Tunnels Protocol

A Self-Governing Blockchain for Contribution Attestation and Deterministic Revenue Distribution

Aaron E. Parsons

February 2026

Abstract

Tunnels is a purpose-built blockchain for recording contributions and distributing revenue. It consists of five primitives (persistent non-transferable identity, immutable project registration, append-only contribution attestations, time-windowed attestation verification, and deterministic revenue distribution) and five invariants that cannot be configured away by any application built on top of it. The protocol does not decide what a contribution is, how it should be weighted, or how a project should be governed. It provides cryptographic guarantees about how contribution is recorded, challenged, and compensated. Everything else is left to the application layer.

Contribution history cannot be purchased or transferred. Attestations cannot be modified after finalization. Distribution cannot be overridden by discretion. Configuration cannot be changed after creation.

The protocol provides three properties that no existing system combines:

Deterministic revenue distribution tied to attested contributions, for humans and agents alike. When revenue enters a project's router, it is split mathematically based on who did what work, weighted by attested units, computed the same way every time. Revenue that never enters is not governed by the protocol, but its absence is visible on-chain.

A portable, anonymous-by-default work history. Your contribution record follows you across every application built on the protocol. It belongs to a cryptographic key you hold, not to any platform. If an application shuts down tomorrow, the record persists on-chain.

Self-policing through economic incentive, not governance. Every fraudulent attestation dilutes existing contributors, who have direct financial motivation to challenge it. The protocol requires no governing body because honest behavior is already the rational strategy.

1. Introduction

> *“I’m not a businessman, I’m a business, man.”*
> *- Jay-Z*

Every individual is a self-contained productive entity, a micro-monopoly of value. The work they do, the knowledge they carry, the creative output they generate: these cannot be replicated by anyone else in exactly the same way. This monopoly is real, but under current systems, its value must be intermediated before it can be captured.

The intermediation takes many forms: a resume filtered by an algorithm, a salary negotiation conducted with asymmetric information, an equity grant determined by factors unrelated to contribution, a credential purchased to signal competence that already exists. Each intermediary step introduces friction, and that friction compounds. The result is a deadweight loss: human

capital that creates value but cannot capture it proportionally because the signal between work and compensation is noisy.

Tunnels proposes to reduce this friction by making contribution the signal. When work is done, it is cryptographically attested. After a verification window, the attestation becomes immutable. Revenue distributes deterministically based on attested contribution weight. The contributor's identity persists across projects, accumulating a verifiable history that replaces the need for intermediary credentials. Attestation history demonstrates competence, the identity graph replaces the professional network, and deterministic distribution replaces negotiation.

Someone must still decide what a contribution is worth. The protocol eliminates the opacity around that decision. The subjective choices are made transparently, recorded immutably, and subject to challenge. Whatever system a project chooses for assessing contribution is visible, challengeable, and deterministic in its consequences.

Tunnels is implemented as a purpose-built blockchain with its own network, its own consensus, and its own nodes, because the protocol's guarantees require infrastructure that no external platform's governance can alter.

2. Design Philosophy: Protocol vs. Application

Tunnels is a protocol, not a platform.

Tunnels does not know whether a contribution was a line of code, a sales call, a design deliverable, or an act of mentorship. It records that an identity attested to a contribution on a project, with evidence, and that the attestation survived its verification window.

The protocol does not solve several hard problems:

- How to measure the relative value of different types of work.
- Who has authority to create or approve attestations.
- What governance structure a project should adopt.
- What categories of contribution are valid.
- How challenges are adjudicated beyond the binary of finalized or rejected.

These are application-layer decisions. A project-based marketplace will answer them differently than a worker-owned cooperative, which will answer them differently than a venture-backed company using the protocol for transparent compensation. All three are valid deployments of Tunnels. The protocol provides the rails; applications decide how to ride them.

What the protocol does decide, and enforces, is a set of invariants. They are not technical necessities. They are values expressed as code.

One architectural property underlies all of them. The identity graph is simultaneously what participants build, what the protocol protects, and what verifies the integrity of the system. Each honest attestation strengthens the graph. The graph is what makes fraud expensive. The system verifies itself through its own accumulated state, without external oracles, juries, or token-weighted votes. These design choices were shaped by the formal properties of multi-agent reinforcement learning environments. Three structural properties make this possible: the state space is fully observable (the graph is public, so every participant can see every other participant's full history), the feedback is binary and deterministic (finalized or rejected, with no subjective gradient to manipulate), and the reward signal is endogenous (honest participation makes the environment more valuable for honest participants). These are the conditions under which a multi-agent system converges on cooperation without central enforcement. Appendix J provides the formal treatment.

The protocol also decides that attested work maps to money through a deterministic router. This is a structural assumption, not a neutral recording of facts. The router embeds a minimal economic model: distribution is proportional to attested units, the waterfall applies in a fixed order, and parameters are immutable after creation. These are assumptions. They are the smallest possible set that still produces deterministic output. The alternative is to separate attestation from distribution entirely, letting the protocol record contributions and leaving applications to decide how those contributions translate to compensation. That separation preserves the purity of the record at the cost of reintroducing the gap between work and payment where intermediaries live. The music industry demonstrates this outcome: songwriting credits have survived for decades as a reliable attestation layer, while millions in royalties sit uncollected because the distribution layer between the record and the money is too complex, too opaque, and too full of intermediaries to function reliably. The router is a deliberate choice to accept a small set of visible, immutable structural assumptions in exchange for money actually reaching the people who did the work. That exchange delivers its full value only when revenue enters voluntarily. A project that collects revenue and never deposits it circumvents the router entirely. The protocol paid for the router with a loss of record-layer purity. It collects on that investment only when projects deposit. This is a real gap. Without the router, the protocol has two problems: capture and distribution. With it, distribution is solved and capture is transparent. A project with attested contributions and an empty router is a publicly readable fact. The router does not solve capture. It makes capture the only remaining problem and makes the failure to capture visible to everyone.

3. Protocol Invariants

The following five properties are non-negotiable. They cannot be configured away by any application built on Tunnels. They are the protocol's identity (what it refuses to do) and each

traces directly to the protocol's purpose of reducing friction between value creation and value capture.

Invariant 1: Identity is non-transferable. You cannot buy, sell, or transfer your contribution history.

