3SAT Coin: Protocol Mechanism and Tokenomics

3SAT LAB

April 2026

Abstract

$3SAT Coin is a bounty protocol for SAT problem instances with on-chain escrow and settlement, decentralized content-addressed data availability, and off-chain computation for solving and verification. The protocol is designed to (i) mitigate copying and front-running via commit–reveal, (ii) guarantee correctness via verifier attestations and challenge-based finality, (iii) reduce enterprise confidentiality frictions through client-side instance obfuscation, and (iv) create non-speculative demand for the native token ($3SAT) through staking, bonds, and governance. This document specifies the system architecture, lifecycle/state machine, security and incentive design, and tokenomics, with a staged bootstrap-to-DAO governance migration.

Contents

1 Introduction

1.1 Motivation

$3SAT Coin is designed as a *verifiable outcome marketplace* for constraint solving. The protocol focuses on a narrow but powerful primitive: solving SAT/CNF instances with economically enforced correctness. The core market need is not raw compute as such, but *high-confidence outcomes*: a correct satisfying assignment, an UNSAT result together with the appropriate proof artifact or certificate, or another solver output whose validity can be independently checked.

This market need arises because solving large-scale discrete constraint problems is often computationally expensive, operationally bursty, and difficult to source efficiently through traditional centralized channels. Demand may spike unpredictably, buyers care about correctness rather than the internal compute path, and a single centralized solver service introduces trust, censorship, and pricing-power concerns. At the same time, naive on-chain submission is not viable: it invites mempool copying and solution theft, while full correctness checking is too expensive to enforce purely on-chain.

A further requirement arises in enterprise settings such as semiconductor EDA, hardware verification, internal configuration checking, and security policy validation: the *instance itself* may contain commercially sensitive semantics. Variable names, module identifiers, internal signals, or rule labels may reveal confidential architecture or workflow information even if the mathematical problem is "just" a CNF file. For this reason, $3SAT Coin supports *client-side obfuscation*. Within the issuer's trusted local environment, semantic variables are mapped to randomly assigned numeric identifiers, and the mapping dictionary is retained locally by the issuer. Only the obfuscated, purely mathematical CNF instance is exported for publication. In this sense, the protocol has zero knowledge of the issuer's proprietary business semantics in the ordinary informational sense: the network can solve the obfuscated constraint system without learning the issuer's internal naming scheme or domain-specific interpretation.

$3SAT Coin addresses these frictions by combining (i) on-chain escrow and deterministic settlement, (ii) creation-time on-chain binding of a content-addressed instance identifier, (iii) decentralized storage for data availability, (iv) commit–reveal for anti-copying at the solution stage, and (v) decentralized off-chain verification with on-chain attestations, challenge procedures, and economic penalties for provable misbehavior. The resulting thesis is that there is meaningful demand for *verifiable constraint-solving-as-a-service*: a market in which buyers pay for outcomes and rely on correctness guarantees backed by stake, slashing, disputes, and open data availability rather than trust in a single operator.

1.2 Protocol positioning: outcome market rather than generic compute market

A useful way to understand $3SAT Coin is to distinguish it from *generic compute marketplaces*. Horizontal compute markets primarily sell capacity: buyers rent computation for arbitrary workloads, while verification, service quality, and trust are often left to off-chain arrangements, reputation, or additional assumptions. By contrast, $3SAT Coin is organized around a narrower product with a clearer economic contract: a *specific, verifiable outcome* for a structured class of problems.

This positioning matters for two reasons. First, buyers in verification, security, EDA, and related domains often care less about who performed the computation than about whether the final answer is correct and can be trusted. Second, a narrow vertical permits stronger mechanism design. Because the protocol is specialized to SAT instances, it can make correctness and anti-copying first-class rather than secondary concerns. Commit–reveal protects solver effort from mempool copying and front-running; attestations, challenge windows, and slashable collateral make correctness economically enforceable; and a standardized artifact type makes it easier to build repeated workflows and market liquidity around the same task family.

In this sense, the protocol does not aim to be a universal market for arbitrary computation. It aims to be a specialized market for *verifiable discrete-computation outcomes*, where narrow scope is a feature rather than a limitation.

1.3 Why SAT as the core primitive?

SAT is a natural protocol primitive because it is both economically meaningful and computationally universal. A large class of discrete decision and verification problems can be reduced to SAT through standard encodings, making SAT a common backend language for a wide range of real-world workloads.

In practice, SAT instances arise in at least three broad settings:

1. **Verification and consistency checking.** Many important verification tasks can be encoded as CNF constraints, including bounded model checking, circuit equivalence, hardware logic verification, configuration and policy consistency checks, and safety or security invariants of the form "no bad state is reachable within K steps." Security and correctness verification for smart contracts can also be formalized as SAT problems.

2. **Constraint satisfaction, planning, and scheduling.** Assignment, scheduling, and discrete planning tasks often reduce to satisfiability or to adjacent formalisms that can be compiled into SAT. This includes resource allocation, combinatorial scheduling, and planning problems with rich logical constraints.

3. **Discrete optimization and related primitives.** Optimization problems can often be handled through decision versions encoded in SAT (for example, via binary search on the objective) or through closely related primitives such as MaxSAT or pseudo-Boolean formulations that remain in the same technical ecosystem.

SAT is also unusually suitable for market design. The task artifact is standardized, correctness checking can often be made substantially cheaper than solving, and the broader SAT ecosystem already has benchmarking norms, solver competitions, and performance-oriented communities. These features make it more plausible to create a liquid and repeatable market for outcomes than in many generic-compute settings.

Accordingly, \$3SAT Coin should be viewed not as a marketplace for one narrow academic benchmark, but as infrastructure for a broader class of verifiable discrete-computation tasks whose outputs can be encoded, checked, and economically settled.

1.4 Market participants and incentives

The protocol is fundamentally a *three-sided market* with distinct but complementary roles.

Bounty proposers (issuers). Issuers are entities that have a real solving or verification problem and are willing to pay for a correct outcome. Representative participants include security and verification teams, EDA and semiconductor verification groups, tooling companies that need burst solving capacity, research labs or benchmark sponsors, and AI-agent operators that require verifiable constraint solutions. What these users want is straightforward: correct results, predictable settlement, strong data availability, and minimal reliance on a single trusted vendor.

