evaluates deterministic security rules, policy compliance, state consistency, evidence validity, challenges, quorum conditions, and final protocol decisions. The enforcement and dispute-resolution role.

Validator Participation Model

- **Registration:** Validators register with the NukleoidRoleRegistry, declaring supported A–T–C–G capabilities.
- **Dynamic assignment:** For each proof, the runtime assigns validators to required roles based on availability, registration, and deterministic assignment logic – not permanent designation. Role assignment is derived from a finalized-block-hash-based assignment beacon, configured through a Root-authorized runtime call. This mechanism operates under an operator/Root trust assumption. The exact deployed beacon procedure is under technical verification. This document does not claim the assignment is autonomous, unpredictable, unbiased, or immune to operator influence.
- **Quorum requirements (A: *adaptive quorum verified in current testnet*):** Role assignment sets, quorum requirements, and security profile are versioned runtime configuration under technical verification. Each lifecycle stage requires the configured quorum of role-assigned validators; the system remains PENDING if quorum cannot be met. This document does not assert false precision about quorum sizes or validator counts per role.
- **Heartbeat monitoring:** Validators emit periodic heartbeat signals; unresponsive validators are flagged and excluded from active assignment pools.
- **Role separation:** Role assignment prefers different eligible validators for A, T, and C whenever sufficient independent validators are available.
- **Guard quorum:** Guard decisions require the configured Guard quorum; a Certification validator must not be able to form the complete Guard quorum alone.

Validator Infrastructure Components

- Validator eligibility registry and bonded validator identity.
- Session-key management and lifecycle-role eligibility declarations.
- Heartbeat and liveness monitoring with deterministic reassignment on timeout.
- Peer-proof verification and assignment epoch management.
- Configurable quorum policies and governance-controlled network parameters.
- StakeRegistry integration and validator admission and removal procedures (C: *planned for mainnet*).
- Economic penalties and slashing hooks (C: *planned for mainnet*).

⚠️ Current testnet operating context: The current testnet operates with a controlled validator set under centralized operator authority. Mainnet decentralisation requires ≥ 5 independent external validators, a completed security audit, active StakeRegistry and slashing hooks, and operational on-chain governance.

Note: Open permissionless validator participation, audited staking, and production slashing are planned protocol capabilities. They are not yet active in the current implementation. Status labels – (A) verified in current testnet, (B) implemented but not independently audited, (C) planned for mainnet – reflect the current state of technical verification and are subject to revision as audit and governance milestones are completed.

Portable Proof Bundles

A core design goal of Helixium is that proof verification must be decoupled from the originating application. Proof bundles are self-contained, exportable packages that carry all information required to verify a proof's lifecycle state — without access to the original UI, issuing system, or raw data.

Bundle Contents

Proof ID Unique runtime-assigned identifier for the proof object.	Compact Proof Receipt Cryptographic hash of the original proof payload; tamper-evident.
Lifecycle State Current state: Pending, Authorized, Transferred, Certified, Finalized, or Invalidated.	Role Audit Trail Ordered record of Nukleoid role actions and validator identities at each lifecycle transition.
Block References Block height and hash for each state transition event; independently verifiable.	Chain Verification Evidence Block references, proof receipts, lifecycle events, and finalized chain state used to verify bundle integrity.

Verifier Endpoints (*A: verified in current testnet*)

Endpoint	Description	Returns
/proof/:id/bundle	Returns the full portable proof bundle via API	Structured JSON object with all bundle contents
/proof/:id/verify	Returns an API-assisted verification result	Current state, validity flag, and role audit summary

Verification Modes

- API-assisted verification (*A: verified in current testnet*):** Proof bundles can be verified through the Helixium proof API (/proof/:id/bundle and /proof/:id/verify). This does not require access to the originating UI or issuing system, but verification results are served by the Helixium API and are not independent of it.
- Offline verification (*C: planned Phase 3 capability*):** The planned Verifier SDK will enable verification of proof bundles against an independently obtained chain state snapshot, without reliance on any Helixium-operated API endpoint. A CLI verifier for scripted and automated proof bundle verification is also planned as part of Phase 3.
- Third-party systems can integrate proof verification via the public API without UI dependency.
- Finalized block history is available from full nodes. Archive-state retention depends on node storage backend configuration and pruning policies; not all nodes retain full historical state.

Privacy and Data Policy

- The originating document or payload is not written to chain state.
- On-chain data includes compact proof receipts, hashes, lifecycle state, role assignments, and block references.
- Metadata, hashes, and references require privacy review before production deployment.
- Proof bundles do not expose raw document payloads to third-party verifiers.
- Verification is possible without access to the original document.

The proof bundle and verify endpoints are part of the Helixium proof API and are verified in the current testnet. The Verifier SDK and CLI verifier are planned Phase 3 deliverables and are not yet active in the current implementation.

Deterministic Role Assignment

Role assignment in Helixium is epoch-bound and proof-specific. In the current Spec-121 testnet, assignment is derived from a finalized-block-hash-based assignment beacon, configured through a Root-authorized runtime call. This mechanism operates under an operator/Root trust assumption. The exact deployed beacon procedure is under technical verification.

 The current assignment mechanism should not be described as autonomous, unpredictable, unbiased, or immune to operator influence. Those properties require a documented threat model and verification evidence that are not yet available. The current testnet retains Sudo/Root authority, which includes the ability to configure the assignment beacon.

Assignment Input Model

Finalized Block Hash
Block hash finalized at the epoch boundary; used as the assignment beacon source. Operator/Root trust assumption applies.

Proof Identifier
Unique runtime-assigned proof ID; ensures assignment is proof-specific.

Validator-Set Hash
Hash of the current eligible validator set; ensures assignment reflects current network state.

Assignment Epoch
Epoch identifier; bounds the validity window of each role assignment.

Role Domain
Separate domain labels (HLX_ROLE_A, HLX_ROLE_T, HLX_ROLE_C, HLX_ROLE_G) ensure independent selection per role.

Role Domain Separation

Domain Label	Role	Purpose
HLX_ROLE_A	Authorization	Selects the validator(s) assigned to the Authorization stage
HLX_ROLE_T	Transfer	Selects the validator(s) assigned to the Transfer stage
HLX_ROLE_C	Certification	Selects the validator(s) assigned to the Certification stage
HLX_ROLE_G	Guard	Selects the validator(s) assigned to the Guard stage

Assignment Properties

- Role separation targets different eligible validators for A, T, and C roles where sufficient independent validators are available (*C: exact parameters under technical verification*)
- Role assignment sets, quorum requirements, and security profile are versioned runtime configuration under technical verification. Exact parameters will be published through runtime metadata and technical release manifests.
- If the required number of eligible independent validators is unavailable, the system remains **PENDING** rather than silently reducing the configured security threshold
- Assignment is recorded on-chain, creating an auditable trail of which validator acted in which capacity for each proof
- **Timeout and reassignment:** if an assigned validator fails to respond within the configured window, the runtime may reassign the role to an eligible alternative

Epoch Management

- Assignment epochs define the validity window for role assignments
- Epoch boundaries are determined by finalized block height and governance-configured epoch length
- Validators must maintain active registration and liveness within each epoch to remain eligible for assignment
- Epoch-bound assignment prevents stale or replayed attestations from prior epochs

The assignment beacon procedure and all role assignment parameters are under technical verification. Security properties, quorum thresholds, and trust model documentation will be published through runtime metadata and technical release manifests as verification progresses.

Guard Security Architecture

Guard is the fourth and final A–T–C–G stage. External observations may be submitted as signed evidence but must not directly determine blockchain state; Guard applies its deterministic rules and produces the authoritative decision.

 The Guard Core described here is an architectural model; exact deployed implementation and event definitions are published through runtime metadata and technical release manifests.

Guard Core Modules (*architectural model; exact deployed implementation under technical verification*)

Reason codes below are architectural examples illustrating intended failure-signalling design; treat them as design intent, not confirmed on-chain events, until verified against Spec-121 runtime metadata.

Structure Validator Validates encoding, field presence, Proof ID format, and receipt hash consistency. Example code: A-101.	Authorization Verifier Re-verifies A-stage LifecycleStageSigned attestation: public key, role assignment, and nonce replay. Example codes: A-101, A-201.	Transfer Integrity Validator Validates transfer-state consistency, routing integrity, and T-stage LifecycleStageSigned signature. Example codes: T-101, T-201.	Certification Validator Re-computes receipt hash against on-chain compact receipt, validates Merkle commitments and C-stage signature. Example codes: C-101, C-201.
State Consistency Validator Enforces A→T→C→G ordering, no skipped or repeated stages, and ascending block references.	Policy Engine Evaluates proof against runtime-configured policy rules; changes require a runtime upgrade (currently via Sudo/Root until Phase 4). Example code: G-101.	Evidence Validator Validates fraud evidence hash, signature, timing, and schema; uses example code G-202 if malformed or out-of-window.	Challenge State Validator Blocks Guard finalization if an unresolved challenge exists (PendingReview). Example code: G-301.
Quorum Validator Enforces required role-assigned attestations for all stages with no epoch replays. Example code: G-201. Quorum logic is runtime-implemented; independent audit is a planned mainnet prerequisite.	Risk Evaluator Aggregates module outputs and signed external observations into LOW/MEDIUM/HIGH/CRITICAL; HIGH or CRITICAL may trigger REVIEW.	Decision Engine Produces a single deterministic outcome – ACCEPT, REJECT, PENDING, or REVIEW. No external system may substitute its own outcome.	Decision Auditor The architectural design calls for an auditable Guard decision trail (outcome, reason information, risk classification, block reference, Guard role identity); which fields are currently persisted on-chain must be confirmed against Spec-121 runtime metadata.

Guard Security Architecture — Continued

Guard Decision Pipeline (*B: architectural model; exact deployed implementation under technical verification*)

Proof Object Ingestion — Accumulated A, T, C attestations and evidence references loaded into Guard pipeline	Structure Validator — Binary encoding, Proof ID format, receipt hash length, field type and presence checks	Authorization Verifier — Re-validates A-stage LifecycleStageSigned signature, public key, nonce, and session context	Transfer Integrity Validator — Confirms custody chain consistency, routing integrity, and T-stage LifecycleStageSigned signature
Certification Validator — Re-computes receipt hash, validates Merkle commitments, timestamp, and C-stage LifecycleStageSigned signature	State Consistency + Policy Engine — Enforces $A \rightarrow T \rightarrow C \rightarrow G$ ordering, no skipped stages, and runtime-configured policy rules	Quorum Validator — Confirms required role-assigned LifecycleStageSigned attestations present for all stages; no epoch replays (<i>runtime-implemented; independent audit is a planned mainnet prerequisite; testnet uses adaptive quorum for small validator set</i>)	Risk Evaluator + Decision Engine — Aggregates module outputs into LOW/MEDIUM/HIGH/CRITICAL; emits ACCEPT, REJECT, PENDING, or REVIEW

Decision Outcomes

Outcome	Meaning	Is a Failure?
ACCEPT	All Guard modules pass; quorum met; no active unresolved challenge; risk classification within configured threshold; ProofLifecycleFinalized emitted	No
REJECT	One or more Guard checks fail under the architectural model, preventing the proof lifecycle transition. Codes such as A-101, C-101, and G-101 are illustrative examples; their actual emission and on-chain representation in Spec-121 require runtime-metadata verification.	No (expected outcome)
PENDING	Quorum threshold not yet reached within the configured epoch window, or challenge window still open; proof transitions to PendingReview; configured quorum threshold is preserved	No (safety state)
REVIEW	Risk Evaluator produces HIGH or CRITICAL classification, or Evidence Validator detects suspicious pattern (G-202); escalated to Guard quorum for manual review	No (safety state)

PENDING and REVIEW are protocol-defined safety states, not error conditions. PENDING preserves the configured quorum threshold rather than proceeding with insufficient attestations. REVIEW triggers Guard quorum escalation rather than silently accepting a suspicious proof.

