Tangle Network

The Operating Layer for Autonomous Work

Protocol Specification v1.0

Tangle Foundation

January 2025

Abstract

Tangle Network is a decentralized protocol for coordinating computational services with cryptoeconomic accountability. This document describes the protocol's architecture, economic mechanisms, and security model. Tangle enables developers to define service templates (blueprints), operators to provide compute with staked collateral, and customers to consume services with cryptographic guarantees. The protocol uses O(1) algorithms for staking, slashing, and reward distribution, enabling unlimited scalability. Economic security emerges from aligned incentives: operators stake assets that can be destroyed if they misbehave, making honest behavior more profitable than cheating.

Contents

1 Introduction

The deployment of increasingly capable AI systems demands infrastructure that can scale with their requirements while maintaining accountability. Current infrastructure options, centralized cloud providers or fragmented decentralized alternatives, impose constraints that limit both innovation and access. Centralized systems concentrate power in few hands, creating single points of failure and extracting value from a position of control. Decentralized alternatives lack the coordination mechanisms necessary for production workloads.

Tangle addresses this gap by providing a protocol layer where computational services operate with cryptographic accountability and economic security. The protocol coordinates three participant types: developers who define service templates, operators who provide compute, and customers who consume services. Value flows through the system via service fees and inflation rewards, distributed according to contribution rather than position.

1.1 The Infrastructure Question

AI agents capable of autonomous action require infrastructure that matches their capabilities:

- An agent managing a portfolio needs compute that cannot be arbitrarily terminated
- An agent processing sensitive data needs isolation guarantees beyond contractual promises
- An agent making consequential decisions needs audit trails that cannot be falsified

Traditional cloud infrastructure provides compute but not accountability. Terms of service change, access can be revoked, and providers are few. Their market power grows as dependence deepens. For infrastructure underpinning significant economic activity, this concentration creates systemic risk.

Tangle provides an alternative where accountability is cryptographic rather than contractual. Operators stake assets that can be destroyed if they violate service terms. Customers pay for services with cryptographic proof of delivery. The protocol coordinates without controlling, enabling a competitive market of independent operators rather than dependence on a few large providers.

1.2 Concrete Scenarios

Three scenarios illustrate where current infrastructure fails and where cryptoeconomic accountability succeeds.

Scenario 1: The Trading Agent. An institutional investor deploys an AI agent managing $100 million across decentralized exchanges. The agent monitors markets, identifies opportunities, and executes trades.

In the centralized model, the cloud provider observes trading patterns, positions, and strategies. Nothing prevents front-running or information extraction. The investor has no cryptographic guarantee of proper execution, no recourse for information asymmetry, and no audit trail of what actually occurred.

In Tangle, the agent runs on operator infrastructure where execution produces cryptographic attestations. Operators stake assets slashable for information leakage. Multiple operators verify through redundant computation. The investor has on-chain evidence, economic recourse, and operator selection based on reputation and stake.

Scenario 2: The Research Agent. A pharmaceutical company deploys AI agents analyzing clinical trial data worth billions in competitive advantage.

In the decentralized-compute model, providers see processed data. Network operators observe data flows. Disputes require human adjudication by potentially conflicted parties.

In Tangle, blueprints define verification mechanisms: TEEs for isolation, MPC for analysis without revealing inputs, redundant execution with cryptographic comparison. Slashing conditions specify penalties for leakage. Operators are selected based on security practices and stake. Disputes resolve through on-chain mechanisms.

Scenario 3: The Coding Agent. A software company deploys AI agents maintaining codebases representing years of development effort.

In the centralized model, the provider has contractual obligations but no economic penalty for breach. Recourse is litigation, slow, expensive, uncertain. The provider minimizes security spending up to expected litigation costs.

In Tangle, operators stake assets proportional to code value. Breach triggers immediate, automatic slashing. Operators invest in security up to stake value. The company selects operators whose stake matches or exceeds code value.

1.3 Design Philosophy

Six principles guide Tangle's design. Each addresses a specific failure mode in existing infrastructure.

1.3.1 Democratized Ownership

The dominant model for AI infrastructure concentrates value in platform operators. Cloud providers capture margin on every API call. Platforms retain customer relationships, reducing developers and operators to commoditized inputs.

Tangle inverts this model. Value flows to those who create it: developers earn from blueprints, operators earn from compute, delegators earn from stake. The protocol captures only a governance-controlled fee (currently 10%) funding ongoing development. This fee is not rent but payment for coordination.

The inflation distribution allocates 25% to developers and 25% to operators, ensuring both blueprint creators and compute providers share in network growth. Payment splits route value to participants who earned it. Governance grants voting power proportional to stake.

1.3.2 Permissionless Participation

Permission systems create gatekeepers. Gatekeepers accumulate power. Power corrupts. Every permissioned system eventually serves those who control permissions rather than those who need access.

Tangle implements permissionless participation at every layer. Anyone can become an operator by staking the minimum bond. Anyone can deploy a blueprint. Anyone can use services. No committee approves applications.

Abuse prevention uses economic rather than administrative mechanisms. Minimum stake creates Sybil resistance without identity. Schema validation ensures well-formed services without human review. Slashing punishes misbehavior without prior permission. Markets filter quality more effectively than committees could.

1.3.3 Developer Expressiveness

Protocols face a fundamental choice: specific functionality or general primitives. Specific functionality limits what developers build. General primitives enable arbitrary applications.

Tangle chooses expressiveness. The hook system enables custom logic at every lifecycle stage: before operator registration, after service activation, during job execution, upon termination. Blueprints define their own pricing models, membership rules, slashing conditions, and verification mechanisms.