Protects against capital capture. The moment contribution units become transferable, capital enters. Someone with money buys units from someone who needs liquidity. The person with capital accumulates weight without contributing. The person who did the work loses their future claim. The system devolves into another financial market. Non-transferability is philosophically equivalent to the non-transferability of a medical license: the credential has value precisely because it cannot be purchased. If someone wants transferable units, other protocols exist. Tunnels' identity is defined by this refusal. A consequence: there is no passive ownership in this protocol. Economic weight accrues only through attested, verified work. A participant who stops contributing stops earning. No allocation, no title, and no prior agreement can substitute for the attestation record.

Invariant 2: Attestations are append-only and immutable after finalization. History cannot be rewritten.

Protects against revisionism and power plays. In traditional systems, compensation agreements can be renegotiated retroactively, equity can be clawed back, and the record of who did what is controlled by whoever holds institutional power. Append-only attestation removes this leverage. Once work is finalized, it exists permanently. No party can alter the historical record.

Invariant 3: Distribution is deterministic. Given the same inputs, the same outputs result. No discretion, no override.

Protects against negotiation bias. If any party can override the distribution function, the system reintroduces the exact intermediary step it was designed to remove. Determinism means the rules are the rules, applied uniformly regardless of who the contributor is.

Invariant 4: Parameters are transparent at creation. Every contributor can see the terms before participating.

Protects against information asymmetry. In current systems, employees often lack visibility into how compensation is structured relative to revenue, how equity dilution works, or what their peers earn. Tunnels requires that all distribution parameters are visible and immutable from the moment a project is created. Contributors join with full information or they do not join.

Invariant 5: Challenge is always possible within the verification window. No attestation can bypass challenge.

Protects against fraud. The verification window is the protocol's minimum safeguard. If an attestation is inaccurate, inflated, or fabricated, any participant with standing can challenge it before finalization. The window duration is an application-layer parameter with a protocol-enforced minimum of 24 hours. The existence of the window is a protocol guarantee. It is the only bound the protocol enforces, and it exists because without it the protocol's integrity collapses.

The protocol enforces a 24-hour minimum verification window. This is the minimum time required for the challenge mechanism to function across all timezones and operational contexts. Projects may set any verification window at or above 24 hours. The protocol establishes the floor. Applications and project owners determine the appropriate window above that floor based on their operational needs.

4. The Five Primitives

4.1 Identity

An identity is a persistent, non-transferable record that accumulates contribution history across the ecosystem.

```
Identity = (
     public_key,             // the identity's public key (address derived from
this)
    type,                // Human | Agent
     parent,                 // if Agent, linked Human identity (revenue routes
here)
    profile_hash,        // hash of off-chain identity profile
    created_at           // timestamp of creation
)
```

4.1.1 Key Architecture

An identity is controlled by a single cryptographic key pair: one private key and one public key, following the same model as Bitcoin and other peer-to-peer cryptographic systems.

The private key is the identity's root of ownership. It signs every transaction the identity performs: attesting contributions, claiming revenue, creating projects, filing challenges, and updating the profile. The private key is held solely by the contributor and never shared with any application or stored on any server. It is the contributor's sole proof of ownership: the equivalent of a deed, not a badge. Loss of the private key means permanent loss of the identity.

Key rotation was considered and rejected. Rotation solves one problem, preserving on-chain reputation history after key compromise, but creates a worse one: it enables clean transfer of identity ownership, which breaks non-transferability. Without rotation, identity sale requires sharing the private key, leaving both parties with access and no clean handoff. With rotation, the

buyer rotates to their own key and the seller is locked out. The sale is final. The identity graph becomes a marketplace.

The security concern that motivates rotation, key compromise, is an application-layer problem with an application-layer solution. A compromised key cannot steal funds because the application controls payouts through its own authentication, not the protocol key. A compromised key cannot destroy future earning potential because the contributor creates a new identity and the application reconnects it to their account. The only loss is on-chain reputation history, which matters but does not justify undermining non-transferability to preserve it. An application can link multiple on-chain identities to a single user account, aggregating earnings and history across identities without the protocol needing to know they belong to the same person. The protocol sees keys. The application sees people. Key management lives at that boundary.

The public key is derived mathematically from the private key and recorded on-chain when the identity is created. It is used by the protocol and all participants to verify that a transaction was signed by the identity's owner. The address, a hash of the public key, is the identity's on-chain identifier. Applications store the address to associate off-chain data (profiles, deliverables, evidence) with the on-chain identity. The address is portable: the same address identifies the same identity across every application built on the protocol.

The contributor signs every transaction in their own browser or device through a client-side SDK that applications embed in their frontends. The SDK generates the key pair, stores the private key locally, and signs transactions when the user initiates an action. From the user's perspective, clicking "Submit work" or "Claim revenue" is the only action required. The SDK handles the cryptographic signing invisibly. The application's server never participates in signing contributor transactions. It receives only the public address and pre-signed transactions.

If an application is compromised, the attacker gains access to contributor addresses and profile data. They cannot sign any transaction as any contributor because the private keys were never on the server. No fraudulent attestations can be submitted. No identity records can be corrupted. No revenue can be claimed. The damage from a server breach is limited to the exposure of public information that was already visible on-chain. This is the same security model as Bitcoin: a compromised exchange that never held your private key cannot spend your coins.