⚠ The reason codes listed in this registry are part of the Guard architectural model. Exact code names and event definitions are published through runtime metadata and technical release manifests; confirmation against deployed Spec-121 runtime metadata is required to verify which codes are actively emitted.

🔍 External analyzers — including AI risk scoring models, malware signature scanners, anomaly detectors, and reputation registries — may submit cryptographically signed observations to the Guard Core via the External Observation Interface. Each observation must carry a valid Ed25519 or SR25519 signature from a registered analyzer identity. Observations are passed to the Risk Evaluator as untrusted evidence inputs; they inform the risk classification but do not independently determine the Guard decision outcome. The Decision Engine applies deterministic protocol rules to all inputs, including external observations, and produces the authoritative outcome. No external system may write directly to proof lifecycle state. Production deployment, security review, and governance policy for the External Observation Interface are mainnet readiness requirements.

Threat Model and Security Architecture

Helixium is designed to detect, contain, and reject unauthorised, modified, conflicting, or suspicious actions according to protocol-defined rules, covering four attack surfaces: the proof submission boundary (Authorization stage), the custody transfer path (Transfer stage), the integrity attestation layer (Certification stage), and the final policy and quorum evaluation (Guard stage). The protocol's primary defence is deterministic, pallet-level enforcement – not heuristic detection – meaning that a proof that fails any Guard module check is designed to be rejected with a recorded reason code regardless of the submitter's identity or stake.

 The threat model described here reflects the protocol design intent. Independent security audit of the deployed runtime is a planned prerequisite for mainnet. Until that audit is complete, security properties should be treated as design claims, not verified guarantees.

Attack-to-Control Matrix

Threat Vector	Protocol Controls	Designed Outcome
Replay Attack – resubmission of a previously valid proof or transaction to produce a duplicate state transition	Authorization stage nonce validation (A-201: Nonce Reused); transaction identifier uniqueness check in the ProofLifecycle pallet; timestamp validity-window enforcement; on-chain state comparison to detect duplicate Proof IDs	REJECT with reason code A-201; duplicate Proof ID is designed to be rejected at extrinsic level
Double-Spend Attempt – concurrent or sequential submission of conflicting proof state transitions to produce inconsistent ledger state	Authorization stage account and ledger-state validation; ProofLifecycle pallet is designed to enforce a single active lifecycle per Proof ID; block-consensus finality is designed to prevent fork-based double-spend; State Consistency Validator detects conflicting transitions	REJECT; conflicting transition designed to be blocked by pallet state enforcement
Data Tampering – modification of proof payload, receipt hash, or attestation after anchoring	Certification Validator re-computes receipt hash at Guard stage and compares against on-chain compact receipt (C-101: Hash Mismatch on failure); Ed25519/SR25519 digital signatures on all attestations; Merkle commitment validation; transfer-integrity checks at T-stage. Finalized block history is available from full nodes; archive-state retention depends on node storage backend configuration.	REJECT with reason code C-101; tampered receipt hash designed to be detected at Guard Certification Validator
Unauthorised Request – submission of a proof or attestation by an account without the required identity, permission scope, or role assignment	Authorization stage identity and public key verification; NukleoidRoleRegistry role assignment check at extrinsic submission (NukleoidRoleNotAssigned error on mismatch); permission policy evaluation by Policy Engine (G-101: Policy Violation); session context validation. Protocol rules are designed to enforce rejection before state transition.	REJECT with NukleoidRoleNotAssigned or G-101; unauthorised extrinsic designed to be rejected before state transition
Malicious Validator – a role-assigned validator submits a fraudulent, inconsistent, or colluding attestation to advance an invalid proof	Role separation is designed so no single validator controls all four A–T–C–G stages; signed attestations recorded on-chain for auditability; quorum requirement is designed to prevent unilateral finalization; NukleoidRoleRegistry epoch binding limits attack window; heartbeat timeout and deterministic reassignment; fraud evidence preserved on-chain; planned Phase 4 slashing hooks for economic deterrence	Designed to be contained; fraudulent attestation designed to be rejected by quorum; evidence preserved on-chain; planned: ValidatorStakeSlashed
Validator Outage – assigned validator becomes unavailable due to network partition, node failure, or deliberate liveness attack	Heartbeat monitoring via ValidatorHeartbeat events; liveness classification in NukleoidRoleRegistry; deterministic epoch reassignment after configured timeout; quorum-aware PENDING state is designed to prevent finalization with insufficient attestations; PendingReview preserves security threshold	PENDING with reason code G-201; epoch reassignment designed to trigger after timeout; proof transitions to PendingReview if reassignment fails
Sybil Behaviour – an attacker registers multiple validator identities to gain disproportionate role assignment probability or quorum influence	Permissioned validator admission during controlled pre-mainnet phases; bonded identity via ValidatorIdentityRegistered; planned Phase 4 minimum bonded stake requirement via StakeRegistry; eligibility requirements enforced by NukleoidRoleRegistry; planned governance-controlled admission policy	Admission designed to be blocked at registration; planned: stake-weighted admission is designed to prevent cost-free Sybil registration
Conflicting Attestations – two or more validators submit attestations with inconsistent lifecycle state claims for the same proof and epoch	All attestations are signed and recorded on-chain with epoch binding; NukleoidRoleRegistry is designed to enforce a single assigned validator per role per epoch; conflicting attestations are designed to trigger G-202 (Suspicious Evidence Pattern) in Evidence Validator; challenge process allows Guard quorum to evaluate conflicting evidence; full decision audit trail recorded by Decision Auditor	REVIEW with reason code G-202; Guard quorum evaluates conflicting evidence; decision audit trail recorded on-chain

Threat Model and Security Architecture — Continued

Reason-Code Registry

 The reason codes listed in this registry are part of the Guard architectural model. Exact code names and event definitions are published through runtime metadata and technical release manifests; confirmation against deployed Spec-121 runtime metadata is required to verify which codes are actively emitted in the current runtime.

Code	Stage	Description
A-101	Authorization	Invalid Signature — the Ed25519 or SR25519 signature on the submitted proof or attestation does not verify against the submitting account's registered public key; the extrinsic is designed to be rejected before any state transition occurs.
A-102	Authorization	Unauthorised Request — the submitting account does not hold the required permission scope or policy eligibility for the requested proof type or operation; designed to be rejected by the Policy Engine at the Authorization boundary.
A-201	Authorization	Nonce Reused — the nonce included in the proof submission has already been consumed by a prior transaction from the same account; the ProofLifecycle pallet is designed to reject the extrinsic to prevent replay attacks.
A-202	Authorization	Replay Pattern Detected — the Authorization Verifier identifies a combination of matching transaction identifier, timestamp, and account state consistent with a replay attempt; the proof is designed to be rejected and the pattern flagged in the decision audit trail.
T-101	Transfer	Invalid Transfer Packet — the Transfer-stage proof packet fails structural validation: the packet encoding is malformed, required routing fields are absent, or the destination custodian identifier does not match the expected transfer target recorded at Authorization.
T-201	Transfer	Transfer State Conflict — the Transfer Integrity Validator detects an inconsistency between the Authorization-stage output state and the Transfer-stage input state, indicating that the proof may have been modified or rerouted between stages without authorisation.
C-101	Certification	Hash Mismatch — the Certification Validator re-computes the SHA-256 or Keccak receipt hash from the proof payload reference and finds it does not match the compact receipt stored on-chain at anchoring time; indicates post-anchor tampering or payload substitution.
C-201	Certification	Invalid Certificate — the Certification Nukleoid's certificate construction fails validation: the Merkle commitments are inconsistent, the certificate timestamp falls outside the configured validity window, or the certificate format does not conform to the expected schema.
G-101	Guard	Policy Violation — the Policy Engine determines that the proof violates one or more runtime-configured policy rules, which may include proof type restrictions, submitter permission scope, custodian eligibility, jurisdiction-specific flags, or time-window constraints; the proof is designed to be rejected and the violated rule identifier recorded.
G-201	Guard	Quorum Not Reached — the Quorum Validator determines that the number of valid, role-assigned LifecycleStageSigned attestations present for one or more A-T-C-G stages does not meet the configured quorum threshold within the epoch window; the proof is designed to transition to PendingReview rather than proceeding with a reduced-security finalization.
G-202	Guard	Suspicious Evidence Pattern — the Evidence Validator or Risk Evaluator detects a pattern in submitted evidence, external observations, or attestation metadata that is inconsistent with normal operation; the proof is designed to be escalated to REVIEW state for Guard quorum evaluation rather than being automatically accepted or rejected.
G-301	Guard	Challenge Opened — a signed account with the required challenge bond has submitted an open_challenge call after Certification and before Guard finalization; the GuardChallengeOpened, ProofChallengeOpened, and FraudProofSubmitted events are designed to be emitted, and Guard finalization is suspended pending challenge resolution.
G-302	Guard	Proof Invalidated — the Guard quorum has accepted a challenge via accept_challenge votes meeting the configured quorum threshold; GuardChallengeResolved and ProofInvalidated events are designed to be emitted; the proof transitions to the terminal Invalidated state while its historical blockchain record is preserved.

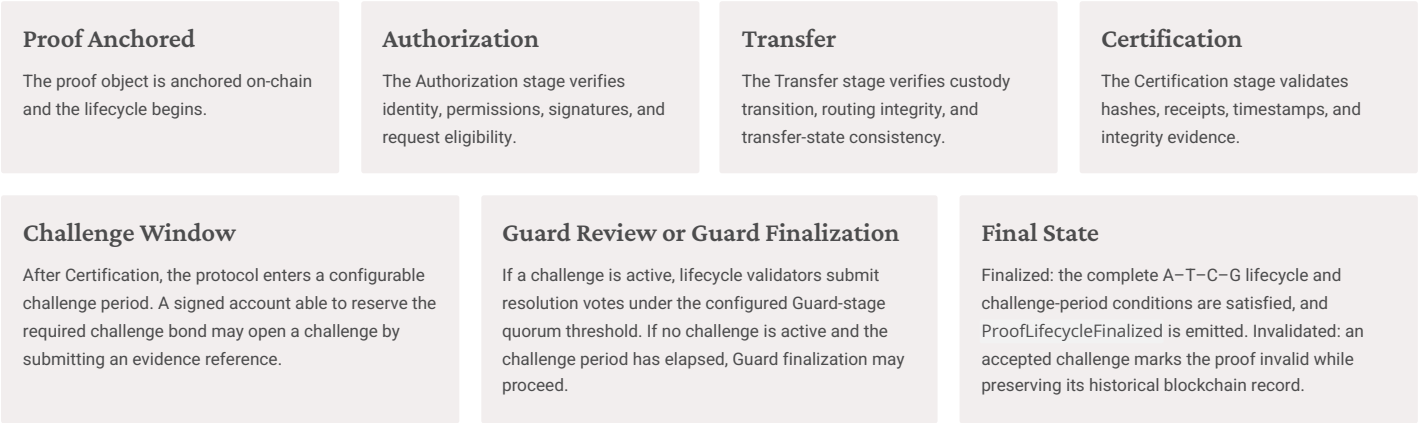
Reason codes are recorded on-chain by the Decision Auditor module as part of the Guard decision trail. Each code maps to a specific module evaluation failure, enabling precise developer diagnostics, enterprise audit reporting, and post-incident analysis. Codes in the A-series are associated with the Authorization Verifier; T-series with the Transfer Integrity Validator; C-series with the Certification Validator; G-series with the Policy Engine, Quorum Validator, Evidence Validator, Challenge State Validator, or Decision Engine. Authoritative code definitions and emission status for the deployed runtime are published through Spec-121 runtime metadata and technical release manifests.