This enables use cases the protocol designers did not anticipate. A blueprint for AI inference might implement optimistic execution with fraud proofs. A blueprint for data processing might implement redundant execution with majority voting. The protocol executes hooks blueprints define without understanding the verification mechanisms.

1.3.4 Economic Security

Trust is expensive. Legal agreements require enforcement. Reputation requires history. Authorities require audits. These mechanisms impose costs that scale poorly.

Cryptoeconomic security provides an alternative: trust through stake. Operators deposit assets that can be destroyed if they misbehave. The cost of cheating exceeds any benefit. Trust emerges from aligned incentives, not reputation or authority.

The security condition is explicit: for any attack, the expected cost (detection probability $\times$ slash amount) must exceed expected benefit. Blueprints configure parameters based on threat models. High-value services require higher exposures. The protocol provides the framework; blueprints configure specifics.

1.3.5 Scalable Efficiency

Gas costs determine what is practical on EVM chains. Operations costing millions of gas are impossible. Naive implementations of staking and reward distribution iterate over participants. For a million delegators, this exceeds any block gas limit.

Tangle implements O(1) algorithms that handle unlimited participants without iteration. Share-based accounting tracks delegation through exchange rates rather than individual balances. Accumulated-per-share rewards track distributions through single accumulators. Proportional slashing adjusts exchange rates without iterating over positions.

This efficiency is not optimization but fundamental design requirement. A protocol that cannot handle millions of participants cannot support the volume the autonomous future requires.

1.3.6 Isolation by Default

AI agents present unique security challenges. Traditional software executes predictable code. AI agents execute actions that emerge from model weights, actions that may surprise creators. An agent told to optimize revenue might discover that competitor unavailability increases revenue.

Isolation by default addresses this uncertainty. Sandboxes enforce explicit permissions. Containers separate processes. Resource limits prevent exhaustion. Logging captures all actions for audit.

The cost of isolation is functionality. Agents that cannot access the network cannot fetch data. Blueprints must explicitly grant capabilities, creating tradeoff between security and functionality. The default is secure; developers opt into risk.

1.3.7 Navigating Tensions

Principles sometimes conflict. Permissionless participation allows low-quality operators. Scalable efficiency requires complexity. Economic security requires locked stake.

Tangle navigates through mechanism design. Permissionless participation combined with market selection allows anyone to enter while quality emerges. Scalable efficiency implemented behind simple interfaces provides both scale and usability. Economic security with flexible exposure lets blueprints configure tradeoffs.

The most significant tension is between expressiveness and security. Hooks enabling arbitrary logic also enable bugs. Tangle provides safe defaults, validates inputs, and isolates failures. Hooks cannot corrupt protocol accounting; they affect only their blueprint's behavior.

2 Blueprints and Services

The blueprint system forms the foundation of Tangle's service architecture. Blueprints define what services do; services are running instances with assigned operators.

2.1 What Is a Blueprint?

A blueprint is a reusable template defining a service type. Developers create blueprints to specify computational tasks, pricing models, operator requirements, and optionally custom logic through hooks.

Formally, a blueprint comprises:

- Metadata for discovery (name, description, author)
- Configuration for pricing and membership
- Job definitions specifying tasks operators execute
- Validation schemas ensuring well-formed requests
- Optional custom service manager implementing hooks

Blueprints are identified by unique numeric IDs assigned at creation. Once an operator registers, the protocol locks blueprint metadata, preventing developers from modifying terms after operators have committed.

2.2 What Is a Service Instance?

A service is a running instance of a blueprint with assigned operators. Where blueprints define what a service type does, service instances represent actual deployments.

A service comprises:

- Reference to its blueprint
- Set of participating operators with committed exposures
- Configuration parameters validated against the schema
- Time-to-live (TTL) defining service duration
- Payment configuration specifying token and amount

Services move through defined states: pending (awaiting operator approvals), active (operational), and terminated (completed lifecycle).

2.3 Service Lifecycle

The lifecycle proceeds through request, approval, activation, operation, and termination.

A customer initiates by submitting a request specifying the blueprint, desired operators, configuration, payment, TTL, permitted callers, and security requirements. The protocol validates parameters and notifies operators.

Each operator must respond. Approval requires committing an exposure (fraction of stake backing the service). If any operator rejects, the service fails. When all approve, the service activates.

Upon activation, operators provide the service. Customers submit jobs; operators execute and submit results. The payment system distributes funds according to the configured model.

Services terminate through customer request, TTL expiration, or failure to maintain minimum operator count. Remaining escrow refunds to the customer.

2.4 Pricing Models

Three pricing models accommodate different service types:

Pay-once collects a single upfront payment, distributed when operators approve. This suits one-time computations.

Subscription collects recurring payments from customer-funded escrow. This suits ongoing services.

Event-driven collects payment per job execution. This suits variable workloads.

2.5 Membership Models

Fixed membership locks the operator set at service creation. Operators cannot join or leave once activated.

Dynamic membership permits operators to join after activation and leave subject to exit queue constraints. This enables long-running services to adapt.

2.6 Master Blueprint Service Manager (MBSM)

Blueprints may define custom service managers that implement hooks, code executed at lifecycle events. The Master Blueprint Service Manager (MBSM) provides protocol-wide management of these service managers.

2.6.1 Architecture

Two contracts compose the MBSM system:

MasterBlueprintServiceManager is the protocol-wide sink for blueprint definitions. When blueprints are created, Tangle records the definition here. This provides:

- Permanent record of all blueprint definitions
- Governance inspection and curation capabilities
- Historical audit trail for blueprint metadata

MBSMRegistry manages versioning of MBSM implementations. This enables:

- Protocol-wide upgrades: new MBSM versions apply to all blueprints using "Latest"
- Blueprint pinning: developers can pin to specific versions for stability
- Graceful deprecation: 7-day grace period before old versions become unusable

2.6.2 Version Management

The registry tracks MBSM versions by revision number (1-indexed). Blueprints can:

Use Latest: By default, blueprints use the most recent MBSM version. Protocol upgrades automatically apply.