Solvers (bounty hunters). Solvers are participants able to produce valid solutions. They may include professional solver developers, researchers, competitive SAT practitioners, or specialized compute providers with tuned solver stacks. Their motivation is economic upside from producing correct outcomes under clear and enforceable rules. For solvers, the protocol must also protect against copying, front-running, and operator-side withholding, which is one reason creation-time CID binding and the commit–reveal lifecycle are both central to the design.

Verifiers (attestation operators). Verifiers independently check correctness and provide the trust layer that turns raw solver competition into a credible market outcome. They may include independent verification operators, specialized oracle-style verifier networks, or auditing and verification service providers where appropriate. Their motivation is verification rewards and possible fee shares, but with accountability: verifier participation is backed by stake and/or bonds that are slashable upon proven misbehavior.

The protocol's economic design is therefore not incidental. It is the mechanism that aligns the incentives of buyers, solvers, and verifiers while reducing trust assumptions through collateral-at-risk, challenge-based review, transparent on-chain settlement, and open access to the underlying obfuscated problem artifact.

1.5 Why this market can sustain repeated use

A recurring weakness of many broad compute marketplaces is that buyer demand, quality assurance, and workflow integration are difficult to standardize across many unrelated task types. By contrast, a specialized SAT outcome market can be embedded into recurring verification and developer workflows. Buyers in these settings often return with repeated instance families, similar correctness requirements, and established internal processes for acting on solver outputs.

This gives the protocol a clearer path to repeated usage. First, the buyer is paying for correctness rather than generic capacity. Second, the product is standardized enough to support typed programs, benchmark-linked subsidy policies, and long-tail coverage mechanisms. Third, solver and verifier participation can deepen around a recognizable task class rather than around arbitrary heterogeneous jobs. These features do not guarantee success, but they create a more coherent product-market fit than a marketplace that attempts to support all forms of compute equally.

1.6 Connection to AI agents and verifiable machine-assisted workflows

Recent AI developments strengthen the case for a protocol such as $3SAT Coin. Many modern AI systems, especially agentic workflows, operate in environments where *planning under explicit constraints* matters. Agents may need to make decisions subject to permissions, resource limits, timing restrictions, configuration rules, or logical consistency requirements. These are precisely the kinds of settings in which SAT encodings are useful.

There are at least three important interaction channels between AI systems and SAT solving:

1. **Agents for planning and constraint-based decision making.** AI agents often need a backend primitive for solving discrete feasibility and planning problems under structured constraints. SAT provides a natural candidate.

2. **Verifiable reasoning and safety checks.** A growing concern in AI deployment is that model outputs may be plausible but incorrect, inconsistent, or policy-violating. Constraint-based post-checking can provide a formal validation layer for certain classes of outputs.

3. **LLM-assisted encoding, solver-based validation.** Large language models may help generate candidate encodings, problem decompositions, or workflow specifications, while SAT solvers provide the rigorous correctness check. In this division of labor, generative models propose and symbolic solvers verify.

This does not mean that all AI tasks reduce to SAT. Rather, it suggests that a growing class of AI-enabled systems will benefit from access to a marketplace for *verifiable discrete reasoning outcomes*, especially when correctness and auditability matter.

The same point is important for enterprise deployment. An internal AI system may compile a sensitive planning or verification task into CNF inside a trusted environment, apply client-side obfuscation to strip semantic identifiers, retain the local decoding dictionary, and publish only the obfuscated instance to decentralized storage. The network then solves or verifies a mathematically faithful problem without learning the issuer's proprietary semantics. Returned satisfying assignments can be interpreted locally by the issuer or its agent against the retained mapping. This makes agentic workflows and enterprise confidentiality compatible rather than mutually exclusive.

1.7 Design goals

The protocol is designed to satisfy:

- **Anti-copying:** reduce feasibility of mempool-based copying and preemptive submission.
- **Correctness:** ensure finality implies high-confidence validity, with explicit dispute procedures.

- **Liveness:** enable timely resolution with verifier quorum acceleration.
- **Economic security:** require collateral-at-risk and apply slashing for misbehavior.
- **Auditability:** keep critical state transitions on-chain; make treasury and governance transparent.
- **Data availability neutrality:** prevent any single operator from gaining a first-look advantage by withholding published bounty files.

1.8 Document organization

Section 2 describes actors, modules, and the end-to-end flow. Section 3 specifies the on-chain contracts. Section 4 gives the normative lifecycle/state machine. Section 5 summarizes security and incentive considerations. Section 6 specifies token supply, distribution, vesting, and value accrual. Section 7 specifies treasury design and governance. Section 8 describes the bootstrap and early-stage subsidization policy.

2 Protocol Overview and System Architecture

2.1 Actors and modules

The system consists of the following roles and modules:

- **Bounty Issuer.** Encodes a SAT instance, optionally applies client-side obfuscation inside a trusted local environment, uploads the obfuscated artifact to decentralized storage, and escrows the reward on-chain together with the content-addressed instance identifier.
- **Solver (Bounty Hunter).** Retrieves the published instance, verifies that the downloaded file matches the on-chain CID, solves locally, and uses commit–reveal to claim rewards.
- **Verifier.** Performs off-chain validation of revealed solutions and submits on-chain attestations; participates in disputes.
- **Decentralized Storage Layer (e.g., IPFS/Arweave).** Stores obfuscated bounty instances and large solution artifacts under content-addressed identifiers. This layer provides public data availability independent of any single protocol operator.
- **Untrusted Indexer & Cache / API Server.** Listens to on-chain events, fetches CID-addressed artifacts from decentralized storage, parses lightweight metadata (e.g., variable count, clause count, timestamps, status), caches raw files, and serves fast Web2-style search and download APIs. This component is operationally useful but not trusted for correctness, custody, or exclusive data access.
- **User.** Observes bounties, status, and history; may participate as issuer/solver/verifier.

Two design principles are central here. First, confidentiality protection begins at the client boundary rather than at the protocol operator: semantic identifiers are stripped before publication. Second, publication means *openly available by CID*, not "available at the discretion of the official server." The indexer/cache improves usability and latency, but it is never the sole data source.