Proof Challenge and Review Lifecycle

The Helixium protocol includes a structured challenge and review mechanism between Certification and Guard finalization. After Certification is verified, the protocol enters a configurable challenge window. A challenge may submit compact evidence references for deterministic on-chain review. Guard finalization is suspended while an active challenge remains unresolved.

 **Implementation Status:** Challenge and invalidation paths are implemented to the extent verified by the runtime and tests **(B)**. Independent audit, production-grade policy, operational hardening, and slashing governance remain pending **(C)**. Guard quorum enforcement is implemented in the runtime. Independent audit of quorum enforcement is a planned prerequisite for mainnet. The current testnet uses adaptive quorum appropriate for a small validator set.

Challenge Lifecycle Stages



Key Properties

- Proof records are append-only; invalidation preserves the historical record rather than deleting it
- During an active challenge, Guard finalization and lifecycle compaction are suspended pending resolution
- Evidence is recorded on-chain as a compact reference; full evidence payloads may be stored off-chain with an on-chain hash commitment
- Invalidation is a terminal state; it does not alter prior block history
- All challenge events, evidence submissions, and Guard decisions are recorded on-chain for auditability

Current Challenge Interface

Runtime Call	Responsibility	Runtime Events
<code>open_challenge</code>	Opens a challenge and records the supplied evidence reference	<code>ProofChallengeOpened</code> , <code>FraudProofSubmitted</code> , <code>GuardChallengeOpened</code>
<code>accept_challenge</code>	Submits a validator vote to accept the challenge	<code>ChallengeResolutionVoteRecorded</code> ; after quorum: <code>ProofChallengeAccepted</code> , <code>GuardChallengeResolved</code> , <code>ProofInvalidated</code>
<code>reject_challenge</code>	Submits a validator vote to reject the challenge	<code>ChallengeResolutionVoteRecorded</code> ; after quorum: <code>ProofChallengeRejected</code> , <code>GuardChallengeResolved</code>

Event emission depends on runtime state and quorum completion. A submitted vote does not independently resolve a challenge.

`open_challenge` may be submitted by a signed account able to reserve the required challenge bond and submit an evidence reference. Challenge-resolution votes are submitted by active lifecycle validators. The configured Guard-stage quorum threshold determines whether the challenge is accepted or rejected.

Blockchain State and Resilience

Each full validator in the Helixium network maintains a synchronised copy of blockchain state and finalized history. The replicated ledger model ensures that no single validator failure can erase finalized proof records. Security still depends on honest quorum assumptions and validator-set governance — compromising one validator does not automatically rewrite finalized network history, but the integrity of the network requires that a sufficient proportion of validators remain honest and operational.

 **Testnet Status:** The current testnet operates with a controlled validator set under centralized operator authority (Sudo/Root retained). The resilience properties described here assume honest validator behavior and are subject to the applicable consensus and honest-validator assumptions.

On-Chain State Components

- Proof receipts, hashes, and lifecycle state for all proof objects
- Role assignment records and signed attestations for each proof epoch
- Challenge events, evidence digests, and Guard decision audit trails
- Validator registration, eligibility status, and heartbeat records
- Planned following governance activation: governance proposals, governed parameter changes, and treasury allocation records.
- Block references and event logs for all lifecycle transitions

Data Resilience Architecture

Replicated Ledger State Every full validator maintains a complete, synchronised copy of finalized blockchain state. State divergence is detected and resolved by the consensus protocol.	Finalized Block History Finalized block history is available from full nodes. Archive-state retention depends on node storage backend configuration and pruning policies; not all nodes retain full historical state. Proof records anchored in finalized blocks cannot be altered without rewriting the chain from that point — which requires overcoming the consensus mechanism.	Genesis and Chain-Spec Consistency All validators operate from the same genesis configuration and runtime specification. Binary and runtime integrity manifests are published for independent verification.
Archive Nodes and Snapshots Archive nodes maintain full historical state where configured to do so. Snapshot strategies enable new validators to sync efficiently without replaying the full chain from genesis.		Validator Recovery Validator removal and replacement procedures ensure that the network can recover from individual validator failures without loss of finalized state.

On-Chain Storage Policy

- The protocol is designed to minimise on-chain data: originating documents and raw payloads are not stored on-chain. Proof receipts, hashes, lifecycle state, role assignments, block references, and other protocol state (such as account balances, validator registrations, and governance records) are stored on-chain as required by the runtime.
- This policy keeps chain state lean, auditable, and scalable
- The originating document or data payload remains with the issuing party
- Proof bundles contain receipts and hashes only; they do not expose raw data

Security Assumptions

- Network integrity requires that a sufficient proportion of validators remain honest and operational
- Compromising one validator does not automatically rewrite finalized network history
- GRANDPA finality provides strong safety guarantees under the applicable honest-validator assumptions. Block finality does not eliminate all operational failure scenarios; the threat model section describes applicable controls.
- Economic security mechanisms (planned: bonded stake, slashing) are designed to increase the cost of validator misbehaviour
- Independent security audits are a planned prerequisite for mainnet launch

HLX Token and Tokenomics

HLX is the native coin of the Helixium network. The token model is designed for long-term network sustainability, validator incentive alignment, and ecosystem growth — without inflationary issuance, hidden minting, or speculative tokenomics. The figures below reflect the mainnet planning allocation. Allocations are a mainnet plan, not proof that separate vaults, multisig, on-chain vesting, automated grants, or immutable governance are already implemented.

Supply Parameters



Supply Cap

46,000,000 HLX — mainnet monetary policy target. Permanent enforcement requires Sudo/Root removal, supply-cap pallet, and independent audit.



Base Unit: nucleo

1 HLX = 100,000,000 nucleo · 1 nucleo = 10⁻⁸ HLX · Eight-decimal precision · All balances denominated in nucleo at runtime.



Total Nucleo

4,600,000,000,000,000 nucleo maximum — equivalent to 46,000,000 HLX at full supply issuance.



No Inflation

No inflationary issuance policy. Current testnet retains Sudo/Root authority; immutable supply lock is not claimed for the current testnet.



Observed Supply

46,000,000 HLX observed at a specific finalized Spec-121 testnet block — a point-in-time measurement, not proof of an irreversible cap.

Token Utility



Transaction Fees

TransactionPayment pallet; observed base fee 100,000 nucleo (0.001 HLX) for HlxTransfer.transfer_keep_alive plus optional tip; other extrinsics use weight-based calculation.



Fee Routing

Fees route to hlx/rewd vault (jT1iB9tgmrndnbsvqHvdxCczNB0Ly5mjwyseUgk1LAg73mQ UPt); NOT burned; total issuance unchanged in all tested transactions.



Reward Distribution

Accumulation in the vault does not constitute an automatic validator payout. Eligibility, calculation, and authorization require a separate specification before mainnet.



Proof Anchoring

Each A–T–C–G stage transition consumes HLX fees proportional to extrinsic weight, payable by the submitting account.



Validator Bonding

Validators will bond HLX via the planned StakeRegistry pallet. Minimum stake, slashing, and unbonding periods are Phase 4 deliverables.



Governance

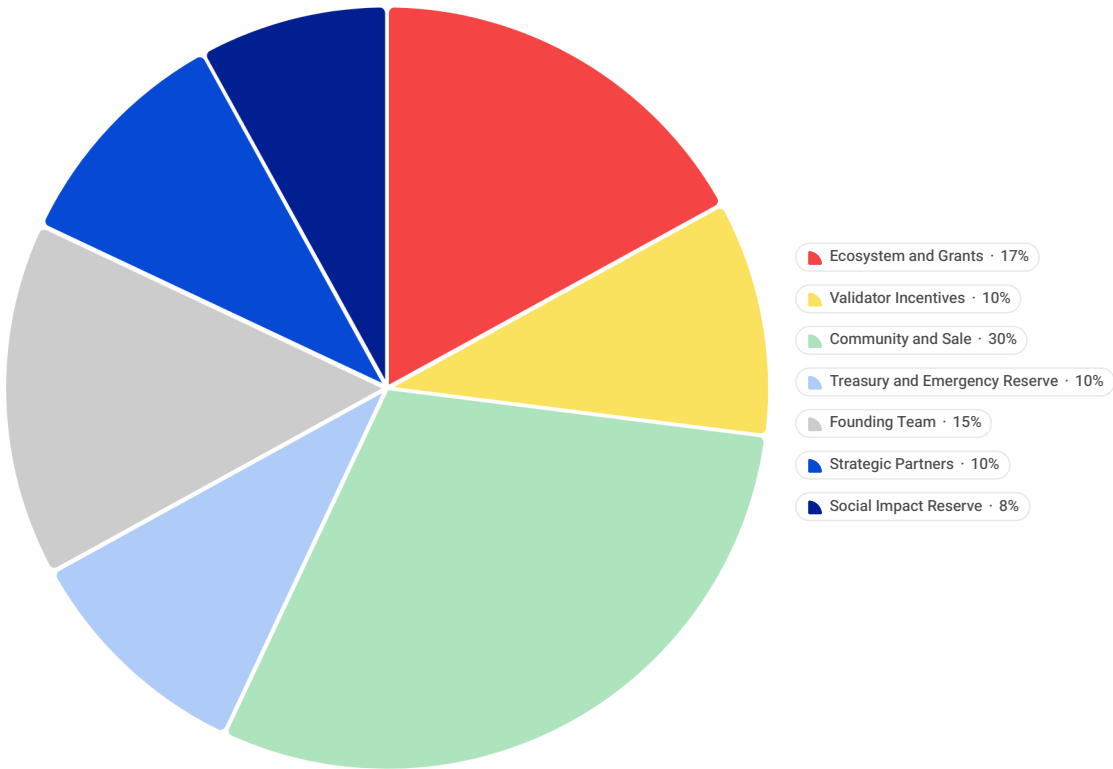
HLX holders will vote on runtime parameters, treasury allocation, and validator admission. Planned Phase 4; replaces current Sudo/Root mechanism.