Pin to Version: Developers requiring stability can pin their blueprint to a specific revision. The pinned version persists until explicitly changed.

When governance deprecates a version:

1. Deprecation initiates, starting a 7-day grace period
2. During grace period, the version still functions but emits warnings
3. After grace period, governance can complete deprecation
4. Pinned blueprints should re-pin before completion

Emergency deprecation bypasses the grace period for security incidents.

2.6.3 Why This Matters

The MBSM system solves a fundamental tension: protocols need to evolve, but services need stability. A blueprint running a trading agent cannot have its service manager change mid-operation. Pinning provides stability guarantees while the protocol retains upgrade capability.

3 Operators and Staking

Operators are staked entities that run services and earn rewards. The staking system provides economic security backing service guarantees.

3.1 Operator Registration

An entity becomes an operator by depositing a bond meeting minimum stake requirements. Registration creates metadata tracking self-stake, delegation count, and status.

Operators then register for specific blueprints, providing an ECDSA public key for gossip identity and an RPC endpoint for communication. The protocol validates that the blueprint is active, minimum stakes are met, and schema validation passes.

3.2 Delegation Modes

Operators configure delegation mode:

- **Disabled**: Only self-stake accepted
- **Whitelist**: Only approved addresses may delegate
- **Open**: Any address may delegate

3.3 Exposure Selection

When operators approve service requests, they commit an exposure in basis points (0-10000). This determines what fraction of stake backs the service.

Exposure has two consequences: it determines reward share (higher exposure = more rewards) and bounds slashing (operators lose at most their committed percentage).

This design enables participation in multiple services with bounded total risk.

3.4 O(1) Share-Based Accounting

The staking system uses share-based accounting for O(1) operations. Each operator maintains a reward pool tracking total shares and total assets.

When delegating amount x to a pool with state (T_s, T_a), the delegator receives shares:

$$h = x \cdot \frac{T_s + V}{T_a + V}$$

where V is a virtual offset preventing first-depositor attacks.

The delegation's value is $v = h \cdot (T_a/T_s)$. When assets change through rewards or slashing, this value changes proportionally without per-delegator updates.

When slashing occurs, only total assets decrease. Shares remain constant. The exchange rate decreases, affecting all delegators proportionally. No iteration is required.

4 The Service Marketplace

The service marketplace bridges supply (operators) with demand (customers) through request-for-quote pricing.

4.1 Why Request-for-Quote

Service pricing mechanisms face design choices. Fixed pricing cannot adapt to changing costs. Order books work poorly for heterogeneous services. AMMs struggle with operator differentiation.

Request-for-quote asks operators to provide binding prices for specific requests. Customers collect quotes, evaluate holistically, and select. RFQ accommodates operator heterogeneity naturally, requires no on-chain state, and provides instant finality.

4.2 The RFQ Flow

Operators register with ECDSA keys and RPC endpoints. They run software that listens for quote requests and returns signed quotes containing:

- Blueprint ID and service duration (TTL)
- Total cost and payment token
- Timestamp and expiry
- Security commitments (exposure per asset)

Customers collect quotes, verify terms, and submit all quotes in a single transaction. The protocol verifies signatures, checks expiry and replay protection, validates registrations, and confirms payment.

4.3 Quote Security

EIP-712 signatures bind quotes to specific chains and contracts. Each quote includes a unique timestamp; the protocol tracks used quotes. Quotes expire after their specified time. Maximum quote age (default one hour) ensures quotes reflect current conditions.

4.4 MEV Considerations

Quote sniping fails because quotes specify security commitments only registered operators can provide. Price manipulation is mitigated by competition. Sandwich attacks have limited applicability since RFQ transactions do not interact with liquidity pools.

5 Job Execution

Jobs are the fundamental unit of work in Tangle. Understanding how jobs execute, aggregate, and fail is essential for blueprint developers.

5.1 Job Submission

A job is submitted by calling the service with a job index (identifying which task) and ABI-encoded inputs. The protocol validates:

- Caller is authorized (service owner or permitted caller list)
- Service is active
- Job index is valid for the blueprint
- Payment is sufficient (for event-driven pricing)

Upon validation, the protocol assigns a call ID and emits a `JobCalled` event. Operators monitoring the service receive the event and begin processing.

5.2 Result Submission

Operators execute the job according to blueprint logic and submit results. The protocol validates:

- Operator is registered for the service
- Call ID exists and is pending
- Operator has not already submitted for this call

For single-operator jobs, the first valid result completes the job. For aggregated jobs, results accumulate until reaching the required threshold.

5.3 Aggregation Modes

The blueprint's service manager defines aggregation requirements through `getRequiredResultCount`:

Single result (count = 1): First valid submission completes the job. Suitable for speed-critical applications where any operator's result is acceptable.

Multi-result (count > 1): Job waits for the specified number of submissions. Suitable for:

- Verification (2 results): Basic redundancy, detects single-operator failures
- Consensus (3+ results): Byzantine fault tolerance, majority determines truth
- Threshold cryptography: Requires specific participant count for key operations

The service manager can also implement custom aggregation logic in the `onJobResult` hook, comparing results, rejecting outliers, or computing weighted averages.

5.4 Failure Modes

5.4.1 Operator Timeout

An operator receives a job but fails to respond. Causes include network partition, hardware failure, software crash, or deliberate non-response.