2.2 High-level flow

The canonical flow is:

1. Issuer prepares a SAT instance. Where confidentiality matters, the issuer locally obfuscates semantic variables into randomly assigned numeric identifiers and retains the private mapping dictionary off-protocol.
2. Issuer uploads the obfuscated instance to decentralized storage (e.g., IPFS/Arweave) and obtains a content-addressed identifier instCID.
3. Issuer creates a bounty on-chain, escrows the reward, and binds instCID and timing parameters in the bounty state.
4. The untrusted indexer/cache observes the on-chain creation event, fetches the instance from decentralized storage, extracts lightweight metadata, and offers fast search/download endpoints for convenience.
5. A solver retrieves the instance either from the public decentralized storage layer or from the untrusted cache, locally verifies that the file hash matches the on-chain instCID, and then submits a commit (hash) while locking a submission bond.
6. The solver later reveals a content-addressed solution reference plus salt.
7. Verifiers independently verify off-chain and submit attestations.
8. Finality occurs after a challenge window; a verifier quorum may shorten residual challenge time.
9. Payout and fee/slashing flows are executed on-chain.

This flow eliminates the "official server sees the instance first" problem. Once the bounty is created, download rights are effectively announced to the whole network by the on-chain publication of instCID. A protocol-operated cache may accelerate access, but it cannot be the gatekeeper.

2.3 Commit–reveal as anti-copying mechanism

To mitigate front-running and copying, solvers first publish a cryptographic commitment and later reveal a solution reference within a fixed window. This reduces the incentive and feasibility of mempool copying because the on-chain commit alone does not expose the solution reference. The normative lifecycle is specified in Section 4.

It is useful to distinguish *two* different anti-front-running layers. At the *pre-solve* stage, the issuer-side instance CID is bound on-chain and the underlying file is available from decentralized storage, which prevents an operator from selectively withholding the problem from the public while solving it privately. At the *post-solve* stage, commit–reveal prevents other parties from copying a solver's discovered answer from the mempool before reveal finality.

2.4 Optional credit/reputation

The protocol may compute a *credit* score for solvers and verifiers based on observable outcomes (validity rate, non-reveal rate, dispute outcomes). Credit is *non-consensus*: it can influence UX (ranking, recommended counterparties) or soft eligibility, but core security relies on stake/bonds and slashing.

2.5 Architecture diagram

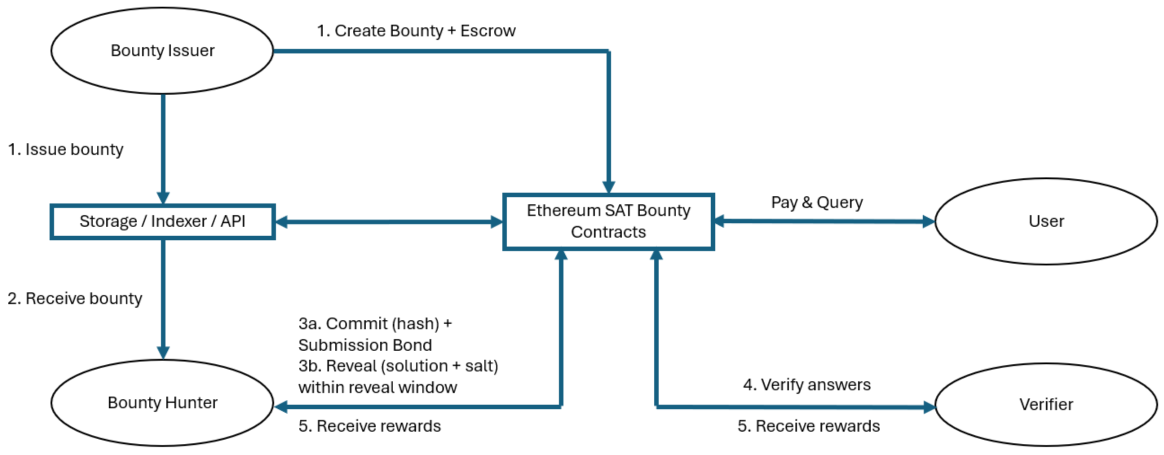

Figure 1: System architecture of $3SAT Coin, including issuer-side obfuscation, decentralized content-addressed storage, on-chain CID binding, and an untrusted indexer/cache layer.

Diagram note. In the intended architecture, client-side obfuscation sits within the issuer's trusted environment; the obfuscated instance is published to IPFS/Arweave; the smart contract stores the instance CID and bounty parameters; and the protocol-operated indexer/cache merely accelerates retrieval by mirroring and indexing public CID-addressed content.

3 On-chain Smart Contract Infrastructure

3.1 Contract responsibilities

Smart contracts provide:

- **Escrow:** lock issuer rewards until finality.

- **State machine:** enforce commit/reveal timing, record attestations, and gate payouts.
- **Content binding:** immutably bind the bounty's content-addressed instance identifier at creation and the solver's revealed solution reference at reveal/finalization.
- **Economic security:** manage staking, bonds, and slashing.
- **Fee routing:** burn/treasury allocation according to governance parameters.

3.2 Core transaction types

The protocol supports:

1. *Bounty creation:* create a new bounty with escrowed reward, timing parameters, and the content-addressed instance identifier `instCID` (rather than the raw file). Optional metadata digests may also be included for typed programs or indexing support.
2. *Commit submission:* post a commitment hash and lock a submission bond.
3. *Reveal submission:* reveal the solution reference and salt; prove consistency with the commit.
4. *Verification attestations:* verifiers submit on-chain attestations for the revealed reference.
5. *Challenge/dispute (optional but recommended):* raise a dispute against an accepted outcome.
6. *Finalization and payout:* release rewards and distribute fees/slashes after finality.

The creation transaction is therefore more than a reward escrow. It is also the cryptographic publication point for the bounty instance. By committing `instCID` on-chain, the issuer fixes the exact obfuscated artifact to be solved and publicly announces the network's right to retrieve it from decentralized storage. No centralized API server is required to define what the bounty "really" is.

3.3 Fee and slashing flows

To align long-term token value with protocol usage, a portion of **net protocol fees** and **slashing penalties** is burned and/or routed to the treasury.

Source	**Destination**
Posting fee	burn α + treasury $(1 - \alpha)$
Slashed solver bond	challenger β + verifiers γ + burn/treasury (remainder)
Verifier staking penalty (optional)	challenger / treasury (by rule)

Table 1: Fee and slashing flows.

4 Lifecycle Specification: Commit–Reveal, Verification, Challenge, and Finality

4.1 Normative goals

The objectives are:

- mitigate copying via commit–reveal,
- ensure correctness via challenge-based finality,
- improve liveness via verifier quorum acceleration.