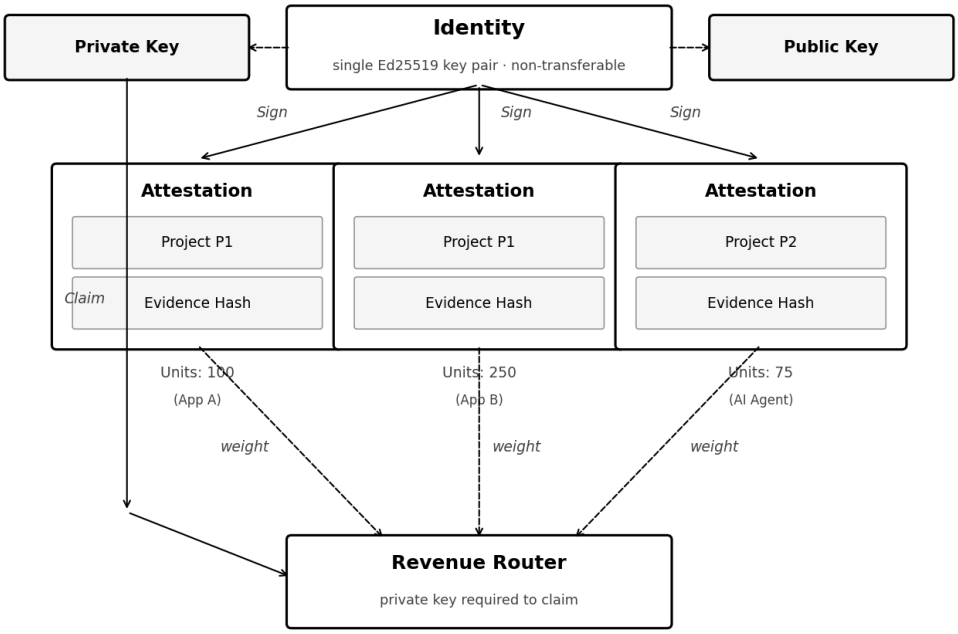

Figure 1. Key architecture. The private key signs all transactions. The public key verifies signatures on-chain. Applications store only the address.

4.1.2 Identity Types

Identities are either Human or Agent. Agent identities are separate identities with their own key pair, linked to a parent Human identity through the create_agent transaction. The link is recorded on-chain. Revenue earned by an Agent's contributions routes to the parent Human's claimable balance automatically at the protocol level. The parent bears accountability for the Agent's attestations and can deactivate the Agent at any time. A deactivated Agent's contribution history persists on-chain; only the revenue link is severed. Projects decide how to treat Agent-contributed work.

The profile_hash field stores a hash of an off-chain identity profile (name, visual mark, description). The protocol stores the hash for cross-platform legibility: every application resolves the same hash, so an identity is recognizable across the ecosystem.

Identity persists across all projects. An identity's full attestation history is available as a queryable, tamper-proof record. The protocol stores this history. It does not interpret it.

4.2 Project

A project is the protocol-level container against which contributions are recorded. It binds attestations to a distribution configuration. A project may operate indefinitely with zero revenue deposits: contributions accumulate, attestations finalize, and the identity graph grows regardless

of whether money ever enters. If and when revenue is deposited, the distribution configuration determines how it splits. The protocol's primary function is recording contributions. Revenue distribution is a consequence of contribution, not the purpose of it.

```
Project = (
    project_id,          // unique identifier
    creator,             // identity that registered the project
    predecessor,         // optional: linked source project
    delta,               // optional: predecessor allocation rate
    rho,                 // reserve rate
    phi,                 // fee rate
    phi_recipient,       // identity receiving fees
    gamma,               // genesis allocation rate
    gamma_recipient,     // identity or project receiving genesis allocation
    verification_window,  // duration of verification period
    created_at           // timestamp of creation
)
```

All parameters are set at creation and are immutable for the lifetime of the project. The creator identity is recorded but confers no special protocol-level privileges after creation. The protocol has no concept of project ownership or administration. Whether the creator has ongoing authority, whether governance is shared, and how project-level decisions are made are outside the protocol's scope. The protocol records who created the project for the same reason it records who signed an attestation: accountability and provenance. Whether a project accepts attestations from any identity or restricts participation to approved identities is an application-layer decision. The protocol provides the same verification mechanics regardless: bonds and challenge windows apply to every attestation whether the project is open or closed.

The optional predecessor and delta fields enable provenance linking. A project may declare that it derives from a prior project and allocate a percentage of its revenue to that source. When revenue enters the derivative project's router, the delta allocation is sent to the predecessor project's router first, where it distributes to the predecessor's contributors by their original attestation weight. The protocol does not know why two projects are linked. It does not distinguish between a sequel, a derivative work, a remix, a project succession, or an AI-trained output. It stores a pointer and a rate. The decision to declare a predecessor, and what percentage to allocate, is made by the project creator at creation. The protocol routes the value. This is among the protocol's simplest mechanics and potentially its most consequential: a single pointer and rate create a deterministic compensation path from derivative works back to their sources, with implications for ethical AI compensation, open source sustainability, and research lineage explored in Appendix C.

A project's parameters are publicly visible from the moment of creation. Any identity considering contributing to a project can inspect the reserve rate, fee structure, genesis allocation, predecessor link, and verification window before attesting any work. This is Invariant 4 in action: full information, no hidden terms.

4.3 Attestation

An attestation is the protocol's atomic unit. It is a cryptographic record that a contribution occurred, backed by an economic commitment.

```
Attestation = (
    contributor,      // identity that did the work and signed the transaction
    project,          // which project
    units,            // contribution weight (positive number)
    evidence_hash,    // hash of proof artifact
    timestamp,        // when created
    finalized_at,     // when finalized (null until state = Finalized)
state,             // Pending | Challenged | Finalized | Rejected
bond_poster,      // identity that posted the bond (may be contributor or third
party)
bond_amount              //  locked  economic  commitment,  minimum  is  project
attestation_bond
)
```

The contributor field identifies who did the work and who signed the transaction. They are always the same identity. The protocol does not support delegation: no identity can sign an attestation on behalf of another identity. You attest your own work with your own key. This is a deliberate constraint. If a manager, peer, or automated system could attest work on your behalf, a compromised application holding a delegated key could permanently corrupt your identity record. By requiring the contributor's own signature, the protocol makes unauthorized attestation cryptographically impossible rather than merely economically discouraged.

The protocol is agnostic about what the attestation represents. It does not know or care whether the units reflect hours worked, deliverables completed, or any other measure. It does not evaluate whether the evidence hash points to a git commit, a design file, a signed contract, or meeting notes. It records that a contribution of a given weight was attested with a reference to proof at a specific time.

The finalized_at field records when an attestation completed its lifecycle: either by surviving the verification window unchallenged or by being resolved in favor of the contributor after challenge. This gives the protocol and every application reading from it a complete timeline: when the work was claimed, and when it was confirmed.

The bond_poster field records which identity locked the economic commitment backing this attestation. This may be the contributor themselves or a third party such as an application acting on the contributor's behalf. The bond_amount is the value locked, which must be at least the project's minimum attestation_bond. On finalization, the full bond is returned to the poster. On challenge rejection, the full bond is forfeited to the challenger. The bond mechanism and its role in Sybil resistance are described in Section 6.

Attestations are append-only. Once created, they cannot be modified. They advance through a state machine: Pending to Finalized (if unchallenged through the verification window), Pending

to Challenged (if challenged), and from Challenged to either Finalized or Rejected through the adjudication mechanism. The default is finalization. Inaction cannot prevent a legitimate contribution from being recorded.