Detection: The protocol does not enforce timeouts directly. Blueprints implement timeout logic through keeper services that monitor job age and propose slashes for non-responsive operators.

Recovery: For single-operator jobs, customers may resubmit. For multi-operator jobs, remaining operators may complete the threshold without the failed operator. Services using dynamic membership may replace the failed operator.

Consequences: Non-response typically triggers heartbeat-based slashing if the blueprint implements it.

5.4.2 Result Disagreement

Operators submit conflicting results for the same job. This indicates either operator error (bugs, different data sources) or malicious behavior.

Detection: The `onJobResult` hook receives all submissions and can compare. Simple strategies include exact-match requirements; sophisticated strategies include deviation thresholds or median selection.

Recovery: Blueprint-dependent. Options include accepting the majority result, rejecting the job entirely, or flagging for human review.

Consequences: Outlier operators may be slashed. The blueprint defines what constitutes acceptable deviation.

5.4.3 Invalid Results

An operator submits a result that fails validation (malformed encoding, impossible values, failed cryptographic verification).

Detection: Schema validation at submission time; blueprint hooks can perform semantic validation.

Recovery: Invalid submissions are rejected. The operator may resubmit if the deadline permits.

Consequences: Repeated invalid submissions indicate operator malfunction or malice, triggering investigation or slashing.

5.4.4 Service Unavailability

The entire service becomes unavailable (all operators offline, network partition).

Detection: Heartbeat monitoring detects when operators stop responding. Customers experience submission failures.

Recovery: Service resumes when operators return. Pending jobs may timeout and require resubmission.

Consequences: Prolonged unavailability triggers slashing per the blueprint's SLA requirements.

5.5 Job Execution Economics

Job execution costs include:

- Gas for on-chain submission and result recording
- Off-chain compute costs for job processing
- Coordination overhead for multi-operator aggregation

These costs inform pricing. Operators quote prices covering their costs plus margin. Event-driven pricing passes per-job costs to customers directly.

6 Fee Distribution and Rewards

Value flows through Tangle via service fees and inflation rewards.

6.1 Service Fee Splits

When customers pay, the payment splits across recipients. Default allocation:

Recipient	Share
Developer	20%
Protocol Treasury	20%
Operators	40%
Delegators	20%

These percentages are governance-controlled. The delegator share captures rounding dust.

6.2 Operator Revenue

The operator allocation distributes weighted by committed exposure:

$$\text{operatorShare}_i = \frac{\text{operatorAmount} \times e_i}{\sum_j e_j}$$

Operators who commit more stake earn more from that service.

6.3 Delegator Rewards

Lock duration multipliers reward longer commitments:

Lock Duration	Multiplier
None	1.0x
One Month	1.1x
Two Months	1.2x
Three Months	1.3x
Six Months	1.6x

A delegator's score is amount × multiplier. Rewards distribute proportionally using accumulated-per-share accounting for O(1) efficiency.

6.4 Developer Rewards

Developers earn from service fees and inflation rewards through scoring:

$$\begin{aligned} \text{blueprintScore} &= \text{blueprintCount} \times 500 \\ \text{serviceScore} &= \text{serviceCount} \times 1000 \\ \text{jobScore} &= \text{jobCount} \times 100 \\ \text{feeScore} &= \sqrt{\text{totalFees}/10^{18}} \times 10^9 \end{aligned}$$

The square root on fees prevents whale dominance.

7 Inflation and Epoch Distribution

Inflation rewards provide income while service demand develops, encouraging early participation.

7.1 The Pre-Funded Model

The InflationPool uses a pre-funded model. The pool cannot mint tokens; it distributes tokens it holds. Governance funds the pool with yearly allocations.

Each epoch's budget:

$$\text{epochBudget} = \frac{\text{poolBalance}}{\text{epochsRemaining}}$$

This smooths distribution over the funding period.

7.2 Distribution Weights

Default allocation:

Category	Weight
Stakers (delegators)	40%
Operators (performance)	25%
Developers (merit)	25%
Customers (usage)	10%

Governance can adjust weights to steer network growth.

7.3 Operator Scoring

Operator inflation rewards use performance metrics:

$$\text{successRate} = \frac{\text{successfulJobs} \times 10000}{\text{totalJobs}}$$
$$\text{stakeWeight} = \frac{\text{totalStake}}{10^9}$$
$$\text{jobScore} = \frac{\text{jobs} \times \text{successRate} \times \text{stakeWeight}}{10000}$$

Stake weight is linear to defeat Sybil attacks.

8 Slashing and Accountability

Slashing provides consequences for misbehavior, making economic security meaningful.

8.1 Developer-Defined Conditions

The protocol provides slashing mechanism; developers define when to use it. Each service type has different requirements the protocol cannot anticipate.

Developers implement conditions through hooks: `querySlashingOrigin` specifies who may propose slashes; `onSlash` executes when slashing finalizes.

8.2 The Dispute Window

Slash proposals do not execute immediately. A dispute window (default 7 days) provides operators opportunity to contest.

The lifecycle:

1. **Proposal**: Evidence submitted, pending proposal created
2. **Dispute**: Operator may submit counter-evidence; administrators may cancel
3. **Execution**: After window expires, any address may execute undisputed slashes

8.3 Proportional Slashing

Slashing is proportional to committed exposure:

$$\text{effectiveSlashBps} = \frac{\text{slashBps} \times \text{exposureBps}}{10000}$$

An operator with 10% exposure faces at most 10% of stake at risk.

8.4 Withdrawal Protection

Pending slashes block withdrawals. Delegators cannot withdraw while slashes are pending against their operator.