Ecosystem Incentives

Developer grants and bounties funded from 17% Ecosystem allocation (7,820,000 HLX) via milestone-based grant criteria.

Mainnet Planning Allocation



HLX Supply Allocation — 46,000,000 HLX

Category	Share	HLX Amount	Notes
Ecosystem and Grants	17%	7,820,000	Released via grant milestones; criteria published before distribution
Validator Incentives	10%	4,600,000	Released per network participation; reward eligibility defined before payout
Community and Sale	30%	13,800,000	Includes Early Supporter Round (920,000 HLX / 2%); per sale terms
Treasury and Emergency Reserve	10%	4,600,000	On-chain governance-controlled release planned for Phase 4; not yet active on current testnet
Founding Team	15%	6,900,000	Initial lock + gradual vesting; exact schedule published before distribution
Strategic Partners	10%	4,600,000	Per measurable contribution; initial lock + gradual vesting per agreement
Social Impact Reserve	8%	3,680,000	Helixium +1 (5.6% / 2,576,000), Academy (1.4% / 644,000), Green-Computing (1% / 460,000)
Total	100%	46,000,000	

Arithmetic verification: 17 + 10 + 30 + 10 + 15 + 10 + 8 = 100%. The seven amounts total 46,000,000 HLX.

This allocation supersedes previous planning versions; version history is published separately.

This allocation framework is presented for protocol and ecosystem planning. It becomes authoritative only when published in a separately versioned token-distribution and governance policy with a reproducible integrity hash. If any conflict exists between this whitepaper and the formally adopted token-distribution policy, the latest publicly versioned and integrity-verified policy governs. No exchange listing is guaranteed. No financial return is guaranteed.

Transaction Fees and Rewards Vault

The following describes the fee model as observed in the live Spec-121 testnet, with honest status boundaries applied throughout.

Fee Model

- Eligible signed transactions incur a base fee plus an optional tip attached by the submitter; holding HLX alone does not cause a fee deduction
- The observed 0.001 HLX base fee applies to the native `HlxTransfer.transfer_keep_alive` path; other calls follow their applicable fee calculations

Fee/Rewards Vault

- In observed live Spec-121 testnet transactions, base fee and tip went to the deterministic fee/rewards vault derived from PalletId "hlx/rewd"
- Vault address: `JT1iB9tgmrndnbsvqHvdxCczNB0Ly5mjwyseUgk1LAg73mQUpt`
- In tested transactions, the vault increased by the charged amounts while total issuance remained unchanged; base fees and tips were credited to the vault rather than burned
- Fees do not generate new tokens or guaranteed validator yield

Reward Distribution — Separate Specification

⚠️ Collection by the vault does NOT mean funds are automatically distributed to validators. Reward eligibility, payout calculation, authorization, and payout rules have not yet been defined or implemented; they require a separate specification before mainnet. Refund and failed-dispatch paths require further acceptance testing before their behavior is presented as fully verified.

For clarity, these three facts are distinct:

- **(A) Fee routing to hlx/rewd vault:** verified in the Spec-121 testnet — base fees and tips are routed to the deterministic vault at the address above, with total issuance unchanged
- Accumulation in the vault does not constitute an automatic validator payout.
- **(C) Reward distribution rules:** reward eligibility, payout calculation, authorization, and payout controls have not yet been defined or implemented and remain planned for a separate specification to be published before mainnet

Fee Path Status

Status legend: (A) = verified in current Spec-121 testnet; (B) = implemented but requires further verification; (C) = planned for mainnet.

Fee Path	Status	Notes
HlxTransfer.transfer_keep_alive base fee	(A) Verified in Spec-121 testnet	Base fee of 100,000 nucleo (0.001 HLX) observed in live transactions
Optional tip routing to hlx/rewd vault	(A) Verified in Spec-121 testnet	Vault balance confirmed to increase by the tip amount
Total issuance unchanged after fee	(A) Verified in Spec-121 testnet	Fee routing does not mint new HLX or alter total supply
Automatic validator reward distribution	(C) Planned	Governed by a separate specification to be defined before mainnet
Refund and failed-dispatch fee paths	(B) Requires further verification	Behavior not fully verified; acceptance testing is ongoing
Other runtime call fee calculations	(B) Requires further verification	Fee calculations vary by call type; not all calls use the transfer base fee

Validator Roles and Incentive Design

Helixium separates block-consensus duties from A–T–C–G proof-lifecycle duties. Validators continue to operate network nodes, participate in Aura block production and GRANDPA finality according to the active authority set, and perform proof-lifecycle evaluations and attestations for which they are eligible and assigned. The runtime checks role authority and lifecycle conditions; it does not replace validators or remove their ability to affect network availability, transaction inclusion, or consensus behavior.

Planned Incentive Model

The planned incentive model aims to make validator participation more predictable through published eligibility, service-quality, and payout rules. The initial validator allocation is 4,600,000 HLX, representing 10% of the planned supply. Transaction fees and optional tips collected in the hlz/rewd rewards vault are a separate, usage-dependent source of funds. A fixed weekly or monthly payment, minimum income, APY, or automatic distribution is not currently defined or guaranteed. Any future base stipend or performance-linked payment must be specified, funded, implemented, tested, and publicly reported before being described as an active protocol rule.

Scope and Limitations of Role-Scoped Attestations

Role-scoped attestations and auditable decisions are intended to reduce unilateral control over proof-lifecycle outcomes. They do not eliminate transaction censorship, block manipulation, collusion, or MEV risk. These risks require a separate threat model and controls covering block production, transaction ordering, independent validator participation, monitoring, governance, and economic accountability.

Performance Objectives

An 800 ms block interval is a future performance objective. This incentive design alone does not prove that the target is safe or achievable. Performance targets will be validated through benchmarking, independent review, and testnet observation before being presented as production commitments.

What This Design Does Not Claim

- Validators are not merely passive server providers; they retain full consensus authority over block production and transaction inclusion
- The A–T–C–G engine does not replace or override validators' consensus votes
- No guaranteed monthly or annual HLX income is defined for validators at this stage
- Helixium does not claim to have eliminated MEV, transaction censorship, bribery, or validator collusion risk
- These risks require dedicated controls, governance, and economic accountability mechanisms beyond the proof-lifecycle model

☐ **Status:** The incentive model described here is a planned framework, planned for mainnet. Payout rules, eligibility criteria, and distribution mechanisms will be published in a separate specification before any distribution event.

Allocation Detail and Governance Framework

The following explanatory notes govern interpretation of the mainnet planning allocation table. All allocations are a mainnet plan; not proof that separate vaults, multisig, on-chain vesting, automated grants, or immutable governance are already implemented.

1. Ecosystem and Grants — 17% / 7,820,000 HLX

Scope

Funds wallets, block explorers, open-source developer tools, independent security audits, third-party integrations, and ecosystem applications that extend the utility of the Helixium network.

Governance Before Distribution

Grant criteria, deliverables, milestones, recipients, and amounts will be documented and made publicly available **before** any distribution event. Unused funds remain in the category unless a documented amendment is formally approved.

No Implied Recipients

No organization is implied to be an already-confirmed recipient. Naming a use case or category does not confer HLX to any specific party.

2. Validator Incentives — 10% / 4,600,000 HLX

Purpose and Budget

An initial budget of **4,600,000 HLX** to supplement validator economics, particularly important during early network operation when transaction volume and fee revenue may be low.

Payout Conditions

Reward eligibility, availability and performance standards, payout frequency, and consequences for non-compliance will be fully defined in a separate specification **before** any payout. Payout controls and authorization are a separate deliverable.

Fee Revenue Is Separate

Fee routing is implemented in the Spec-121 testnet: eligible fees and tips are directed to the `hlx/rewd` vault. Accumulation does not constitute an automatic validator payout. Reward eligibility, calculation, authorization and payout rules remain to be specified before mainnet.

3. Community and Sale — 30% / 13,800,000 HLX

Disclosure Before Each Program

Eligibility, limits, pricing or reward rules, delivery, vesting, and treatment of unsold tokens will be disclosed in full before each sale or community distribution program is opened.

Early Supporter Round

The Early Supporter Round of 920,000 HLX (2% of total supply) is included within this 30% allocation — it is not additional supply beyond the 46,000,000 HLX cap. Eligibility, pricing, lock, and vesting terms will be published before any distribution; no specific schedule is committed in this whitepaper.

No Guarantees

No exchange listing is guaranteed. No financial return is guaranteed. This allocation does not constitute a securities offering or investment solicitation.

4. Treasury and Emergency Reserve — 10% / 4,600,000 HLX

Permitted Uses

Funds long-term network operations, independent security reviews, critical infrastructure, and documented urgent needs that meet defined emergency criteria.

Published Authorities

Spending authorities, approval thresholds, ordinary vs. emergency procedures, conflict-of-interest rules, and reporting requirements will be published before any use.

Governance Timeline

On-chain governance-controlled release is planned as a Phase 4 deliverable. Until mainnet governance is implemented, the current testnet retains Sudo/Root authority; the treasury is not yet under immutable on-chain governance control.

Allocation Detail — Team, Partners, and Social Impact

This continues the allocation governance framework for sections 5–7.

5. FOUNDING TEAM — 15% / 6,900,000 HLX

Lock and Vesting The 6,900,000 HLX founding team allocation WILL have an initial lock period followed by gradual vesting. Tokens will NOT all become freely transferable at launch.	Details Published Before Distribution Exact lock dates, vesting durations, wallet addresses, release amounts, eligibility conditions, and treatment of unvested amounts if a team member departs will be published before any distribution.
--	--

6. STRATEGIC PARTNERS — 10% / 4,600,000 HLX

Qualifying Contributions Reserved for measurable technical, operational, integration, or distribution contributions. Merely naming a partner or category does not automatically confer HLX to any party.	Per-Agreement Terms Each agreement will define a ceiling, deliverables, milestone schedule, an initial lock, and gradual vesting. No partners are named or implied to be already confirmed.
---	--

7. SOCIAL IMPACT RESERVE — 8% / 3,680,000 HLX

The Social Impact Reserve is divided into three sub-allocations, each with distinct purposes and governance requirements.

7a: Helixium +1 — 5.6% / 2,576,000 HLX

A global program supporting people with Down syndrome across employment, research, and education. The symbolic derivation "4.6% + 1% = 5.6%" references the 46,000,000 HLX supply and the aim of opening one more meaningful opportunity. A–T–C–G is inspired by the four DNA letters but is not a biological process or scientific security guarantee. Genetic forms of Down syndrome differ. Helixium +1 is a global program, not limited to any single country.

Employment and Independent Living 1,288,000 HLX — Supporting workplace inclusion, independent living programs, and economic participation for people with Down syndrome globally.	Genetics and Chromosome Research 858,667 HLX — Funding applied research into genetic and chromosomal science relevant to Down syndrome, in partnership with qualified research institutions.	Accessible Education and Digital Tools 429,333 HLX — Developing accessible, multilingual educational resources and digital tools designed with and for people with Down syndrome.
--	---	--

Arithmetic note: 1,288,000 + 858,667 + 429,333 = 2,576,000 HLX ✓

☐ **Helixium +1 Governance Requirements:** Applications under published criteria; accessible and multilingual submission routes; participation of people with Down syndrome and representative organizations in design and assessment; milestone-based payments where appropriate; outcome reporting without disclosing beneficiaries' identities or health data on a public blockchain. Cross-border payment, tax, and data-protection requirements will be assessed before disbursement. No organization is implied to be an already-confirmed recipient.