The contributor creates their own attestation by signing the transaction with their private key. The protocol requires that every attestation is signed by the contributor's registered identity key. Applications may structure the workflow around attestations (defining deliverables, collecting evidence, computing unit weights), but the cryptographic act of attestation is always performed by the contributor themselves.

4.4 Attestation Verification

The protocol constrains its verification mechanism to a single binary question: is this attestation finalized or rejected? There is no partial credit, no quality score, no weighted devaluation.

This is a deliberate design constraint grounded in the Gibbard-Satterthwaite impossibility theorem, proven independently by Gibbard (1973) and Satterthwaite (1975). Any decision mechanism with more than two possible outcomes is either dictatorial or manipulable. Rating systems, peer review scores, weighted assessments, star ratings: all fall within this impossibility. The protocol stays inside the mathematical boundary where strategy-proof mechanisms exist. This constraint has consequences beyond verification: it is what makes the protocol's entire incentive structure strategy-proof, and it is the foundation on which the cooperative equilibrium described in Section 4.4.2 and formalized in Appendix J depends.

The question the protocol asks is "did this work happen?" not "was this work good enough?" Quality assessment is an application-layer concern. The protocol detects fraud, not mediocrity.

Every attestation passes through a time-windowed verification period before finalization. The protocol defines a state machine with four states:

```
Pending → Challenged → Finalized | Rejected
```

Pending: The attestation has been created and is within its verification window. It is visible to all participants but does not yet affect distribution weight.

Challenged: A challenge has been filed. The attestation is frozen. It cannot finalize while in this state. The challenger must provide a counter-evidence hash explaining the basis for the challenge.

Finalized: The verification window has closed without challenge, or a challenge has been adjudicated in favor of the attestation. The attestation is now immutable and its weight counts toward distribution.

Rejected: A challenge has been adjudicated against the attestation, or the original contributor failed to respond within the adjudication window. The attestation is invalidated and its weight never enters distribution.

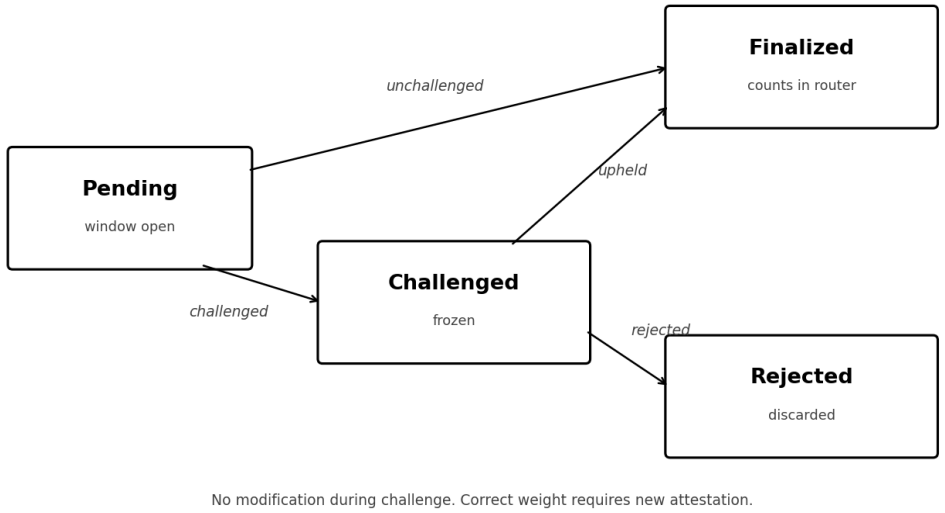

Figure 2. Attestation state machine. No modification during challenge. Correct weight requires a new attestation.

A challenged attestation can only be finalized as originally written or rejected entirely. There is no mechanism to modify an attestation's weight during challenge. If a challenge reveals that the correct weight should be different from what was attested, the original attestation is rejected and a new attestation with the corrected weight must be submitted, entering its own verification window. The protocol does not edit records. It accepts or rejects them.

The default adjudication mechanism is timeout-based: if a challenged attestation receives no counter-response from the original contributor within an adjudication window, the challenge succeeds automatically and the attestation is rejected.

Applications may override the default adjudication with their own: majority vote of unit holders, designated arbiters, multi-signature panels, AI-assisted evidence review, or any other system. The protocol requires only that the application's mechanism ultimately calls one of two protocol functions (finalize or reject) to advance the state machine. The existence of the verification window, the freezing of challenged attestations, and the state transitions are protocol-level guarantees. How adjudication works is up to the application. The verification window duration is an application-layer parameter with one protocol-enforced minimum: no window shorter than 24 hours.

4.4.1 Bonds

The protocol uses two distinct bond mechanisms. Attestation bonds are protocol-level and mandatory: every attestation requires a bond locked at submission, described fully in Section 6. The bond's minimum amount is set by the project's attestation_bond parameter. On finalization, the full bond is returned to the poster. On challenge rejection, the full bond is forfeited to the challenger. Attestation bonds serve as Sybil resistance by imposing economic risk on attestations, and the forfeiture incentivizes challenges by rewarding participants who detect fraud.

Challenge bonds are application-layer and optional. Whether a challenger must also post a bond is the application's choice. A three-person collective where members know each other does not need financial deterrence against frivolous challenges. An open marketplace with thousands of anonymous contributors may need strong anti-griefing protection. The protocol provides lock-and-release mechanics to support challenge bonds when applications choose to require them.

When an application requires a challenge bond, the protocol locks it for the duration of the challenge and releases it automatically based on the outcome. If the attestation is finalized (challenge fails), the challenge bond is forfeited to the attestor. If the attestation is rejected (challenge succeeds), the challenge bond is returned and the attestation bond is awarded to the challenger. The protocol handles locking and release. The application decides whether to require challenge bonds and at what amount.

The protocol enforces that challenge_bond cannot exceed attestation_bond. If the cost to challenge is higher than the cost to attest, the challenge mechanism is suppressed. A project owner could set challenge bonds prohibitively high, making attestations effectively unchallengeable. The symmetric cap prevents this. The cost to dispute a claim can never exceed the cost to make the claim. Projects may set challenge_bond to any value from zero up to and including attestation_bond, but not above. This follows the established principle from dispute resolution mechanism design: the party whose behavior is being checked must not control the cost of checking it.