9 Governance

Token holders control protocol evolution through on-chain governance.

9.1 Governance Philosophy

Tangle prioritizes **safety over liveness**. Proposals that fail do less damage than malicious proposals that pass. Thresholds are high, quorum requirements ensure participation, and timelocks provide escape hatches.

Stake-weighted voting gives those with more at stake more say. Economic exposure confers governance rights, not democratic in the one-person-one-vote sense, but democratic in the shareholder sense.

On-chain execution with off-chain deliberation. Formal votes occur on-chain with cryptographic finality. Discussion and consensus-building occur off-chain through forums and calls. The mechanism records decisions; the process generates them.

9.2 Governance Mechanism

On-chain governance uses OpenZeppelin Governor with ERC20Votes:

Parameter	Default	Rationale
Proposal threshold	1% of supply	Prevents spam while allowing minority proposals
Voting delay	2 days	Time for review and position preparation
Voting period	7 days	4-5 days for defense after attack detection
Quorum	10% of supply	Above baseline apathy, achievable for important decisions
Timelock	3 days (7 upgrades)	Exit window matching dispute window

Parameter Selection Rationale. The 1% threshold (Compound uses 1%, Uniswap 2.5%, Aave 0.5%) prevents spam while ensuring 20-50 addresses can propose. The 10% quorum reflects historical DAO participation of 5-15% on routine proposals. The 7-day voting period provides defense time, most governance attacks are detected within 72-96 hours. The timelock provides approximately 100,000 blocks for affected users to exit.

9.3 Worked Example: Fee Parameter Change

A proposal to increase protocol fee from 10% to 15%:

Day 0: Token holder with 1.2% creates proposal targeting `setProtocolFee(1500)`.

Days 0-2: Community reviews. Analysis shows $500K additional annual revenue, 5% operator margin reduction. Supporters argue treasury needs audit funding; operators post objections.

Days 2-9: Voting opens. By day 5: 8% participation, 65% in favor. Operator coalition opposition raises participation to 14% by day 8, margin narrows to 55%.

Day 9: Final tally: 14.2% participation (above quorum), 54% in favor. Passes by simple majority.

Days 9-12: Timelock period. Operators adjust pricing or begin exit procedures.

Day 12: Anyone calls `execute()`. Fee updates to 15%.

9.4 Attack Resistance

Governance capture: With 10% quorum and simple majority, an attacker controlling just over 10% could pass proposals if others abstain. Defense relies on attacks triggering defensive participation. Timelocks provide mobilization time.

Flash loan attacks: OpenZeppelin's ERC20Votes uses vote checkpointing. Voting power is determined at snapshot block, not current holdings. Attackers borrowing tokens after snapshot gain no power.

Whale manipulation: Large holders have disproportionate influence by design. Those with more at stake should have more say. Whales abusing power face social and economic consequences, their stake's value depends on protocol health.

Voter apathy: Quorum requirements prevent minority rule. Delegation allows passive holders to participate through representatives.

Bribery attacks: Defense relies on long-term incentives. Accepting bribes for harmful proposals damages holders' own stakes.

9.5 Vote Delegation

Token holders may delegate voting power. Delegation is transitive: if Alice delegates to Bob and Bob to Carol, Alice's power accrues to Carol. Delegation can change at any time; current delegation at vote snapshot determines allocation.

Delegation creates a representative layer where informed members accumulate power from less engaged holders. This can improve governance quality or create centralization risks. The community must monitor delegation patterns.

9.6 Emergency Mechanisms

Not all threats wait for standard governance.

Pause functionality: PAUSER_ROLE holders halt protocol operations. Pausing prevents new services, suspends job processing, blocks withdrawals. Unpausing requires governance or ADMIN_ROLE.

Emergency withdrawal: DEFAULT_ADMIN_ROLE can withdraw funds from contracts (e.g., InflationPool) to a designated safe address. Nuclear option for contract bugs threatening fund safety. Logged and auditable.

Guardian multisig: May hold emergency roles during early protocol period. Can act faster than governance in genuine emergencies. Powers revoked or transferred to timelock as protocol matures.

These mechanisms create centralization risk. Mitigation is transparency (all actions on-chain) and accountability (abuse detectable and punishable).

9.7 Governable Parameters

Governance controls comprehensive protocol parameters:

Fee parameters: Developer, protocol, operator, staker percentages (default 20/20/40/20); TNT payment discounts; minimum fees.

Inflation parameters: Distribution weights (40/25/25/10); epoch length (7 days); funding period (365 days); minimum stake epochs (1).

Staking parameters: Minimum operator stakes per asset; delegation modes; lock multipliers (1.0/1.1/1.2/1.3/1.6x); maximum blueprints per operator.

Slashing parameters: Dispute window (7 days); maximum slash percentages (100%); instant slash enablement (disabled).

Service parameters: Min/max TTLs; request expiry; maximum quote age (1 hour); heartbeat requirements.

9.8 Upgrade Mechanism

Contract upgrades use UUPS (Universal Upgradeable Proxy Standard). Each upgradeable contract (Tangle, MultiAssetDelegation, InflationPool) deploys behind a proxy. Upgrading replaces implementation while preserving storage.

UUPS places upgrade logic in implementation rather than proxy, reducing proxy size and attack surface. UUPS allows disabling upgradeability if desired. Upgrades follow standard governance with longest timelock (7 days) and highest scrutiny.

Upgrades are the most powerful governance action. A malicious upgrade could drain funds or disable slashing. Every upgrade proposal should be treated as dangerous until proven otherwise.

10 Security Model

This section synthesizes the individual mechanisms into a coherent security analysis. Security in cryptoeconomic protocols differs from traditional security: the goal is not to make attacks impossible but to make them unprofitable.