7b: Helixium Academy — 1.4% / 644,000 HLX

Funds free, accessible technical education for developers, validators, and community members. Curriculum, eligibility, and distribution terms will be published before use.

7c: Green-Computing — 1% / 460,000 HLX

Verifiable incentives for measurable energy-efficiency improvements in Helixium network infrastructure. No carbon-neutrality claims without supporting methodology. Eligibility published before disbursement.

Social Impact arithmetic: 2,576,000 + 644,000 + 460,000 = 3,680,000 HLX ✓

Implementation and Transparency

All allocations are a mainnet plan. They are not proof that separate vaults, multisig wallets, on-chain vesting, automated grant disbursement, or immutable governance are already implemented on the current testnet.

1 Plan vs. Implementation

All allocations are a mainnet plan. They are not proof that separate vaults, multisig wallets, on-chain vesting, automated grant disbursement, or immutable governance are already implemented on the current testnet. The absence of on-chain vesting does not mean vesting will not be enforced; it means enforcement mechanisms are a mainnet deliverable. Until wallet addresses, multisig configurations, and vesting contract addresses are published and independently verifiable, all allocation figures represent intended distribution policy, not executed transfers. Participants should not treat the allocation table as evidence that any HLX has been set aside, locked, or reserved in any on-chain account.

2 Published Before Distribution

Wallet addresses, lock schedules, spending authorities, approval thresholds, and reporting procedures will be published in a separately versioned token-distribution and governance policy before any distribution event occurs. Publication will occur through official Helixium communication channels with a reproducible document integrity hash. Each publication will identify: the allocation category, the total HLX amount, the wallet address or multisig configuration, the lock start date, the vesting schedule (if applicable), the authority or signatories required to authorize a release, and the reporting cadence for post-distribution transparency. If any of these fields cannot be confirmed at the time of publication, the distribution event will not proceed.

3 Amendment Procedure

Future changes to any allocation or governance parameter require a dated public amendment that preserves all prior versions in the version history. An amendment must identify: the prior version it supersedes by date and integrity hash, the specific allocation or parameter being changed, the rationale for the change, and the authority that approved it. Amendments that increase any individual allocation category or reduce any lock or vesting period require a higher approval threshold than routine operational amendments. No retroactive amendment may alter the terms of a distribution that has already occurred.

4 Arithmetic Verification

Social Impact Reserve: $2,576,000 + 644,000 + 460,000 = 3,680,000$ HLX ✓ | Helixium +1 sub-allocations: $1,288,000 + 858,667 + 429,333 = 2,576,000$ HLX ✓ | Total supply allocation: $17 + 10 + 30 + 10 + 15 + 10 + 8 = 100\%$ / $46,000,000$ HLX ✓

These figures are cross-checked against the mainnet planning allocation table in Section 15. Any discrepancy between this section and the allocation table should be treated as a drafting error; the allocation table governs. Readers are encouraged to independently verify all arithmetic before relying on these figures for any purpose.

This section describes the allocation framework and governance requirements for the mainnet planning allocation. It does not constitute an offer to sell tokens, a securities offering, an investment solicitation, or a guarantee of financial return. All participation terms, grant criteria, and distribution schedules are subject to applicable law and will be published separately prior to any relevant distribution or participation event.

Use Cases

Helixium's proof-native lifecycle architecture is applicable across any domain where verification must be persistent, auditable, portable, and resistant to post-hoc manipulation. The following use cases represent primary target verticals for ecosystem development. They are presented as architectural applications of the A–T–C–G lifecycle model, not as confirmed deployments or partnerships.

Primary Use Case Verticals



Document Provenance and Certification

Notarisation, contract execution, and official document certification with role-accountable validation, tamper-evident lifecycle history, and portable proof bundles.



AI-Generated Content Provenance

Attach verifiable, portable provenance records to AI model outputs, training data lineage, and decision trails – enabling regulated industries to audit AI-generated content with structured lifecycle evidence.



Legal and Institutional Evidence

Healthcare, financial services, and legal sectors can anchor compliance records with full lifecycle audit trails, challenge mechanisms, and portable verification for regulators.



Certificates and Credentials

Issue, transfer, and archive academic qualifications and professional credentials with portable proof bundles that employers and institutions can verify independently.



Software Build Attestations

Anchor software build artefacts, dependency manifests, and release signatures with A–T–C–G lifecycle tracking for supply-chain security and compliance.



Supply Chain and Logistics

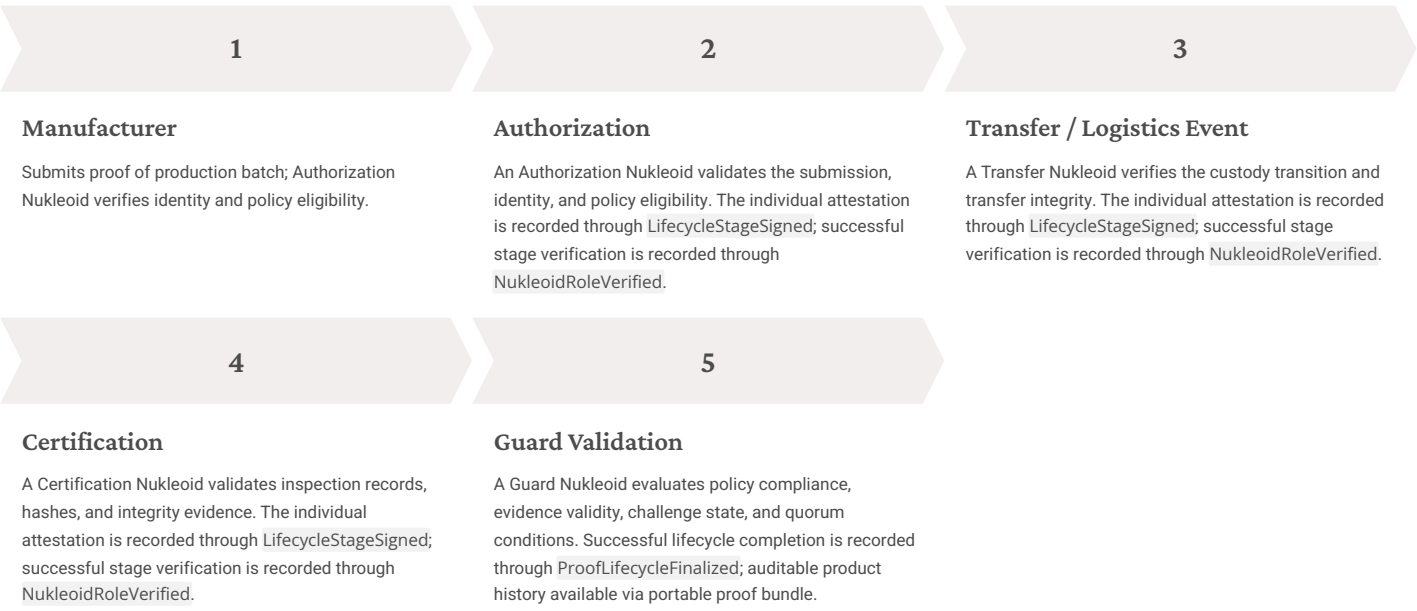
Anchor and lifecycle-track certificates of origin, inspection records, and custody transfers across multi-party supply chains without centralised intermediaries.



IoT Evidence and Regulated Data

IoT sensor readings, maintenance records, and regulated data workflows anchored with verifiable provenance and challenge-aware lifecycle management.

Supply Chain Lifecycle Example



Each use case benefits from the same core properties: lifecycle visibility, role accountability, portable bundles, and challenge/invalidation mechanisms that hash anchoring alone does not natively provide.

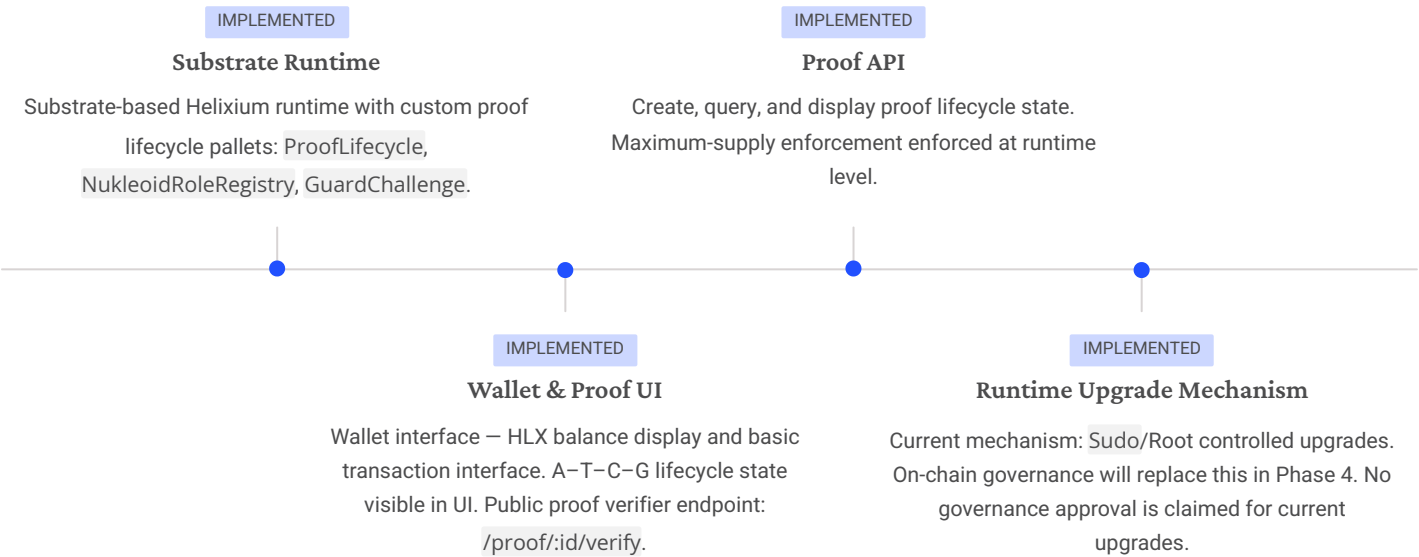
Network Evolution and Roadmap

Helixium is designed to expand from an initial controlled validator set to broader independent validator participation without changing the proof lifecycle model. The roadmap is structured in five phases, each with defined deliverables and honest status boundaries. The team commits to transparent public communication of progress against each phase.