4.2 Timing parameters

Let the protocol define the following windows:

- Δ_{rev}: reveal window after a commit.
- Δ_{ver}: verifier attestation window after a reveal.
- Δ_{ch}: full challenge window.
- $\Delta_{\mathrm{ch}}^{\mathrm{short}} < \Delta_{\mathrm{ch}}$: residual challenge period after quorum (early finality backstop).

All timing parameters are set at bounty creation or by global governance policy, and are updated only through timelocked governance actions (Section 7).

4.3 Commitment object and off-chain solution reference

Before solving begins, the bounty instance is already fixed by a *creation-time content-addressed commitment*. Let instCID denote the content-addressed identifier of the obfuscated CNF instance uploaded by the issuer to decentralized storage at bounty creation. The smart contract stores instCID immutably as part of the bounty state, so every solver and verifier can independently retrieve the same file and verify that its hash matches the on-chain identifier.

Later, after solving, let ref be a content-addressed reference to the solution artifact (e.g., IPFS/Arweave CID). The solver commits to:

$$c \;=\; H(\mathsf{bountyId} \,\|\, \mathsf{solverAddr} \,\|\, \mathsf{ref} \,\|\, \mathsf{salt})\,.$$

The hash function $H(\cdot)$ must be a collision-resistant primitive (e.g., Keccak-256).

Thus, the protocol uses content-addressed objects at *both* ends of the lifecycle: ex ante for the published problem instance and ex post for the revealed solution artifact. The former ensures immutable problem definition and public data availability; the latter supports reveal integrity and efficient off-chain artifact retrieval.

4.4 Commit phase

A solver *must* retrieve the published instance, verify locally that the downloaded file matches the on-chain instCID, and then submit $(\mathsf{bountyId}, c)$ before the bounty expires while locking a submission bond b_S in \$3SAT. The reveal *must* occur before $t_{\text{commit}} + \Delta_{\text{rev}}$. Failure to reveal on time triggers slashing per Section 4.8.

4.5 Reveal phase

A solver *must* submit $(\mathsf{ref}, \mathsf{salt})$ during the reveal window. The contract verifies:

$$H(\mathsf{bountyId} \| \mathsf{solverAddr} \| \mathsf{ref} \| \mathsf{salt}) = c.$$

If the check fails, the reveal is invalid and the solver bond is slashed.

4.6 Verification attestations and quorum

After a valid reveal, eligible verifiers verify the artifact referenced by ref off-chain and submit on-chain attestations, e.g., `attest(bountyId, ref, resultDigest)`. Let t_{quorum} be the first time a quorum of consistent attestations is reached.

4.7 Challenge-based finality with quorum acceleration

Absent quorum, finality occurs at $t_{\text{reveal}} + \Delta_{\text{ch}}$ provided no successful challenge has been raised. With quorum, the protocol may allow early finality at $t_{\text{quorum}} + \Delta_{\text{ch}}^{\text{short}}$:

$$t_{\text{final}} \;=\; \min\left\{ t_{\text{reveal}} + \Delta_{\text{ch}},\;\; t_{\text{quorum}} + \Delta_{\text{ch}}^{\text{short}} \right\},$$

provided no successful challenge occurs before t_{final}.

4.8 Bonds and slashing rules

The protocol uses \$3SAT-denominated collateral-at-risk:

- **Solver submission bond b_S:** slashed for non-reveal, inconsistent reveal, or invalid solution.
- **Verifier stake s_V and/or per-attestation bond b_V:** slashed if attestations are proven incorrect by a successful challenge (rules defined by governance).
- **Challenger bond b_C (optional):** required to deter frivolous disputes; refunded and rewarded upon successful challenge.

4.9 Parameterization and security–liveness trade-off

Governance calibrates $(\Delta_{\text{rev}}, \Delta_{\text{ver}}, \Delta_{\text{ch}}, \Delta_{\text{ch}}^{\text{short}}, b_S, s_V, b_V, b_C)$ to satisfy incentive compatibility and liveness targets. Updates are executed under timelock (Section 7).

5 Incentives and Security Considerations

5.1 Economic security intuition

Security derives from (i) *collateral-at-risk* (stake/bonds), (ii) *slashing* for provable misbehavior, and (iii) *challenge rewards* that increase the detection probability of incorrect outcomes.

5.2 Front-running and copying

There are two distinct attack surfaces.

First, **pre-solve withholding or operator-side racing** can arise if the bounty file is available only through an official server. $3SAT Coin mitigates this by requiring the issuer to publish the obfuscated instance to decentralized storage and bind its CID on-chain at bounty creation. Once the CID is on-chain, any solver can retrieve the same file from public IPFS/Arweave infrastructure, independent of the official indexer/cache. Even if the protocol-operated API is offline or maliciously withholds the file, solvers can still access the instance and locally verify that the downloaded content matches the on-chain CID. This removes the information asymmetry that would otherwise permit an operator to solve privately before public release.

Second, **post-solve copying** is mitigated by commit–reveal. The protocol should additionally encourage:

- using private transaction relays where available,
- limiting reveal window length, and
- requiring solver bond b_S to make spam commits expensive.

5.3 Sybil resistance

Sybil resistance is primarily economic: participation in solver/verifier roles requires $3SAT-denominated collateral, and rewards are capped in bootstrap programs (Section 8).

5.4 Liveness risks

Liveness depends on verifier participation. The treasury may subsidize verifier activity in early phases subject to spend caps (Section 7).

5.5 Parameter calibration for incentive compatibility

A sufficient condition for discouraging verifier collusion or false attestations is that the *expected penalty* from a successful challenge dominates the *maximum extractable benefit* from misbehavior. A practical, implementation-friendly approach is to calibrate verifier collateral *as a function of bounty risk exposure.*

Risk-exposure proxy. Let $G_{\max}$ denote an upper bound on the maximum benefit from a false attestation (e.g., the escrowed reward plus an allowance for side payments). A conservative proxy is

$$G_{\max} \approx R_{\text{escrow}} + \lambda_{\text{side}} R_{\text{escrow}},$$

where R_{escrow} is the bounty escrow and $\lambda_{\text{side}} \geq 0$ is a governance-set multiplier.

Rule-of-thumb collateral constraint. For each attestation, require slashable collateral to satisfy

$$s_V + b_V \;\geq\; \gamma\, G_{\max},$$

with $\gamma > 1$ (safety margin) and optional tiering by bounty size. Governance may implement a tier schedule (small/medium/large bounties) rather than a fully continuous function.