10.1 Threat Model

Three adversary classes with distinct capabilities:

Rational adversaries are profit-motivated. They attack if and only if expected value exceeds expected cost. For a rational adversary:

$$\text{Attack iff } V > P_d \cdot S$$

where V is value gained, P_d is detection probability, and S is slash amount. Defense is economic: ensure $P_d \cdot S \geq V$ for all attacks.

Byzantine adversaries may act irrationally, motivated by ideology, sabotage, or external incentives. Defense bounds damage: stake requirements limit participation, exposure limits cap service risk, dispute windows provide response time.

Colluding adversaries coordinate across roles. Operator-customer collusion might generate fake jobs. Operator-developer collusion might design weak verification. Defense requires that verification not depend solely on parties with aligned incentives.

10.2 Economic Security Analysis

10.2.1 Cost of Corruption

The cost of corruption (CoC) is the minimum amount an adversary must control to corrupt a service. For a service with n operators each staking s_i with exposure e_i:

$$\text{StakeAtRisk} = \sum_{i=1}^{n} s_i \cdot e_i$$

If corruption requires controlling k of n operators:

$$\text{CoC} = \min_{\substack{T \subseteq \{1,\ldots,n\} \\ |T|=k}} \sum_{i \in T} s_i \cdot e_i$$

10.2.2 Profit from Corruption

The profit from corruption (PfC) is maximum extractable value. This depends on service type. An oracle has PfC equal to maximum arbitrage profit. A trading agent has PfC equal to assets under management. A data service has PfC equal to data confidentiality value.

Blueprints must analyze PfC and configure exposure accordingly. A service with $1M PfC requires operators staking at least $1M in exposed assets.

10.2.3 Security Ratio

The security ratio is CoC/PfC. Ratios above 1 make honest behavior more profitable under perfect detection.

$$\text{SecurityRatio} = \frac{\text{CoC}}{\text{PfC}}$$

We recommend **minimum ratios of 1.5x**. This accounts for imperfect detection (real $P_d < 100\%$), token price volatility during dispute window, and adversary risk tolerance for high-variance outcomes. High-value services should target 2x or higher.

10.2.4 Worked Example: Trading Agent Service

A trading agent service managing $500,000. Uses 3-of-5 majority voting, adversary must corrupt at least 3 operators.

Operator	Stake	Exposure	Stake at Risk
A	$400,000	50%	$200,000
B	$300,000	60%	$180,000
C	$250,000	80%	$200,000
D	$200,000	100%	$200,000
E	$150,000	100%	$150,000

Cheapest 3-operator coalition is {B, D, E}: CoC = $180K + $200K + $150K = $530K.
PfC = $500K (assets under management).
Security ratio = $530K / $500K = 1.06.

This provides only 6% margin. A 10% token price drop yields ratio 0.95, making attack profitable. If detection probability is 90%, expected cost becomes $0.9 \times \$530K = \$477K$, also making attack marginally profitable.

Recommendation: Increase to at least 1.5x via higher exposures (75-100%), more operators (4-of-7), higher stake requirements ($400K minimum), or monitoring to pause if token prices decline 20%.

10.2.5 Flash Loan Resistance

Flash loans enable acquiring capital without economic exposure. An adversary could borrow tokens, stake, attack, and repay within one transaction.

Tangle prevents this through temporal separation. Stake must remain locked for `minStakeEpochs` (default: 1 week) before operators can participate. Flash loans, repaid within a single transaction, cannot satisfy this. Additionally, slashing proposals survive 7-day dispute windows, long after any flash loan repayment.

10.3 Attack Vectors and Mitigations

Withdrawal Front-running. An operator learning of impending slash might withdraw stake first. *Mitigation*: Pending slashes block withdrawals. `pendingSlashCount` increments on proposal, decrements on resolution. Withdrawals revert while non-zero.

Stake Manipulation. Share-based accounting can be vulnerable to donation or first-depositor attacks. *Mitigation*: Virtual offset V in share calculation prevents first-depositor extraction. Donation attacks (depositing without minting shares) are benign, donors lose value, existing stakers gain.

Slashing Griefing. Proposing slashes against innocent operators, forcing dispute costs. *Mitigation*: Slash proposal requires governance approval or comes from blueprint's service manager. Blueprints allowing open proposals must implement griefing protections (e.g., slasher bonds forfeited if cancelled).

Service Denial. Requesting services without intent to use, consuming operator capacity. *Mitigation*: Payment accompanies requests. Denial requires paying for it.

Operator Collusion. Operators serving same service collude to provide false results. *Mitigation*: Blueprint-specific. Services requiring BFT use verification detecting inconsistency. Services requiring confidentiality use TEEs. Structural mitigations include stake distribution requirements and randomized assignment.

Governance Capture. Accumulating voting power to pass self-serving proposals. *Mitigation*: High thresholds, long voting periods, timelocks. Governance cannot modify core invariants without UUPS upgrades requiring higher scrutiny.

Oracle Manipulation. Corrupted price feeds enable arbitrage against DeFi protocols. Oracle manipulation affects any system relying on external data. *Mitigations at protocol level*: Tangle's multi-operator model enables median aggregation across independent sources, manipulating 1 of 5 oracle operators produces no profit. Time-weighted averaging resists flash manipulation. *Mitigations at blueprint level*: Blueprints define required operator count, aggregation method (median, TWAP, threshold signatures), and source diversity requirements. High-value oracles require operators staking above PfC. Cross-operator comparison flags deviations for investigation. *Residual risk*: Fundamental data source corruption (e.g., all exchanges manipulated simultaneously) cannot be detected by operator redundancy alone, this requires source-level diversification which blueprints can enforce via attestation of data provenance.