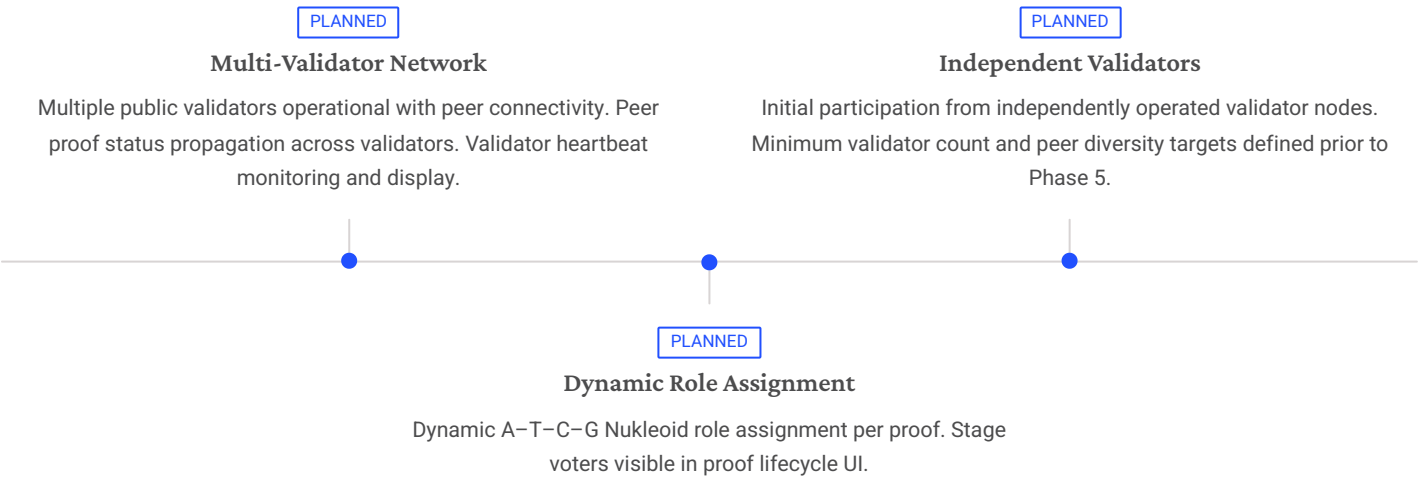
**Current Upgrade Mechanism — Honest Status Boundary**

The current testnet uses Sudo/Root for runtime upgrades. On-chain governance is **NOT** the current mechanism for runtime upgrades. The planned future governance model (Phase 4) will replace Sudo/Root with on-chain governance once the relevant pallets are audited and deployed.

Phase 1 — Protocol Foundation

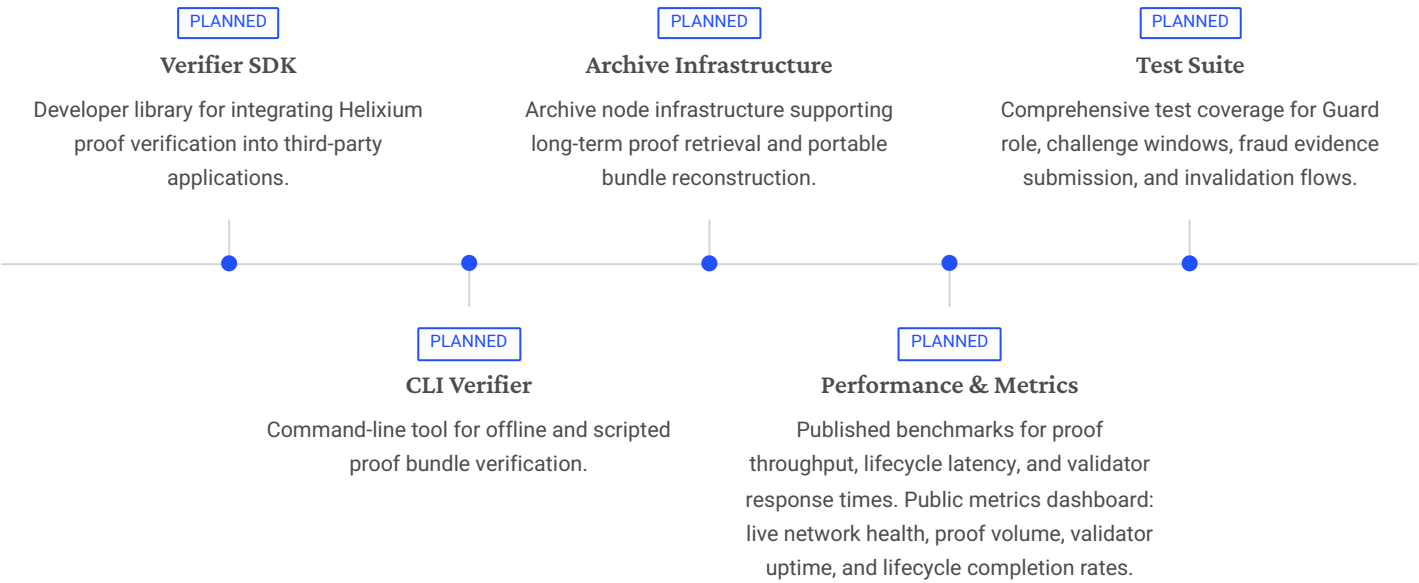


Phase 2 — Multi-Validator Network Validation

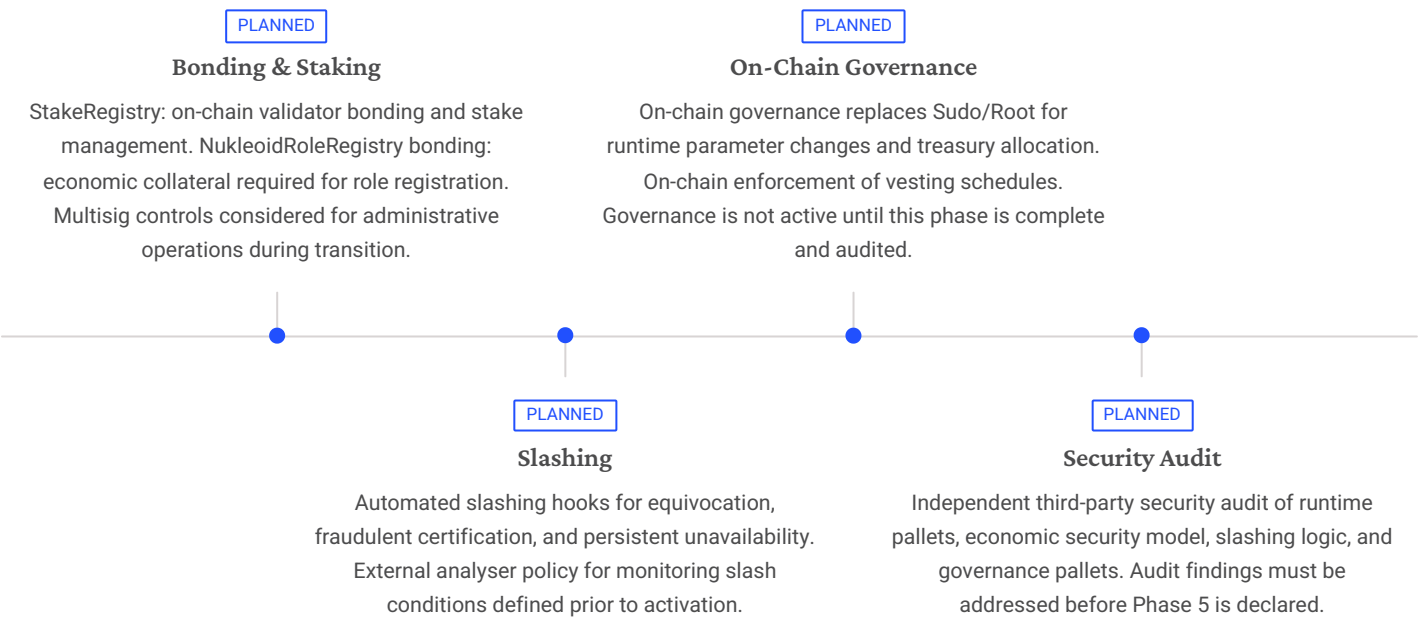


Roadmap — Phase 3 to Mainnet

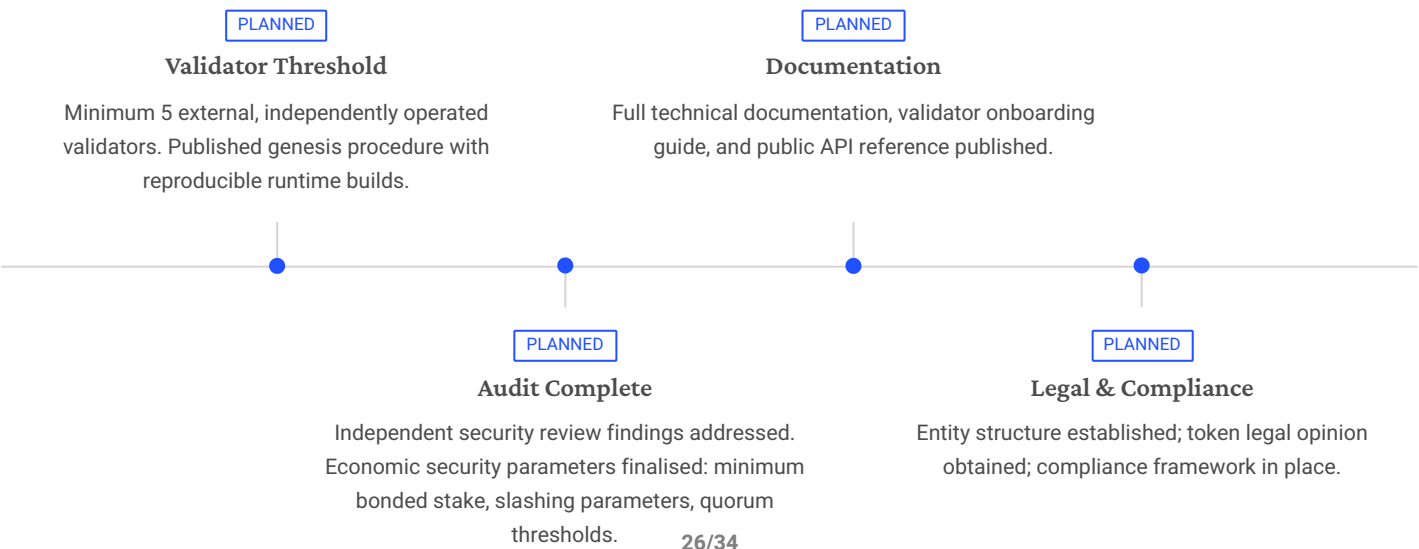
Phase 3 — Developer Tooling and Hardening



Phase 4 — Audited Economic Security



Phase 5 — Mainnet Candidate



Security Boundaries and Limitations

Helixium is committed to transparent communication about its current status, architectural limitations, and the risks associated with participation in an early-stage blockchain protocol. The following disclosures are provided in good faith and should be read carefully by all participants.

⊗ Status Boundary

Helixium is pre-mainnet. The public demo and testnet demonstrate technical direction and architectural feasibility. They are not production infrastructure. Do not treat participation in the Early Supporter Round as a guaranteed financial return. All forward-looking statements are subject to change.