Dynamic adjustment and governance guardrails. Parameters (s_V, b_V) and tier thresholds can be adjusted by timelocked governance based on observable metrics (challenge rate, reversal rate, verifier participation, token volatility). To reduce governance attack surface, updates should respect hard-coded bounds (floors/ceilings) and require non-zero timelock delays.

6 Tokenomics: Supply, Demand, and Incentives

6.1 Token specification

$3SAT is the native ERC-20 token of $3SAT Coin, deployed on Ethereum.

- **Standard:** ERC-20.
- **Decimals:** 18 (configurable).
- **Core uses:** (i) staking collateral for verifiers (and optionally solvers), (ii) protocol bonds, (iii) governance, and (iv) phase-based fee discounts and access control.

6.2 Supply policy

We consider a fixed maximum supply. Let $S_{\max}$ denote maximum supply and S_0 initial circulating supply at genesis.

- **Minting:** $S_{\max}$ is minted at genesis and allocated to vesting, timelock, incentive-controller, liquidity, and treasury contracts. Circulating supply evolves only through lockup/vesting and governance-controlled distribution (see Section 6.4.1).
- **Burning:** protocol may burn tokens via fee/slashing flows (Table 1).
- **Important accounting distinction:** the *genesis treasury reserve allocation* is a stock concept, while later treasury inflows from fees and slashing are operating-flow concepts. Hence, treasury balances may exceed the genesis treasury allocation over time without changing total token supply.

6.3 Initial distribution

Bucket	**Share**	**Tokens**	**Purpose / Notes**
Community incentives	35%	$0.35S_{\max}$	Solver/verifier bootstrapping, long-tail subsidies, growth, bug bounties.
Genesis Treasury / DAO reserve	20%	$0.20S_{\max}$	Genesis reserve for bootstrap operations, strategic reserve, insurance, and grants. This is a *genesis allocation*, not a lifetime cap on treasury balances.
Team & contributors	15%	$0.15S_{\max}$	Core development; vesting schedule below.
Investors / strategic partners	25%	$0.25S_{\max}$	Early funding, strategic partnerships, and integrations; vesting schedule below.
Liquidity / market making	5%	$0.05S_{\max}$	Liquidity provisioning to support staking/bonding access.

Table 2: Initial allocation of $3SAT.

6.4 Vesting and lockups

Let T denote months since genesis. Define $A_{\text{team}} = 0.15S_{\max}$ and $A_{\text{inv}} = 0.25S_{\max}$.

Team & contributors. 12-month cliff then linear vesting over 24 months:

$$V_{\text{team}}(T) = \begin{cases} 0, & T < 12, \\ A_{\text{team}} \cdot \dfrac{T-12}{24}, & 12 \le T \le 36, \\ A_{\text{team}}, & T > 36. \end{cases}$$

Investors / strategic partners. 6-month cliff then linear vesting over 24 months:

$$V_{\text{inv}}(T) = \begin{cases} 0, & T < 6, \\ A_{\text{inv}} \cdot \dfrac{T-6}{24}, & 6 \le T \le 30, \\ A_{\text{inv}}, & T > 30. \end{cases}$$

Community incentives and treasury reserve. Community incentives are released programmatically under caps. The treasury reserve is governance-controlled and split into an immediately usable bootstrap operating sub-pool plus a more slowly available strategic reserve (see below).

6.4.1 Circulating supply path: $S_0 \to S_{\max}$

To make dilution and adoption incentives transparent, we distinguish *maximum supply* from *circulating supply*. Let $S_{\text{circ}}(T)$ denote the circulating supply at month T. A useful accounting decomposition is:

$$S_{\text{circ}}(T) = S_0 + V_{\text{team}}(T) + V_{\text{inv}}(T) + R_{\text{comm}}(T) + R_{\text{liq}}(T) + R_{\text{treasury}}(T),$$

where:

- $R_{\text{comm}}(T)$ is cumulative released community incentives (programmatic release; capped),
- $R_{\text{liq}}(T)$ is liquidity provisioning released to market making/LP operations (subject to policy),
- $R_{\text{treasury}}(T)$ is any treasury-controlled amount that becomes circulating (e.g., via grants/subsidies/market purchases). Treasury balances that remain locked or undeployed are *not* considered circulating.

Genesis circulating supply S_0 (policy choice). We set a low-but-sufficient initial float to support liquidity and early staking:

$$S_0 \;=\; S_{\text{liq},0} + S_{\text{boot},0},$$

where $S_{\text{liq},0}$ is the initially deployed liquidity/MM amount and $S_{\text{boot},0}$ is a one-time bootstrap distribution from the Community incentives bucket (e.g., early solver/verifier onboarding). As a reference configuration, one may set

$$S_{\text{liq},0} = 0.05 S_{\max}, \qquad S_{\text{boot},0} = 0.01 S_{\max},$$

so that $S_0 = 0.06 S_{\max}$. If liquidity deployment is staged, the non-deployed remainder enters circulation gradually via $R_{\text{liq}}(T)$ under governance/LABS policy.

Multi-year release schedule for Community incentives (upper bound). To make $S_0 \to S_{\max}$ transparent, Community incentives follow a pre-committed annual cap schedule (distributed across epochs and still subject to spend-cap rules):

Year since genesis	Max release	Cumulative (community)
Genesis (T=0)	1% $S_{\max}$	1%
Year 1	6% $S_{\max}$	7%
Year 2	6% $S_{\max}$	13%
Year 3	5% $S_{\max}$	18%
Year 4	5% $S_{\max}$	23%
Year 5	4% $S_{\max}$	27%
Year 6	3% $S_{\max}$	30%
Year 7	3% $S_{\max}$	33%
Year 8	2% $S_{\max}$	35%

This schedule sums to the full $0.35 S_{\max}$ Community incentives allocation.