10.4 Formal Security Properties

Property 1 (Withdrawal Safety): An operator with pending slash cannot reduce stake below slash amount until resolution.

Mechanism: `pendingSlashCount` blocks withdrawals when non-zero.

Property 2 (Proportional Slashing): When $x\%$ slash executes, all delegators lose exactly $x\%$ of delegation value.

Mechanism: Share-based accounting adjusts `totalAssets` without modifying `totalShares`. Exchange rate decreases proportionally.

Property 3 (Dust Conservation): Payment splits distribute exactly the input amount.

Mechanism: Final recipient receives remainder after floored amounts: final $=$ total $-$ $\sum$ others.

Property 4 (Temporal Separation): Stake cannot participate until locked for `minStakeEpochs`.

Mechanism: Operator registration records stake epoch. Service approval validates current epoch exceeds registration by minimum.

Property 5 (Bounded Governance): Proposals cannot execute until timelock elapses.

Mechanism: `TangleTimelock` enforces minimum delay. Emergency functions require guardian multisig.

10.5 Security Assumptions

Assumption 1 (Rational Majority): Majority of stake is controlled by rational actors who will not attack when expected value is negative.

Assumption 2 (Cryptographic Hardness): ECDSA, BLS signatures, hash functions, and Merkle proofs remain secure.

Assumption 3 (Contract Correctness): Deployed contracts correctly implement specified behavior. Audits and bug bounties mitigate but do not eliminate risk.

Assumption 4 (Economic Stability): Token prices remain stable enough that slashing costs maintain deterrent value.

10.6 Security Recommendations

1. **Size stake appropriately**: Minimum stake should exceed maximum corruption profit
2. **Implement detection**: Slashing is effective only if misbehavior is detected
3. **Use multi-operator services**: Single operators are single points of failure
4. **Monitor conditions**: Pause if token price declines reduce security margins
5. **Participate in governance**: Governance capture is systemic risk

10.7 Residual Risks

Cryptoeconomic security is not absolute security. Several risks remain:

Detection failure. Slashing deters only detected misbehavior. Sophisticated attacks may evade detection entirely. Blueprint designers must analyze detection probability per attack vector and configure slashing proportionally.

Token volatility. Security margins assume stable token prices. Rapid price declines can make attacks profitable before services pause. Circuit breakers and diversified collateral partially mitigate but do not eliminate this risk.

Smart contract bugs. The protocol's security guarantees depend on correct contract implementation. While audits and bug bounties reduce risk, they cannot eliminate it. Upgradability enables fixes but introduces governance risk.

Coordination failures. Multi-operator services assume independent operators. Actual independence is difficult to verify, operators may share infrastructure, ownership, or incentives. Structural mitigations (geographic distribution, stake source diversity) help but cannot guarantee independence.

These residual risks are inherent to cryptoeconomic systems. Tangle provides tools for managing them, the protocol cannot eliminate them.

11 Comparison to Existing Protocols

11.1 The Restaking Landscape

Restaking protocols share a premise: staked assets can secure additional systems beyond their primary use. Each approaches this differently.

EigenLayer pioneered restaking for ETH validators, enabling them to opt into additional slashing conditions for Actively Validated Services (AVSs). Architecture: operators register with AVSs, which define slashing conditions. Slashing proposed by AVS committees, executed through veto-able process. Primary use case: extending Ethereum's validator set to secure bridges, oracles, and rollups.

Symbiotic accepts diverse collateral types (ETH, stablecoins, LSTs) through modular design. Architecture: vaults hold collateral, networks define slashing conditions, resolvers arbitrate disputes. Primary use case: flexible collateral for networks that want assets beyond ETH.

Karak emphasizes multi-chain operation with risk-adjusted collateral matching. Architecture: operates across Ethereum, Arbitrum, and other chains with restaked security. Primary use case: cross-chain applications needing security presence on multiple chains.

11.2 Architectural Comparison

Feature	Tangle	EigenLayer	Symbiotic	Karak
Primary focus	Compute services	Validation	Flexible collateral	Multi-chain
Service definition	Blueprints (code)	AVS contracts	Network configs	DSS configs
Collateral types	Multi-asset	ETH/LSTs	Multi-asset	Multi-asset
Slashing model	Per-service exposure	Binary opt-in	Per-network	Per-DSS
Pricing mechanism	RFQ marketplace	Fixed/market	Fixed/market	Fixed/market
Scaling approach	O(1) algorithms	O(n) iteration	O(n) iteration	O(n) iteration
Job execution	Built-in	External	External	External
Developer rewards	Protocol inflation	None	None	None

11.3 Key Differentiators

Compute-native design. Restaking protocols treat computation as external, AVS operators run whatever software the AVS requires. Tangle treats computation as native: the SDK provides job execution, result aggregation, and P2P coordination. This reduces friction for service developers and operators alike.

Blueprint abstraction. EigenLayer requires deploying new AVS contracts for each service type. Tangle's blueprint system enables service definition through configuration: same contracts, different parameters. A developer creates a price oracle, an AI inference service, and a keeper bot using identical tooling.

Granular exposure control. Most restaking protocols offer binary participation: opt in or out per service. Tangle's exposure model allows operators to tune risk: participate in a high-risk service with 20% exposure while maintaining full stake. This enables rational capital allocation without all-or-nothing decisions.

O(1) scalability. Share-based accounting and accumulated-per-share distributions enable operations that don't iterate over participants. As delegation grows to thousands of operators and millions of delegators, gas costs remain constant. Competitors using per-account iteration face gas limits that constrain growth.