4.4.2 The Identity Graph as Verification

The verification window is the protocol's minimum safeguard. The identity graph is its primary defense. The endogenous verification described in Section 2 has a concrete mechanism.

A single attestation from a new identity carries no accumulated trust. An attestation from an identity with finalized contributions across twelve independent projects, co-attested by dozens of independent identities, carries a fundamentally different signal. The protocol does not judge

attestations. It makes the identity graph legible so that applications, AI systems, and human participants can read the accumulated signal and make their own trust decisions.

The cost structure is asymmetric. Each new attestation in a consistent history retroactively strengthens every previous attestation in that history. An identity's second attestation, from a different project with different co-contributors, does not merely add to the first. It cross-validates it. The third validates both. The cross-referential value of the graph compounds superlinearly with each new independent data point, while the cost of faking any single attestation remains constant. At sufficient network density, the cost of fabricating a consistent cross-project history exceeds any plausible return from fraud.

This is not an assertion. It is a consequence of known results in game theory. The folk theorem [7] proves that cooperation is the rational strategy in repeated games when participants value future payoffs sufficiently. Identity persistence ensures they do: because the identity graph accumulates value indefinitely, the effective discount factor approaches 1, which is the condition under which the folk theorem holds. The honest contribution equilibrium, where no participant can improve their expected outcome by deviating from honest attestation while others attest honestly, is a Nash equilibrium in this system. The binary constraint from the Gibbard-Satterthwaite result [5][6] ensures the feedback signal cannot be manipulated. The public graph ensures deviations are observable. The penalty for fraud grows with accumulated history while the reward for fraud remains bounded by a single project. These are the formal conditions for a stable cooperative equilibrium, and the protocol satisfies all of them by construction.

None of this requires knowing who anyone is. The identity graph operates at the key level, not the person level. A key's accumulated history (every project, every finalized contribution, every co-attestor) is readable without knowing the person behind it. You can see that key 0x7f3a has fifty attestations across twelve projects without knowing it belongs to anyone in particular. The trust signal is the pattern of the key's history. Privacy and legibility are not in tension. They are the same design.

Existing unit holders are the protocol's natural auditors. Any fraudulent attestation dilutes their share of future revenue. They have direct economic incentive to challenge. This self-policing dynamic scales with project value. Even without active policing, the identity graph creates a passive verification layer: patterns of fraud are visible to anyone reading the graph, and an identity's accumulated history is the most expensive thing in the system to fake. The most significant threat the identity graph does not solve on its own is Sybil attacks: a single human creating multiple identities to fabricate cross-referential reputation. This is the protocol's most serious threat. The protocol addresses it through mandatory attestation bonds described in Section 6, which impose economic risk on every attestation. Bonds are forfeited when

attestations are successfully challenged, making detectable fraud economically costly. The identity graph provides additional natural friction: each identity must accumulate history over time, and self-dealing clusters become forensically visible as the graph grows. The public, auditable nature of the chain means suspicious attestation patterns can be identified and challenged by any participant.

4.5 Deterministic Distribution

The revenue router accepts incoming revenue and distributes it according to a deterministic function based on attested contribution weight. The same inputs always produce the same outputs. No human discretion can override the distribution.

```
Revenue R
    |
    +-- Predecessor = R x delta      (provenance allocation, if linked)
    +-- Reserve     = R' x rho       (operational buffer)
    +-- Fee         = R'' x phi      (optional external fee)
    +-- Genesis     = R''' x gamma   (founder allocation)
    +-- Labor Pool  = remainder      (pro-rata by units)
```

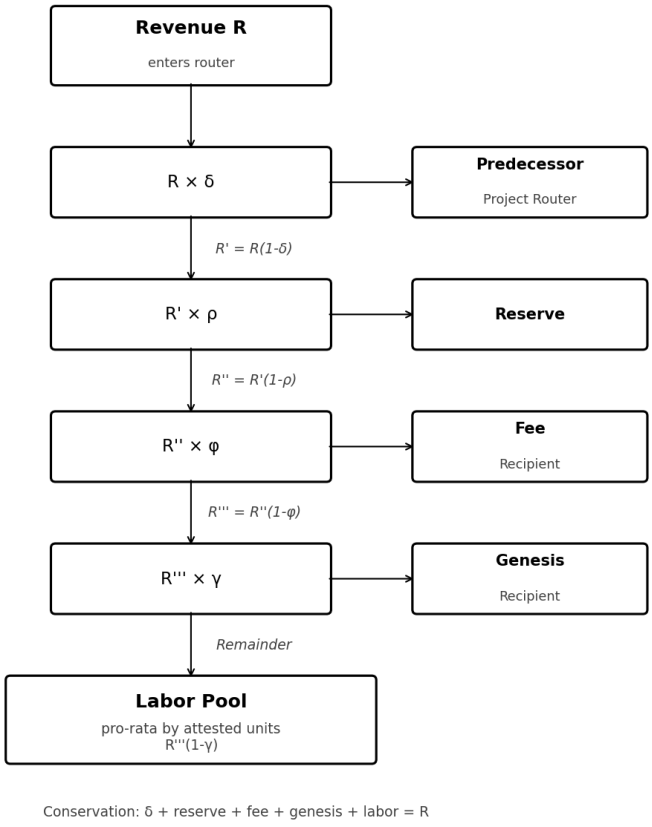

Figure 3. Revenue distribution waterfall. Each slice is computed on the remainder after prior slices. Conservation: all slices sum to R.

The router is the protocol's deterministic distribution mechanism. When a deposit transaction is submitted, the router records the amount, applies the waterfall, and updates each contributor's claimable balance. The deposit is a cryptographically signed commitment made by a known identity: the amount, the project, the depositor, and the timestamp are immutable and publicly auditable. The protocol guarantees that given the same deposits, the same distribution always results. It does not custody the underlying value. Settlement (the actual movement of dollars, cryptocurrency, or any other form of payment from the project to the contributor) occurs at the application layer. What the protocol provides is enforceable accounting: every deposit, every distribution, and every claim is recorded against identities whose full history is visible in the graph. A project that records deposits without settling them builds a visible pattern of non-payment. Contributors stop working for projects that do not pay. The identity graph makes non-settlement expensive for the same reason it makes fraudulent attestation expensive: the graph is public, the history is permanent, and reputation compounds. The protocol is agnostic to what form the underlying value takes. Revenue that enters the router distributes deterministically. Revenue that never enters the router is outside the protocol's scope. How much of a project's total revenue is routed through Tunnels is left to the application. A project may route all of its revenue, a percentage, or only its profit margin. The protocol guarantees the integrity of whatever enters. It cannot compel entry. Critically, deposits are not required for a project to function. A project may operate indefinitely with zero deposits. Contributions accumulate, attestations finalize, and the identity graph grows regardless of whether money ever enters. The router is dormant infrastructure, ready to distribute the moment value arrives, irrelevant until it does.