Treasury availability vs. circulation. To avoid an overly rigid bootstrap while still protecting long-run reserves, the genesis Treasury / DAO reserve is split into two sub-buckets:

$$0.20S_{\max} \;=\; A_{\text{boot}} + A_{\text{res}},$$

where:

$$A_{\text{boot}} = 0.04S_{\max}, \qquad A_{\text{res}} = 0.16S_{\max}.$$

Here, A_{boot} is the *bootstrap operating treasury*: it is governance-controlled and available from genesis, but subject to strict epoch-level spend caps. By contrast, A_{res} is a *strategic reserve / insurance tranche* that becomes available gradually:

$$A_{\text{res}}(T) = \begin{cases} 0, & T < 6, \\ 0.16S_{\max} \cdot \dfrac{T-6}{60}, & 6 \le T \le 66, \\ 0.16S_{\max}, & T > 66. \end{cases}$$

Thus the treasury's *available* upper bound at month T is

$$A_{\text{treasury}}(T) = A_{\text{boot}} + A_{\text{res}}(T).$$

This design allows measured early spending for verifier subsidies, monitoring, and bootstrap programs, while keeping the bulk of the reserve unavailable for immediate discretionary use. As before, the circulating contribution $R_{\text{treasury}}(T)$ depends on actual governance-approved spending and can be strictly smaller than $A_{\text{treasury}}(T)$.

Illustrative milestones for $S_0 \to S_{\max}$ (upper bound). Combining time-based vesting, community cap schedules, and treasury availability yields the following indicative milestones for the *maximum unlocked/available* supply (not necessarily circulating):

Time since genesis	**Max unlocked / available share**	**Main drivers**
0 months	$\approx 10\%$	Liquidity deployment, bootstrap community distribution, and bootstrap operating treasury.
6 months	$\approx 13\%$	Ongoing Year-1 community streaming; investor vesting begins at $T = 6$.
12 months	$\approx 24\%$	Investor vesting, Year-1 community release, and early availability of the strategic reserve.
24 months	$\approx 53\%$	Team vesting, investor vesting, community releases through Year 2, and continued strategic-reserve availability.
48 months	$\approx 83\%$	Team fully vested, investors fully vested, community releases through Year 4, and strategic reserve largely available.
96 months	100%	Community allocation fully released and all vesting/availability schedules completed.

Table 3: Illustrative upper bound on unlocked/available supply (not necessarily circulating).

Programmatic release caps for community incentives. Let epochs be indexed by t (e.g., weekly or monthly). Define the community-incentive sub-cap in epoch t as

$$\mathrm{SpendCap}_t^{\mathrm{comm}} := \omega_{\mathrm{comm}} \mathrm{SpendCap}_t,$$

where $\omega_{\mathrm{comm}} \in (0, 1]$ is governance-set. Let $\overline{U}_{\mathrm{epoch}}$ denote a hard per-epoch release cap enforced by the IncentivesController contract. Then the cumulative release R_{comm} must satisfy

$$R_{\mathrm{comm}}(t+1) - R_{\mathrm{comm}}(t) \leq \min\{\mathrm{SpendCap}_t^{\mathrm{comm}},\ \overline{U}_{\mathrm{epoch}}\}.$$

Optional category sub-caps may further restrict releases (e.g., long-tail vs. verifier subsidies vs. bug bounties).

6.4.2 Unlock schedule summary

Bucket	**Cliff**	**Vesting**	**Release control**	**On-chain lock**
Team & contrib.[a]	12m	24m linear	time-based	yes (vesting)
Investors/strategic[b]	6m	24m linear	time-based	yes (vesting)
Community incentives[c]	none	programmatic	cap per epoch	partial
Bootstrap operating treasury[d]	none	none	governance / spend cap	yes (treasury)
Strategic reserve / insurance[e]	6m	60m linear availability	governance	yes (timelock)
Liquidity/MM[f]	optional	policy-based	governance/Labs	optional

[a] Standard founder/contributor vesting; predictable S_{circ} increase.
[b] Predictable release; mitigates near-term sell pressure.
[c] Released only via approved programs; $\overline{U}_{\mathrm{epoch}} + \mathrm{SpendCap}$ bounds.
[d] Immediately usable from genesis, but only under strict epoch caps and approved programs.
[e] Not circulating unless deployed; emergency and insurance uses face higher scrutiny.
[f] Released gradually; should be bounded and transparently reported.

Table 4: Unlock and release schedule overview (policy-level; subject to governance).

6.5 Value accrual and mandatory demand

A sustainable token economy requires non-speculative demand. In \$3SAT Coin, \$3SAT is a **work-permission and accountability asset** that secures protocol correctness via collateral-at-risk (Section 4).

- **Verifier staking:** verifiers stake s_V \$3SAT; slashable upon successful challenges.
- **Attestation bond:** per-attestation bond b_V discourages spam.
- **Solver submission bond:** solvers lock b_S at commit; the bond is slashed for non-reveal, inconsistent reveal, or invalid solutions.
- **Challenge bond (optional):** challengers post b_C to reduce dispute spam.

Fees and slashing penalties are routed to burn and/or the treasury (Table 1), linking protocol usage to long-run token value.

Comparable utility patterns (examples). $3SAT's utility follows a common "stake-secures-work" pattern observed in networks where service provision is gated by collateral and misbehavior is punishable. Examples include The Graph (indexers stake to provide indexing/query services) and Filecoin (storage providers post collateral and face penalties), where economic roles require collateral-at-risk. The key shared mechanism is that *economic roles require collateral-at-risk*, creating non-speculative demand tied to protocol activity.

Bootstrap security and the early-stage value problem. In early stages, endogenous $3SAT demand may be insufficient to support a deep market price, which would weaken the real-dollar value of $3SAT-denominated collateral if left unaddressed. The protocol therefore does *not* rely on token price discovery alone at launch. Instead, bootstrap security is supported by a bundle of transitional mechanisms:

- **Exposure caps:** early bounties are subject to maximum escrow tiers and stricter size-based admission, so protocol risk does not outrun security capacity.
- **Price-indexed collateral floors:** required $3SAT bonds/stakes are calibrated against a governance-set external value target (e.g., a conservative TWAP-based reference), so the required quantity of $3SAT rises if $3SAT's market price falls.
- **Multi-asset work subsidies:** solver/verifier/challenger rewards may be topped up in USDC/ETH during bootstrap, while *slashable security collateral remains $3SAT-denominated*.
- **Phased verifier admission:** bootstrap may begin with a curated or higher-threshold verifier set before opening to broader participation as dispute history, liquidity, and participation depth improve.

These measures are explicitly transitional and are intended to prevent a low-price / low-security / low-activity spiral. As protocol activity deepens, endogenous $3SAT demand from staking, bonds, fees, and governance should become the dominant support for token value.

6.6 Denomination policy

Hard security flows (stake/bonds) are $3SAT-only. In bootstrap and recovery periods, user-facing *rewards/subsidies* may be multi-asset (e.g., USDC/ETH/$3SAT) to reduce adoption friction, while governance commits to a steady-state convergence.