Centralization and Upgrade Boundaries

- The current testnet retains Sudo/Root authority. Runtime upgrades can be executed by the Root account without on-chain governance approval. This is a known centralization boundary.
- The current validator set is not externally independent or fully decentralised; this is a planned capability, not a current state.
- Slashing and economic security mechanisms are planned protocol capabilities; they are not yet active.
- On-chain governance is planned for Phase 4 and will replace Sudo/Root once the relevant pallets are audited and deployed.

Architectural Boundaries

- A-T-C-G is a proof lifecycle architecture, not a block consensus mechanism. Block consensus and proof lifecycle finalization are separate security boundaries.
- A-T-C-G does not replace block consensus — proof lifecycle finality is distinct from block consensus finality. ProofLifecycleFinalized denotes proof lifecycle closure, not block-level consensus finality.
- Raw source documents are not intended to be stored on-chain. The chain may persist compact proof receipts, hashes, lifecycle records, account balances, role assignments, attestations and other protocol state. The exact data fields and privacy boundaries must be verified against the deployed runtime and published technical specifications.

Technical Risks

- Runtime bugs or vulnerabilities may be discovered during audit or expanded validator operation.
- Guard quorum enforcement is implemented in the runtime. Independent audit of quorum enforcement is a planned prerequisite for mainnet.
- The challenge and invalidation paths are implemented in the runtime. Independent audit, production-grade quorum policy, automated slashing governance, and full operational hardening remain pending. Implementation in the runtime does not by itself imply that the mechanism has completed independent security review or production hardening.

Economic and Regulatory Risks

- Token purchases carry risk, including the risk of total loss; HLX is not a guaranteed financial return instrument.
- No exchange listing is guaranteed; secondary market liquidity is not assured.
- Regulatory treatment of HLX may vary by jurisdiction; participants should seek independent legal advice.
- Development timelines may change due to technical, regulatory, or resource constraints.
- Detailed vesting schedules and governance parameters are not yet finalised; they will be published before the relevant distribution events.

What Helixium Does Not Claim

- Helixium does not claim that its protocol is immune to attack or fully secure.
- Helixium does not claim mainnet decentralisation at this stage.
- Helixium does not claim confirmed partnerships, independent audits, exchange listings, or regulatory approvals.
- Helixium does not claim that A-T-C-G replaces block consensus.
- Helixium does not claim that ProofLifecycleFinalized means block-consensus finality.

Conclusion

Helixium represents a principled architectural response to a structural problem in digital infrastructure: verification has been treated as a point-in-time event when it needs to be a persistent, auditable, portable, and challenge-aware lifecycle. The A–T–C–G proof lifecycle model, Nukleoid role architecture, and deterministic Guard security engine are designed to make verification a runtime-native capability — integrated at the protocol layer rather than delegated to the application layer.

What Helixium Delivers

Proof-Native Runtime

Proof objects are first-class runtime entities with visible state, role accountability, and compact lifecycle receipts. Current proof state and compact receipts are queryable from runtime storage. Complete historical transitions remain available through block events and may require an archive node or indexer.

A–T–C–G Lifecycle Model

A structured four-stage proof lifecycle — Authorization, Transfer, Certification, Guard — with invalidation support and append-only audit records. Audit completeness and challenge-path security remain subject to independent review.

Nukleoid Role Architecture

Dynamic A–T–C–G validator roles providing specialised, accountable participation at each lifecycle stage, with deterministic epoch-bound assignment. Role independence and slashing mechanisms are planned capabilities pending mainnet readiness.

Portable Proof Bundles

Independently verifiable proof packages decoupled from the originating application, accessible via standardised endpoints and a planned Verifier SDK. Portability guarantees are subject to ongoing protocol development and audit.

Fixed-Supply Token Economics

Maximum-supply policy target of 46,000,000 HLX; base unit nucleo (1 HLX = 100,000,000 nucleo); no inflationary issuance policy; permanent supply enforcement is a mainnet requirement pending implementation and independent verification. Utility spans fees, bonding, governance, and ecosystem incentives.

Honest Commitments

- Helixium will not claim mainnet status until all readiness criteria are publicly met
- The team commits to transparent progress reporting against each roadmap phase
- No guaranteed returns, no guaranteed listings, no speculative promises
- The current testnet retains Sudo/Root authority; on-chain governance will replace this in Phase 4 following audit and deployment of the relevant pallets
- All forward-looking statements are subject to change; participants should seek independent legal and financial advice

Next Steps for Interested Parties

Validators Monitor Phase 2 announcements for public testnet validator onboarding and eligibility requirements. Validator allocation tiers and bonding parameters will be published as part of the revised allocation framework prior to distribution events.	Developers Access the public proof API and verifier endpoint at the demo environment. Verifier SDK is planned for Phase 3. Full runtime reference documentation is available on request.
Institutional and Ecosystem Partners Contact the Helixium team regarding validator operation, technical collaboration, infrastructure support, research partnerships, protocol integrations, or ecosystem grant opportunities. Allocation and vesting details will be published before relevant distribution events.	Grant Reviewers Full technical documentation and runtime reference available on request. Governance, token-distribution policy, and the revised allocation framework to be published separately prior to applicable milestones.

Frequently Asked Questions

Technical

1

What is the difference between A-T-C-G and block consensus?

A-T-C-G is a proof lifecycle architecture at the application and runtime layer. Block consensus is determined by Aura (block production) and GRANDPA (finality). They are complementary, not equivalent. A-T-C-G does not replace block consensus.

2

What does ProofLifecycleFinalized mean?

It means the proof has completed its full A-T-C-G lifecycle through the Guard stage — proof lifecycle closure, not block-level consensus finality. Both are required for production-grade security.

3

Is raw data stored on-chain?

Raw source documents are not intended to be stored on-chain; they remain with the issuing or responsible party. The chain can store compact proof data as well as account, role, assignment and other protocol records. The precise on-chain data set is defined by the deployed runtime.

4

Can proof bundles be verified without the original UI?

API-assisted verification is available via `/proof/:id/bundle` and `/proof/:id/verify` without the originating UI. These endpoints are served by the Helixium API and are not independent of it. Offline verification against independently obtained chain evidence is a planned Phase 3 capability (Verifier SDK and CLI verifier).

5

What does "Engine does not equal Role" mean?

Every validator may run the complete A-T-C-G engine locally. However, a validator may only issue an authoritative attestation for the role assigned to it for a specific proof and epoch. Full computation does not grant unilateral state-transition authority.

6

What consensus does Helixium use?

The current Spec-121 testnet uses Aura for block production and GRANDPA for finality. The chain does not use BABE. A-T-C-G role assignment uses a finalized-block-hash-derived assignment beacon configured through a Root-authorized runtime call.

7

What runtime pallets are deployed?

The deployed Spec-121 testnet runtime includes pallets such as HelixProof, BlockHelix, HelixValidatorRegistry, HlxTransfer, Aura, Grandpa, Balances, TransactionPayment, Sudo, Assets, and Session. "ProofLifecycle," "NukleoidRoleRegistry," and "GuardChallenge" are logical protocol responsibility domains, not necessarily separately deployed pallets under those exact names.

Frequently Asked Questions — Continued

Token and Investment



Is HLX inflationary?

No inflationary issuance policy. Total maximum-supply policy target is 46,000,000 HLX. Permanent supply enforcement is a mainnet requirement pending implementation and independent verification. The current testnet retains Sudo/Root authority.



What is the base unit?

1 HLX = 100,000,000 nucleos. 1 nucleos = 10^{-8} HLX. Total nucleos equivalent: 4,600,000,000,000,000 nucleos.



What are transaction fees?

Eligible signed transactions incur a base fee. Observed live Spec-121 base fee for HlxTransfer.transfer_keep_alive is 100,000 nucleos (0.001 HLX) plus any optional tip. Fees route to the deterministic fee/rewards vault (PalletId hlx/rewd). Fees are not burned. Reward distribution to validators is a separate specification.



When will HLX be listed on exchanges?

No exchange listing is guaranteed or scheduled at this time. Do not make investment decisions based on an assumption of exchange listing.



What is the Early Supporter Round vesting?

The Early Supporter Round comprises 920,000 HLX, equal to 2% of total supply, and is included within the 30% Community and Sale allocation. Its eligibility, pricing, lock, and vesting terms will be published before any distribution. No specific schedule is committed in this whitepaper.



What is the token allocation?

The revised mainnet planning allocation is: Ecosystem and Grants 17% (7,820,000 HLX), Validator Incentives 10% (4,600,000 HLX), Community and Sale 30% (13,800,000 HLX, includes Early Supporter Round), Treasury and Emergency Reserve 10% (4,600,000 HLX), Founding Team 15% (6,900,000 HLX), Strategic Partners 10% (4,600,000 HLX), Social Impact Reserve 8% (3,680,000 HLX). Total: 100% / 46,000,000 HLX. Detailed distribution terms, wallet addresses, and vesting schedules will be published separately before distribution.



Is this a securities offering?

This document is for informational purposes only and does not constitute a securities offering. Participants should seek independent legal advice in their jurisdiction before participating.

Project

What runtime is Helixium built on?

Substrate (Polkadot SDK), current reference Spec-121. Custom pallets implement the HelixProof, BlockHelix, HelixValidatorRegistry, and HlxTransfer logic. Exact runtime version details are published through runtime metadata and technical release manifests.

When is mainnet?

Mainnet is contingent on meeting all Phase 5 readiness criteria. No fixed date is committed at this stage. The team commits to transparent public communication of progress.

Does Helixium use Sudo/Root?

Yes. The current testnet retains Sudo/Root authority. Runtime upgrades can be executed by the Root account without on-chain governance approval. On-chain governance is planned for Phase 4 and will replace Sudo/Root once audited and deployed.

What does C mean in A-T-C-G?

C always means Certification in the Helixium protocol. It refers to the Certification Nukleoid role and the Certification stage of the proof lifecycle. C does not mean Consensus.

Runtime Architecture

Helixium is built on a Substrate-based runtime with custom pallets implementing the A–T–C–G proof lifecycle, Nukleoid role registry, and validator coordination logic — designed for auditability, determinism, and forkless upgrades.

Deployed Runtime Pallets

The deployed Spec-121 testnet runtime includes pallets: HelixProof, BlockHelix, HelixValidatorRegistry, HlxTransfer, Aura, Grandpa, Balances, TransactionPayment, Sudo, Assets, and Session. Proof Lifecycle, Nukleoid Role Registry, and Guard/Challenge are logical responsibility domains, not necessarily separately deployed pallets; exact call names and event definitions are published through runtime metadata.

Proof Lifecycle Domain Proof object creation, A–T–C–G state transitions, lifecycle events, role records, and block references.	Nukleoid Role Registry Domain Validator role registrations, A–T–C–G capabilities, per-proof epoch assignments, and role audit trails.	Guard/Challenge Domain Challenge windows, evidence submission, Guard quorum evaluation, invalidation, and decision recording.	StakeRegistry (<i>Planned — Phase 4</i>) Validator bonding, minimum stake, slashing hooks — not yet active.
--	---	---	---

Consensus and Upgrade Mechanism

Block Production Aura (Authority Round) for deterministic block production; the chain does not use BABE.	Finality GRANDPA for block finality; proof lifecycle finality is separate and does not replace block-consensus finality.	Runtime Upgrades Current testnet uses Sudo/Root; forkless upgrades available via Substrate; governance-controlled upgrades planned for Phase 4.
--	--	---

Conceptual Protocol Interface

Operations below are based on the logical protocol model; exact deployed call and event names are published through runtime metadata and technical release manifests.