The parameters delta, rho, phi, and gamma are set at project creation and are immutable for the lifetime of the project. If no predecessor is declared, delta is zero and the predecessor slice is skipped.

Distribution uses reward-index accounting with O(1) computational cost regardless of the number of contributors. Whether a project has 10 contributors or 10,000, the cost of processing a deposit is constant.

4.6 Infrastructure Requirements

The protocol requires sovereign control over its invariants. Its immutability guarantees, zero-cost participation, and independence from third-party governance are not features that can be negotiated with an underlying platform. If a contributor must acquire a token issued by another system before they can attest work, the protocol has reintroduced the intermediary friction it was designed to remove. If an external governance mechanism can alter the protocol's state transition rules, the invariants described in this paper are not invariants. The protocol's cost to the contributor must be zero, and the protocol must control the infrastructure that enforces this.

How these requirements are satisfied (the consensus mechanism, node operator economics, and the case for or against deploying on existing infrastructure) is an implementation decision addressed in the accompanying technical specification. Spam prevention is achieved through per-transaction proof of work whose difficulty self-adjusts based on network transaction volume, requiring no governance or external calibration. This paper defines what the infrastructure must provide. It does not prescribe how to build it.

5. Incentive Structure

For the protocol to be stable, honest behavior must be the rational strategy for all participants.

Contributors cannot claim unearned units because every attestation must be signed by the contributor's own private key, requires an evidence hash, and must survive the verification window. A fraudulent attestation requires either the contributor themselves lying about their work, or an attacker who has stolen the contributor's private key. The first is detectable through the challenge mechanism: it is an economic attack on every existing unit holder in the project, diluting their share. Existing contributors have direct financial incentive to monitor and challenge. The second is the same threat model as stolen Bitcoin keys, not an application-layer vulnerability but a personal security failure.

The incentive to monitor exists. Whether contributors act on it is a different question. In practice, people do their work and move on. They do not audit a ledger for dilution. They do not calculate their share and notice when it drops. The theoretical self-policing dynamic is real but it depends on someone paying attention, and the protocol cannot assume anyone will. This is why mandatory attestation bonds described in Section 6 are the protocol's primary defense rather than contributor vigilance. Bonds impose economic risk on every attestation, and forfeiture when challenged successfully creates direct cost for fraud. Applications can make monitoring practical by surfacing anomalies, flagging suspicious patterns, and alerting contributors when something looks wrong. The protocol provides the data. Applications make it visible.

Founders receive a transparent genesis allocation that compensates early-stage risk without requiring ongoing extraction. The genesis recipient may be a single identity or a project address. When the recipient is a project, the genesis allocation flows into that project's router and distributes to its contributors by their attestation weights. This allows multiple founders to share the genesis allocation through a "founders project" where they attest their founding contributions among themselves. The same protocol mechanics (attestation, verification, waterfall) govern the internal split. Because the genesis parameter is immutable and visible at creation, it functions as a market signal: a high genesis allocation must attract contributors despite reduced labor pool share, while a low genesis allocation signals collaborative intent. The market decides what works.

These individual incentives are self-reinforcing, but they also compound at the network level. When every participant's rational behavior produces an increasingly dense and reliable identity graph, the system generates a collective asset that no individual created and no individual can replicate.

6. Sybil Resistance

If a single human can create multiple identities and attest contributions to themselves across those identities, they can build fraudulent reputation at the protocol level. The identity graph, which this paper identifies as the protocol's entire moat, is poisoned. Every claim made in the preceding section about self-policing incentives and every claim made in the following section about network effects depends on the graph being trustworthy. Sybil resistance is not one problem among many. It is the problem that, if unsolved, invalidates the system.

The protocol's design provides natural friction against casual Sybil attacks. Each identity must accumulate attestation history over time and survive verification windows, making rapid reputation fabrication expensive. The identity graph creates forensic surface area: patterns of self-dealing (identities that only attest to each other, clusters with no external connections) become detectable as the graph grows denser. The economic cost of maintaining multiple fake identities that appear legitimate scales with the monitoring incentive of the existing participants whose revenue share is being diluted. These frictions are real. They are not sufficient. A patient, well-resourced attacker willing to build fake reputation slowly will not be stopped by verification windows alone.

The conventional framing presents this as a trilemma. Identity creation must be free, because the protocol cannot erect an access barrier against the contributors it was designed to serve. Anonymity must be default, because the protocol's privacy model requires it. And Sybil resistance requires some cost to identity, because without cost there is no scarcity and without scarcity there is no trust. Every standard approach (proof of personhood, economic staking at the identity level, mandatory social vouching) resolves the trilemma by sacrificing one of the other two properties. The protocol refuses to make that sacrifice.

The trilemma dissolves when the cost is moved from identity to attestation. Creating an identity is free. It is a key pair with no history. A fake identity is indistinguishable from a real one and equally harmless: it cannot affect revenue distribution until it has finalized attestations. The only action that matters is attestation, because attestation is the only action that creates units, and units are the only thing that earns revenue. If attestation is where the economic consequence lives, attestation is where the economic cost belongs.

Every attestation requires a bond. When a contributor submits an attestation, a bond denominated in the project's revenue currency is locked by the protocol's rules. The bond is

posted by any identity: the contributor themselves, an application acting on their behalf, or a third-party bonding service. The protocol records the bond poster's identity and the locked amount. If the attestation finalizes unchallenged, the full bond is returned to the poster. If the attestation is challenged and rejected, the full bond is forfeited to the challenger. The bond poster bears the economic risk of the attestation's legitimacy.