Steady-state commitment. In steady state, issuer-posted bounty rewards, protocol fees, and verifier/challenger rewards settle in $3SAT, and stable-asset subsidies are permitted only under explicitly time-bounded bootstrap/recovery programs approved by governance with a sunset clause (Section 8).

6.7 Rewards

This subsection summarizes *who earns what* in steady state; exact amounts and splits are determined by bounty parameters and governance-set fee rules (Section 3.3) and enforced by the lifecycle/state machine (Section 4).

Bounty issuer. Issuers deposit rewards into escrow at bounty creation, locked until a valid solution is finalized. Posting fees or refundable deposits may be phase-based to support adoption (Section 8).

Solver. The first solver to reach finality for a valid solution receives the escrowed reward, subject to commit–reveal rules and slashing conditions (Section 4, Section 4.8). In bootstrap, additional solver incentives may be provided from the community incentives bucket under spend caps (Section 7).

Verifier. Verifiers earn verification rewards and may receive a share of protocol fees, with eligibility gated by staking/bonding. Incorrect attestations can be penalized via slashing upon successful challenges (Section 4, Section 4.8).

Challenger (optional role). Successful challengers receive rewards sourced from slashing proceeds and/or capped treasury top-ups, while challenger bonds deter frivolous disputes (Section 4.8, Section 7).

7 Treasury and Governance

7.1 Treasury purpose and accounting

The treasury accumulates a fraction of fees and slashing penalties (Table 1) and serves as a security and growth budget. Treasury balances and flows are on-chain and auditable. Spending is executed only through governance-approved transactions subject to timelock.

The **genesis Treasury / DAO reserve** in Table 2 is an initial stock allocation, not a hard cap on future treasury balances. After launch, treasury balances evolve according to

$$\mathrm{TreasuryBalance}_{t+1} = \mathrm{TreasuryBalance}_t + \mathrm{FeeInflows}_t + \mathrm{SlashingInflows}_t - \mathrm{TreasurySpending}_t.$$

Accordingly, treasury balances may be above or below the genesis reserve at different points in time. This does not change total supply, because fee/slashing inflows are transfers within a fixed-supply token system rather than new issuance.

7.2 Permitted uses

Treasury outflows are restricted to:

1. long-tail bounty subsidies,
2. verifier subsidies,
3. challenge rewards / monitoring incentives,
4. security reserve / insurance (stricter approval),
5. ecosystem grants (tooling, audits, integrations),
6. protocol team operations (Labs) during bootstrap under explicit caps.

7.3 Governance process: proposal lifecycle and voting modality

This protocol adopts a *timelocked execution* model for all material actions, including treasury spending and parameter changes. The governance stack is designed to support a pragmatic bootstrap phase (lower operational friction) while maintaining a clear migration path to fully decentralized, on-chain governance.

Bootstrap governance: off-chain signaling + on-chain execution. During bootstrap, governance decisions are formed via *off-chain voting* (e.g., Snapshot-style) to reduce participation friction and gas costs, while *execution remains on-chain and timelocked.* Concretely:

- **Voting layer (off-chain):** token-weighted voting where results are publicly auditable (signed votes, verifiable snapshot of voting power).
- **Execution layer (on-chain):** a *timelock contract* queues successful proposals, after which an execution account (initially a Labs-controlled multisig with published signers and a sunset clause) submits the on-chain transactions.
- **Binding rule:** the execution account *must* execute the exact transactions committed in the proposal payload (or a hash thereof) after the timelock delay, unless an explicit emergency veto condition is triggered (see below).
- **Scope restriction (important):** the multisig is *not* authorized to move funds or change parameters outside the timelock. Its sole power is to call timelock execution for queued payloads.

Steady-state governance (target): on-chain proposals + on-chain execution. After readiness criteria are met (Section 7.6), governance transitions to on-chain proposal creation and voting (e.g., Governor + Timelock). In steady state:

- **Proposal creation, voting, and queueing** are performed on-chain.
- **Execution** occurs through the timelock, ensuring a non-zero delay for review and risk monitoring.
- **Emergency controls (if any)** are strictly limited in scope, time-bounded, and transparently logged.

Proposal types. To address common treasury and security needs, proposals are categorized into a small set with distinct thresholds and safeguards:

1. **Treasury spending proposals:** subsidies, grants, monitoring incentives, verifier programs.
2. **Protocol parameter proposals:** timing windows, bond/stake requirements, quorum rules, fee routing.
3. **Security-reserve proposals:** drawdowns from insurance/security reserve (higher quorum; stricter timelock).

4. **Code upgrade proposals (if upgradeable):** contract upgrades or module enablement (highest scrutiny).

Proposal lifecycle. Each proposal follows a standardized lifecycle to reduce governance ambiguity:

1. **Draft:** proposer publishes a structured template, including an execution payload (or its hash) and a budget bound.
2. **Discussion:** a minimum public discussion period is recommended to collect risk feedback (duration set by governance policy).
3. **Vote:** token holders vote (off-chain in bootstrap; on-chain in steady state).
4. **Queue:** if passed, the proposal is queued in the timelock with a delay.
5. **Execute:** transactions are executed exactly as specified.
6. **Report:** within a fixed time after execution, the proposer (or program steward) publishes an outcome report against pre-committed metrics.

Standard proposal template (required fields). To make treasury decisions reproducible and auditable, proposals *should* include:

- **Objective and rationale:** why the action is needed; which failure mode it addresses.
- **Budget:** max amount and time window; fit within SpendCap.
- **Eligibility rules:** who/what qualifies; anti-sybil constraints.
- **Mechanism:** payout rule, matching formula, caps, and settlement asset(s).
- **Risks and mitigations:** attack vectors (sybil, wash participation, collusion) and safeguards.
- **Success metrics:** measurable KPIs and an automatic expiry (*sunset*) / renewal condition.
- **Execution payload:** concrete on-chain calls or a hash-committed payload for timelock execution.

7.4 Long-tail bounty subsidies: scope, eligibility, and DAO decision rules

Long-tail bounty subsidies are intended to support *low-reward or socially valuable* SAT *instances* that are likely to be undersupplied by solvers in a purely market-driven equilibrium. This improves market coverage, robustness, and the protocol's public-good footprint, while preserving a path to sustainability under SpendCap. The protocol supports two subsidy modes:

Mode A: instance-specific top-up (by bountyId). A proposal can target a specific bounty bountyId and allocate a *top-up* amount to increase the effective reward.