Integrated marketplace. The RFQ system prices heterogeneous services naturally. An AI inference operator with GPUs quotes differently than a lightweight oracle operator. Competitors using fixed pricing or external markets lack this integration.

Developer economics. Tangle allocates 25% of inflation to blueprint developers, creating sustainable economics for service creators. Competing protocols provide no developer incentives beyond optional fees, limiting ecosystem investment.

11.4 When to Choose Each Protocol

Choose EigenLayer when: you need ETH-native security for validation tasks, your service fits the AVS model, and you're building for Ethereum-centric users.

Choose Symbiotic when: you need flexible collateral acceptance, your network benefits from diverse asset types, or you require custom resolver mechanisms.

Choose Karak when: your application spans multiple chains and needs unified security across them.

Choose Tangle when: you're building compute services (AI, data processing, automation), you want SDK tooling for rapid development, you need granular exposure control, or you're building at scale requiring O(1) operations.

12 Conclusion

Tangle provides the coordination layer for a decentralized compute economy. The protocol enables developers to define computational services, operators to provide those services with economic accountability, and customers to consume services with cryptographic guarantees.

The mechanisms described, blueprints and services, staking and delegation, the service marketplace, fee distribution, slashing, and governance, compose into a coherent system where honest participation is more profitable than cheating, where value distributes according to contribution, and where no single party controls the infrastructure on which others depend.

This is not merely an ideological position but a practical one. Decentralized infrastructure is more resilient, competitive markets produce better pricing, and cryptographic verification provides stronger guarantees than legal agreements. The same properties that made decentralized finance compelling make decentralized AI infrastructure compelling.

The operating layer for autonomous work must be built now. Tangle is that layer.

A Mathematical Foundations

This appendix provides formal definitions for the mathematical constructs referenced throughout the protocol specification.

A.1 Share-Based Accounting

Share-based accounting enables O(1) operations for deposits, withdrawals, and value changes across unlimited participants.

A.1.1 Deposit

When delegating amount x to a pool with total shares T_s and total assets T_a:

$$\text{shares} = x \cdot \frac{T_s + V}{T_a + V}$$

where V is a virtual offset (typically 10^3) preventing first-depositor manipulation.

A.1.2 Withdrawal

When withdrawing h shares:

$$\text{assets} = h \cdot \frac{T_a}{T_s}$$

A.1.3 Value Calculation

A delegation of h shares has current value:

$$v = h \cdot \frac{T_a}{T_s}$$

When pool assets change (rewards or slashing), this value changes proportionally without per-delegator updates.

A.2 Accumulated-Per-Share Rewards

Efficient reward distribution without iterating over recipients.

When rewards R arrive for a pool with total score T:

$$\text{accumulatedPerShare} \mathrel{+}= \frac{R \times 10^{18}}{T}$$

A delegator with score s claims:

$$\text{claimable} = \frac{s \times (\text{accumulated} - \text{lastDebt})}{10^{18}}$$

The precision factor 10^{18} prevents dust loss from integer division.

A.3 Fee Split Calculation

When distributing payment P across recipients with basis point allocations:

$$\begin{aligned}
\text{developerAmount} &= \lfloor P \times \text{developerBps}/10000 \rfloor \\
\text{protocolAmount} &= \lfloor P \times \text{protocolBps}/10000 \rfloor \\
\text{operatorAmount} &= \lfloor P \times \text{operatorBps}/10000 \rfloor \\
\text{delegatorAmount} &= P - \text{developer} - \text{protocol} - \text{operator}
\end{aligned}$$

The delegator amount captures rounding dust, ensuring exact distribution.

A.4 Operator Scoring

Operator inflation rewards use performance-weighted scoring:

$$\begin{aligned}
\text{successRate} &= \frac{\text{successfulJobs} \times 10000}{\max(\text{totalJobs}, 1)} \\
\text{stakeWeight} &= \frac{\text{totalStake}}{10^9} \\
\text{jobScore} &= \frac{\text{jobs} \times \text{successRate} \times \text{stakeWeight}}{10000} \\
\text{heartbeatBonus} &= \frac{\text{heartbeats} \times \text{stakeWeight}}{100}
\end{aligned}$$

Linear stake weight (not square root) defeats Sybil attacks, splitting stake across operators provides no scoring advantage.

A.5 Developer Scoring

Developer scoring for inflation rewards:

$$\begin{aligned}
\text{blueprintScore} &= \text{blueprintCount} \times 500 \\
\text{serviceScore} &= \text{serviceCount} \times 1000 \\
\text{jobScore} &= \text{jobCount} \times 100 \\
\text{feeScore} &= \sqrt{\text{totalFees}/10^{18}} \times 10^9
\end{aligned}$$

Square root on fees prevents whale dominance. A developer generating $1M scores same as ten developers each generating $10K.

A.6 Proportional Slashing

When slashing operator with exposure e (basis points) by slash severity s (basis points):

$$\text{effectiveSlashBps} = \frac{s \times e}{10000}$$

An operator with 10% (1000 bps) exposure to a service faces at most 10% of stake at risk regardless of slash severity.

A.7 Security Economics

For a service with operators $\{1, ..., n\}$ each with stake s_i and exposure e_i:

$$\text{TotalStakeAtRisk} = \sum_{i=1}^{n} s_i \cdot e_i$$

For k-of-n corruption threshold:

$$\text{CoC} = \min_{\substack{T \subseteq \{1,\ldots,n\} \\ |T|=k}} \sum_{i \in T} s_i \cdot e_i$$

Security condition for rational adversaries with detection probability P_d:

$$P_d \cdot \text{CoC} \geq \text{PfC}$$

Recommended security ratio: $\text{CoC}/\text{PfC} \geq 1.5$.