The challenge mechanism is central to Sybil defense. Without active challengers, an attacker can create a fake project, cross-attest between fake identities, and recover the full bond every time because no challenger exists. The protocol's defense is that all attestations are public and auditable. Suspicious patterns (many attestations from the same creator with no external contributors, clusters of identities that only attest to each other) are visible to anyone reading the chain. Applications and participants can identify these patterns and file challenges. The cost of a successful challenge attack is bond forfeiture, which the attacker loses entirely.

The contributor pays nothing. The application that facilitates the attestation posts the bond on the contributor's behalf. If the attestation finalizes, the bond is returned in full. The contributor's experience is unchanged: submit work, the SDK signs, the attestation enters the verification window. The bond mechanics are invisible. Zero-cost participation means the contributor pays nothing. The protocol has an economic cost of operation (opportunity cost of locked bonds during verification windows). That cost is borne by the entities that facilitate attestations, not by the contributors who produce them. A contributor with capital can also self-bond, preserving the sovereign path. The protocol does not require an application to be involved. It requires a bond.

This resolves the fragmentation problem that would otherwise be fatal to a unified identity graph. Without bonds, Sybil defense quality depends on each application's implementation, and the weakest application poisons the graph for everyone. With bonds, an application with weak contributor vetting loses money on every fraudulent attestation it sponsors when those attestations are challenged. The protocol does not need every application to implement good Sybil resistance. It needs every application to bear the economic cost when they fail at it. The incentive is not to be vigilant. The incentive is to not go broke. Applications that vet poorly lose bond capital when challenged. Applications that vet well keep it. The protocol punishes bad Sybil defense economically rather than hoping every participant gets it right voluntarily.

The remaining attack is self-funding: an attacker runs a minimal node, creates a fake project, deposits real revenue, self-bonds, and self-attests. No challengers exist because no legitimate contributors are present. The attestations finalize. The bonds return in full. The revenue distributes back to the attacker's own identities. The attacker has manufactured backed reputation at the cost of capital locked during verification windows plus whatever phi they paid through the application layer. The protocol's defense is that this activity is public and auditable. Any observer can identify suspicious patterns: projects with only self-attestations, clusters of

identities that attest only to each other, rapid creation of multiple projects with the same structure. These patterns can be flagged and challenged by any participant.

Bonds change the protocol's atomic unit. An attestation is no longer a pure cryptographic record. It is a record with an economic commitment attached. A contributor who cannot post a bond or find someone willing to post one on their behalf cannot attest. This is a real constraint that the protocol did not have before. But the protocol's purpose is not architectural minimalism. It is fairness: ensuring that people who do real work receive compensation proportional to their contribution. Without Sybil resistance, an attacker who manufactures fake reputation dilutes every legitimate contributor's share of the revenue pool. That is the least fair outcome the protocol can produce. The bond is the economic commitment that gives challenge incentives teeth. Bonds serve that purpose directly.

This paper claims in Section 4.6 that the protocol's cost to the contributor must be zero. That claim still holds. The bond is posted by the bond poster, which for most contributors is the application. On successful finalization, the bond is returned in full. The contributor submits work and gets paid. The economic risk of Sybil resistance (bond forfeiture on successful challenge) is real, but it is borne by the infrastructure layer, not the labor layer. The protocol is free to participate in.

Sybil resistance is not a problem that gets solved once. It is an adversarial dynamic that evolves for as long as the protocol exists. Mandatory bonds make Sybil attacks economically constrained at the protocol level rather than deferring the entire problem to applications. Combined with verification windows, challenge economics, and the forensic surface area of a public identity graph, it creates a layered defense where each layer handles a different attack profile: bonds impose risk on attestation, challenges impose cost on fraud through forfeiture, and graph analysis makes coordinated deception visible over time. No single layer is sufficient. Together they cover the attack surface. The protocol does not claim to make Sybil attacks impossible. It claims to make them detectable and economically costly when challenged. Whether that is sufficient depends on the density and vigilance of the ecosystem that forms around it. That is an empirical question. It is not answered here.

7. Network Effects

The protocol can be forked. The identity graph cannot. Any party can replicate Tunnels’ code, but the accumulated attestation history of every identity in the ecosystem exists only within the network and grows more valuable as the network grows. Each new identity that attests honestly adds signal. Each cross-project attestation strengthens the graph’s reliability as a trust layer. The loop is self-reinforcing: more participants produce a denser graph, a denser graph produces a stronger signal, and a stronger signal attracts more participants. This is the protocol’s moat,

and it is a social consensus moat, not a cryptographic one. A competing protocol could import Tunnels' attestation data, but it could not import the social consensus that those attestations are trustworthy. Trust is earned through the challenge mechanism over time. It cannot be copied. The full network effects analysis appears in Appendix H.

8. Privacy

The protocol records work, not people. An identity is a cryptographic key pair: one private key held by the owner, one public key recorded on-chain. The link between these keys and a human being exists only in the holder's mind. The protocol never asks who you are, never stores who you are, and provides no mechanism to compel disclosure. It sees keys and contributions. It does not see people.

All attestation metadata is on-chain and public because the challenge mechanism requires it. Work is transparent. Identity is anonymous by default. Every act of identification is voluntary and occurs at the application layer. The full privacy model, including key management, custody, new identity creation, and the treatment of abandoned revenue, appears in Appendix I.

9. Conclusion

Tunnels is a self-governing protocol with five primitives and five invariants. It records contributions, enforces verification windows, and distributes revenue deterministically. It does not decide what a contribution is, how it should be weighted, or how a project should be governed. Its constraints are not arbitrary: binary verification, a public identity graph, non-transferable history, and deterministic distribution are the formal conditions under which a multi-agent system converges on cooperation without central enforcement.

The protocol embeds a minimal economic model in exchange for money reaching the people who did the work. That is a conscious tradeoff. The alternative, a pure attestation layer with no distribution, preserves the record and loses the money. The router accepts a small set of visible, immutable structural assumptions so that contribution has economic consequences, not just historical ones. Where the router cannot reach, it makes the gap visible. Revenue that never enters is not governed by the protocol, but its absence is a publicly readable fact.

There is no passive ownership. Economic weight accrues only through attested, verified work. No title, no allocation, and no prior agreement substitutes for the record. Every parameter self-adjusts from on-chain data. No governance vote, no external oracle, and no committee is required to keep the protocol running. Governance is needed to change the protocol, not to operate it.

What gets built on Tunnels is not the protocol's concern. The protocol provides the record, the verification, and the distribution. Applications decide everything else. The protocol does its job and gets out of the way.