Mode B: programmatic subsidies for a class/type of instances. A proposal can create a *Subsidy Program* that applies automatically to any bounty satisfying explicit criteria ("type of instances").

Eligibility: objective criteria and category taxonomy

A subsidy proposal *must* define eligibility using verifiable signals. Recommended taxonomy:

1. **Benchmark/Public-good category:** instances tied to widely used benchmarks or research/public datasets.
2. **High-effort category:** instances whose expected solving effort is high according to pre-specified difficulty metrics, such as clause/variable counts, benchmark-family statistics, or a standardized ex ante hardness score estimated from reference solver runs.
3. **Diversity/coverage category:** instance families that increase solver diversity and reduce concentration on only high-reward tasks.
4. **Reliability/Security category:** instances important for audits, verification tooling, or protocol safety monitoring.

To ensure auditability, eligibility metadata *should* be content-addressed (e.g., a CID) and its hash stored or referenced on-chain.

Subsidy sizing rule (example implementation)

Let R_I be the issuer-posted reward (escrowed), and let $R^\star$ be a target effective reward for the eligible category. Define the subsidy U as:

$$U \;=\; \min\Big\{\overline{U},\;\; \theta\, R_I,\;\; \max(0,\, R^\star - R_I)\Big\},$$

with program-level epoch cap:

$$\sum_{\text{eligible bounties in epoch } t} U \;\le\; \mathrm{SpendCap}_t,$$

and optional sub-caps per category.

Denomination and adoption backstop

During bootstrap/recovery, the DAO *may* authorize subsidies in stable assets (e.g., USDC), while keeping *collateral-at-risk* \$3SAT-denominated. Each program must specify an automatic expiry and/or milestone-based termination condition, so that the protocol converges toward \$3SAT-only settlement in steady state.

Anti-abuse controls

A subsidy proposal *must* include at least:

- **Issuer skin-in-the-game:** require $R_I \geq \underline{R}$ to qualify.
- **Per-issuer and per-address caps:** cap subsidized volume per issuer per epoch; cap solver/verifier subsidy earnings per epoch.
- **Collateral gating:** solvers/verifiers must still post the required bonds/stakes; subsidies must not weaken security constraints.
- **Clawback linkage to disputes:** if an outcome is overturned, subsidy payouts *should* be netted/clawed back when feasible, or future subsidies reduced.
- **Transparency:** program parameters and payouts are on-chain auditable; off-chain metadata is hash-committed.

Program stewardship, reporting, and renewal

For Mode B, the DAO may appoint a *program steward* with strictly bounded authority, subject to budget ceilings, transparent logs, periodic reporting, and automatic expiry unless renewed. Suggested KPIs:

- solver participation on eligible instances,
- time-to-first-valid-solution and finality latency,
- dispute/challenge rate and reversal rate,
- subsidy efficiency and crowd-in effect on issuer rewards.

7.5 Budget cap and sustainability rule

For routine programs, let T_t^{op} denote the operating treasury balance available at the start of epoch t. This includes available bootstrap operating funds and operating inflows from fees and slashing. Enforce:

$$\mathrm{SpendCap}_t \;=\; \min\left\{\kappa\, T_t^{\mathrm{op}},\; \overline{B}\right\},$$

with roll-over of unused budget. Drawdowns from the strategic reserve / insurance tranche require a separate proposal type, higher quorum, and/or stricter timelock, and should not be used for routine subsidy programs except under explicitly defined emergency or recovery conditions.

7.6 Governance migration: Labs $\to$ DAO

Migration is milestone-based rather than time-based. Readiness criteria may include:

- sustained verifier participation and quorum formation rates,
- stable dispute outcomes and low reversal rate,

- sufficient token distribution across non-team holders,
- recurring protocol fee revenue reducing reliance on subsidies,
- operational maturity of monitoring and incident response.

The bootstrap multisig has an automatic expiry (*sunset*) clause: once criteria are met (or after a hard deadline), the multisig's authority is revoked and on-chain governance becomes the sole execution authority.

7.7 Execution safeguards

Timelock protection, role separation, and limited-scope emergency controls (with explicit scope limits and automatic expiry clauses) protect users from rushed parameter changes or treasury misuse.

8 Bootstrap and Early-Stage Subsidization Policy

8.1 Three-sided market bootstrapping

The protocol is a three-sided market (issuers, solvers, verifiers). Bootstrapping policies attract early participation while preserving a path to steady-state sustainability.

8.2 Principle: early multi-asset UX, $3SAT-only security

In early stages, rewards/subsidies may be denominated in USDC/ETH/$3SAT to reduce adoption friction, while mandatory collateral-at-risk remains $3SAT-denominated. Governance progressively shifts user-facing flows to $3SAT-only as traction metrics are met.

8.3 Protocol-seeded bounties and onboarding campaigns

The treasury (or Labs during bootstrap) may post a curated set of protocol-seeded bounties with tiered rewards to:

- create visible tasks and activity,
- incentivize running reference solver tooling,
- generate early data (solve latency, dispute rate) for calibration (Section 5.5),
- establish early solver/verifier participation before endogenous $3SAT demand is sufficiently deep.

8.4 Security bootstrapping and exposure control

Because $3SAT demand and token price may be thin at launch, bootstrap policy includes explicit security guardrails:

- capped bounty tiers in early phases,

- stricter collateral requirements for larger bounties,
- optional verifier admission thresholds or curated verifier sets,
- stable-asset top-ups for work incentives while preserving $3SAT-only slashable collateral.

These rules are transitional and should be relaxed only as participation depth, dispute history, and market liquidity improve.

8.5 Anti-sybil and abuse controls

Bootstrap programs must include:

- per-address and per-epoch reward caps,
- collateral gating (stake/bonds remain required),
- objective eligibility rules based on on-chain signals where feasible,
- clawback/netting linkage if outcomes are overturned via disputes.

8.6 Automatic expiry (*sunset*) and steady-state transition

All bootstrap programs are time-bounded, budget-bounded, or milestone-bounded, and automatically *expire* (*sunset*) unless renewed under SpendCap. As organic issuer activity and fee revenue grow, protocol-seeded bounties taper and shift toward matching subsidies or reduced discretionary rewards. In steady state, fees and rewards settle in $3SAT, and stable-asset subsidies occur only under explicitly time-bounded, governance-approved recovery programs.