Operation	Protocol Responsibility	Confirmed Event
Anchor Proof	Submit a new proof object for on-chain anchoring	ProofAnchored
Submit A/T/C Attestation	Role-assigned Nukleoid validates its stage (Authorization, Transfer, or Certification)	LifecycleStageSigned; NukleoidRoleVerified after successful stage verification
Submit Guard Attestation	Guard Nukleoid evaluates policy, quorum, and issues final lifecycle decision	LifecycleStageSigned; NukleoidRoleVerified; ProofLifecycleFinalized after all conditions are satisfied
Open Challenge	A signed account with the required challenge bond opens a challenge after Certification and before Guard finalization by submitting an evidence reference.	ProofChallengeOpened; FraudProofSubmitted; GuardChallengeOpened
Resolve Challenge	Lifecycle validators submit resolution votes; the Guard-stage quorum threshold accepts or rejects the challenge.	ChallengeResolutionVoteRecorded; after quorum: GuardChallengeResolved and the corresponding accepted or rejected outcome event.
Register Validator Identity	Validator registers its on-chain identity and operational metadata	ValidatorIdentityRegistered
Submit Validator Heartbeat	Validator emits liveness signal for eligibility monitoring	ValidatorHeartbeat

Runtime Properties

- On-chain storage** minimises data: originating documents and raw payloads are not stored; proof receipts, hashes, lifecycle state, role assignments, block references, and other protocol state are stored as required by the runtime; all state transitions are deterministic.
- Forkless upgrade path** via Substrate allows pallet updates without chain restarts; governance-controlled upgrades are planned for Phase 4.

Glossary

The following definitions are provided to ensure precise, unambiguous interpretation of the technical and economic terminology used throughout this whitepaper. Terms are defined in the context of the Helixium protocol and should not be assumed to carry identical meanings in other blockchain or cryptographic contexts.

Term	Definition
A–T–C–G	The Helixium proof lifecycle model consisting of four sequential stages: Authorization (A), Transfer (T), Certification (C), and Guard (G). A–T–C–G defines the protocol-level sequence through which a proof object passes from submission to finalization or invalidation. It is not a block consensus mechanism.
Aura	Authority Round — the block production mechanism used in the current Spec-121 testnet. Aura provides deterministic, slot-based block authoring. The chain does not use BABE.
Assignment	The deterministic, epoch-bound process by which the runtime selects validators for each A–T–C–G role for a specific proof.
Assignment Epoch	A bounded time window defined by finalized block height and governance-configured epoch length, within which role assignments are valid.
Challenge	A bonded dispute opened after Certification and before Guard finalization by a signed account able to reserve the required challenge bond and submit an evidence reference. An active challenge suspends Guard finalization. Active lifecycle validators submit resolution votes, and the configured Guard-stage quorum threshold determines whether the challenge is accepted or rejected. An accepted challenge may transition the proof to Invalidated; a rejected challenge returns the Guard stage to its pending finalization path.
Decision	The authoritative output of the Guard Decision Engine: ACCEPT, REJECT, PENDING, or REVIEW. Decisions are recorded on-chain as part of the decision audit trail.
Early Supporter Round	A token allocation of 920,000 HLX, representing 2% of the total maximum supply. Included within the 30% Community and Sale allocation. Eligibility, pricing, lock, and vesting terms will be published before any distribution; no specific schedule is committed in this whitepaper.
Eligibility	The set of conditions a validator must satisfy to be considered for role assignment: active registration, declared capability, liveness, and (planned) minimum bonded stake.
Engine	The complete A–T–C–G software stack that a validator runs locally to evaluate proof objects across all four lifecycle stages. Running the engine does not grant authority over all four roles.
Evidence	A compact, on-chain-referenced record submitted to the Guard Core for evaluation. Evidence may include fraud evidence, external observations, or challenge submissions.
Finalization	The completion of the Guard stage of the A–T–C–G lifecycle, recorded as a <code>ProofLifecycleFinalized</code> event. This is a proof lifecycle closure event and is distinct from block-consensus finality.
GRANDPA	GHOST-based Recursive ANcestor Deriving Prefix Agreement — the block finality mechanism used in the current Spec-121 testnet. GRANDPA provides Byzantine fault-tolerant finality for the block chain. A–T–C–G proof lifecycle finality is a separate, complementary concept and does not replace or redefine block-consensus finality.
Guard	The fourth stage of the A–T–C–G lifecycle and the Nukleoid role responsible for deterministic security evaluation, policy compliance, quorum validation, and final protocol decisions.
GuardChallenge	A logical protocol responsibility domain encompassing challenge windows, evidence submission, Guard quorum evaluation, invalidation logic, and on-chain decision recording. GuardChallenge is a conceptual grouping of protocol responsibilities and is not necessarily a separately deployed pallet under that exact name.
HLX	The native token of the Helixium network. The protocol defines a maximum-supply policy target of 46,000,000 HLX. Permanent enforcement of this supply cap is pending mainnet implementation. HLX is subdivided into nucleo (1 HLX = 100,000,000 nucleo).
hlx/rewd vault	The deterministic fee and rewards vault derived from the PalletId "hlx/rewd". Transaction fees and tips are directed to this vault. Reward distribution to validators is governed by a separate specification and is not automatically disbursed from this vault.
Lifecycle State	The current phase of a proof object within the A–T–C–G lifecycle: Pending, Authorized, Transferred, Certified, Finalized, Invalidated, or PendingReview.

Glossary — Continued

Term	Definition
Local Evaluation	The process by which a validator runs the complete A–T–C–G engine against a proof locally. Local evaluation produces a Local Verification Receipt, not an authoritative network decision.
Local Verification Receipt	A validator-internal artefact produced by local evaluation. It is not a network-level state transition and does not constitute an authoritative role attestation.
Logical Protocol Component	A conceptual grouping of related protocol responsibilities used to describe functional domains within the Helixium runtime. Logical protocol components are distinct from deployed runtime pallets; a logical component may be implemented across one or more pallets, and the exact pallet names, call names, and event definitions are published through runtime metadata and technical release manifests.
Module	A discrete, single-responsibility component within the Guard security architecture (e.g., Structure Validator, Policy Engine, Quorum Validator). Each module evaluates one specific security domain.
nucleo	The smallest indivisible unit of HLX. 1 HLX = 100,000,000 nucleo; 1 nucleo = 10^{-8} HLX. All on-chain balances and fee calculations are denominated in nucleo.
NukleoidRoleRegistry	A logical protocol responsibility domain that tracks validator role registrations, declared A–T–C–G capabilities, per-proof epoch assignments, and role audit trails. NukleoidRoleRegistry is a conceptual grouping of protocol responsibilities and is not necessarily a separately deployed pallet under that exact name.
Observation	A signed output from an external analyzer (e.g., AI risk model, malware scanner) submitted as evidence to the Guard Core. Observations inform Guard evaluation but do not directly control blockchain state.
Proof Lifecycle	The complete sequence of A–T–C–G stages through which a proof object passes from submission to finalization or invalidation, with each transition recorded as an on-chain event. ProofLifecycle is also used as a logical protocol responsibility domain name and is not necessarily a separately deployed pallet under that exact name.
ProofLifecycleFinalized	A runtime event emitted when all A–T–C–G lifecycle and quorum conditions for a proof object have been satisfied and the Guard stage has concluded. ProofLifecycleFinalized is a proof lifecycle closure event and does not replace, redefine, or interact with block-consensus finality as provided by GRANDPA.
Quorum	The minimum number of valid, role-assigned validator attestations required to advance a proof through a lifecycle stage or to reach a Guard decision.
Role	A specific functional capability within the A–T–C–G lifecycle — Authorization (A), Transfer (T), Certification (C), or Guard (G) — assigned to a validator for a specific proof and epoch.
Role Attestation	A cryptographically signed statement issued by a validator for its assigned role in a specific proof epoch. Only role-assigned validators may issue authoritative attestations.
Sudo/Root	The privileged upgrade authority used on the current Spec-121 testnet to authorize runtime upgrades and administrative operations. Sudo/Root access is planned to be removed and replaced by on-chain governance in Phase 4.
Validator	A network participant that registers Nukleoid capabilities, bonds HLX (planned), and is assigned roles dynamically per proof and assignment epoch.

Legal Notices and Disclaimers

This section contains the legal notices applicable to this whitepaper and the Helixium protocol. Nothing in this document constitutes legal, financial, tax, or investment advice.

Informational Purpose Only

This document is published for informational purposes only. It does not constitute an offer to sell, a solicitation to buy, or a recommendation to purchase any token, security, or financial instrument. Readers should seek independent legal, financial, and tax advice before making any decision related to the Helixium protocol or HLX.

No Securities Offering

This document does not constitute a prospectus, offering memorandum, securities offering, or invitation to purchase HLX or any other financial instrument. No exchange listing of HLX is guaranteed. Secondary market liquidity is not assured.

Risk of Loss

Token purchases carry risk, including the risk of total loss. HLX is not a guaranteed financial return instrument. No financial return is represented or implied. Do not invest funds you cannot afford to lose.

Forward-Looking Statements

This document contains forward-looking statements regarding the Helixium protocol, roadmap, and token economics. These statements are subject to change due to technical, regulatory, or resource constraints. Actual outcomes may differ materially from those described.

Regulatory Variation

The regulatory treatment of blockchain networks, digital assets, validator participation, and related activities may vary by jurisdiction. Each participant is responsible for determining and complying with the applicable requirements in their jurisdiction.

Supply Policy Disclaimer

The protocol-defined maximum supply of 46,000,000 HLX is a mainnet monetary policy target; permanent enforcement is pending implementation and independent verification. The Early Supporter Round of 920,000 HLX (2% of total supply) is included within the 30% Community and Sale allocation and is not additional supply beyond the 46,000,000 HLX target.

Testnet Authority Notice

The current testnet retains Sudo/Root authority. Runtime upgrades can be executed by the Root account without on-chain governance approval. Sudo/Root access is planned to be replaced by on-chain governance in Phase 4.

Token Allocation Disclaimer

The token allocation framework is a mainnet planning allocation — not proof that vaults, multisig, on-chain vesting, automated grants, or immutable governance are already implemented. Distribution terms, wallet addresses, and vesting schedules will be published before any distribution event.

Limitation of Liability

To the maximum extent permitted by applicable law, the Helixium team, contributors, and associated entities shall not be liable for any damages arising from reliance on this document, participation in the Helixium network, or any token-related activity. This document is provided on an 'as is' basis without warranties of any kind.

Intellectual Property

All protocol designs, architecture descriptions, and technical specifications in this whitepaper are the intellectual property of the Helixium project. The Helixium name, HLX ticker, Nukleoid terminology, and A–T–C–G lifecycle model are proprietary designations. Third parties may reference this document for informational purposes with appropriate attribution.

Document Integrity

This document is Technical Whitepaper v1.1, a substantive revision of v1.0. Readers are encouraged to verify document authenticity through official Helixium communication channels.