References

[1] Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System.

[2] Buterin, V. (2014). Ethereum: A Next-Generation Smart Contract Platform.

[3] Weyl, E.G., Ohlhaver, P., Buterin, V. (2022). Decentralized Society: Finding Web3's Soul.

[4] Coase, R.H. (1937). The Nature of the Firm. Economica.

[5] Gibbard, A. (1973). Manipulation of Voting Schemes: A General Result. Econometrica, 41(4), 587–601.

[6] Satterthwaite, M.A. (1975). Strategy-Proofness and Arrow's Conditions. Journal of Economic Theory, 10(2), 187–217.

[7] Friedman, J.W. (1971). A Non-cooperative Equilibrium for Supergames. Review of Economic Studies, 38(1), 1–12.

[8] Sutton, R.S., Barto, A.G. (2018). Reinforcement Learning: An Introduction. 2nd ed. MIT Press.

The following appendices address domains that each warrant dedicated formal treatment. What follows establishes the groundwork: security arguments, the protocol–application boundary, application examples, explicit scope limitations, provenance economics, prior art differentiation, network effects, privacy architecture, and the protocol's formal correspondence to multi-agent reinforcement learning environments.

Appendix A: Security Arguments

The following arguments establish the protocol's security properties informally. The conservation proof and O(1) equivalence are mathematically rigorous. The remaining arguments (non-extractability, fraud resistance, and self-policing) are economic reasoning supported by incentive analysis. Formal verification of these properties is identified as future work.

A.1 Conservation

Every unit of value deposited is accounted for across all distribution channels. When a predecessor link exists (δ > 0):

```
predecessor + reserve + fee + genesis + labor
= Dδ + D(1−δ)ρ + D(1−δ)(1−ρ)φ + D(1−δ)(1−ρ)(1−φ)γ + D(1−δ)(1−ρ)(1−φ)(1−γ)
= D[δ + (1−δ)ρ + (1−δ)(1−ρ)φ + (1−δ)(1−ρ)(1−φ)γ + (1−δ)(1−ρ)(1−φ)(1−γ)]
= D[δ + (1−δ)(ρ + (1−ρ)(φ + (1−φ)(γ + 1 − γ)))]
= D[δ + (1−δ)(1)]
= D
```

When no predecessor exists (δ = 0), the proof reduces to the four-channel case and the result is identical.

A.2 O(1) Distribution Equivalence

Using a reward-index accumulator, the claim for identity i after deposits d_1 through d_n equals the sum of individual pro-rata calculations:

```
claim(i) = units(i) × (R_n − R_entry(i))
where R_n = Σ(d_k / totalUnits_k) for k = 1..n
```

This is mathematically equivalent to iterating over each deposit and computing i's share, but executes in constant time regardless of the number of contributors or deposits.

A.3 Non-Extractability

A participant cannot extract more value than their attested contribution weight entitles them to. The deterministic distribution function and immutable parameters ensure that claim(i) / totalClaimed = units(i) / totalUnits for all i. No configuration of application-layer parameters can violate this property because the router is a protocol-level primitive.

A.4 Fraud Resistance

Unauthorized attestation is cryptographically impossible. Every attestation requires the contributor's private key signature. An attacker who compromises an application gains no signing capability over any contributor's identity. The remaining attack surface is a contributor submitting inflated or fabricated attestations under their own key. For such a fraudulent attestation to succeed, it must survive the verification window without challenge. The economic cost of unchallenged fraud to existing unit holders is proportional to (fraudulent_units / total_units) × future_revenue. As project value increases, the incentive to monitor increases proportionally. The expected cost of fraud (probability of detection × loss of all units + reputation damage) exceeds the expected benefit (marginal revenue share) for any project with non-trivial economic activity.

A.5 Self-Policing

For any unit holder with stake s in a project with revenue r, the expected loss from an unchallenged fraudulent attestation of weight f is: s/(s + totalUnits + f) × r − s/(s + totalUnits) × r. This loss is always negative, which proves the economic incentive to challenge exists for every unit holder on every project with non-zero revenue. The incentive scales with stake and with project value. Whether any individual contributor acts on this incentive is a behavioral question, not a mathematical one. The mandatory bond mechanism described in Section 6 provides the baseline defense: fraudulent attestations that are challenged result in full bond forfeiture to the challenger. The self-policing incentive is real and provable. It is complementary to bond forfeiture, not a substitute for it.

Appendix B: Protocol vs. Application Layer

The following table clarifies the boundary between what the protocol provides and what applications built on Tunnels must decide for themselves.

Protocol Layer (Tunnels)	Application Layer (Projects)
Identity creation and persistence	User onboarding and verification
Single-key architecture	Key custody and storage method
Non-transferability enforcement	Contributor recruitment and matching
Attestation data structure	What constitutes a valid contribution
Append-only immutability	How many units a contribution is worth
Verification window mechanism	Verification window duration (above minimum)
Verification window existence guarantee	Who has standing to challenge
Bond mechanics (lock, return, forfeit)	Bond amounts above protocol minimum and whether challenge bonds are required
Deterministic revenue router	Router parameters (ρ, φ, γ, δ)
Predecessor revenue routing	Whether to declare a predecessor and what delta to set
O(1) distribution accounting	Governance and decision-making
Minimum verification window enforcement	Category definitions
Identity history as queryable data	How to interpret or weight that history
Human/Agent identity distinction	Privacy and visibility settings
Profile hash storage (cross-platform legibility)	Profile content, rendering, and visual design

Evidence hash storage	What evidence means or requires

Appendix C: Provenance Chains and Derivative Works

The predecessor and delta fields on the Project struct are simple: a pointer and a rate. The implications are not.

Consider an animated film produced as a Tunnels project by 100 contributors (animators, voice actors, musicians, writers) each with attested, finalized contributions. A second team creates a sequel. They declare the original film as their predecessor and set δ = 10%. Every dollar of revenue the sequel earns sends ten cents to the original project's router, where it distributes to the 100 original contributors by their attestation weight. The original contributors are compensated for the value their work continues to generate, not by legal enforcement, but by protocol mechanics.

The chain can extend further. A third project derives from the sequel, declaring it as predecessor. Revenue flows backward through two links: the third project pays the second, the second pays the first. Each link's delta is set independently by the project creator. The original contributors earn from every generation of derivative work built on their foundation.

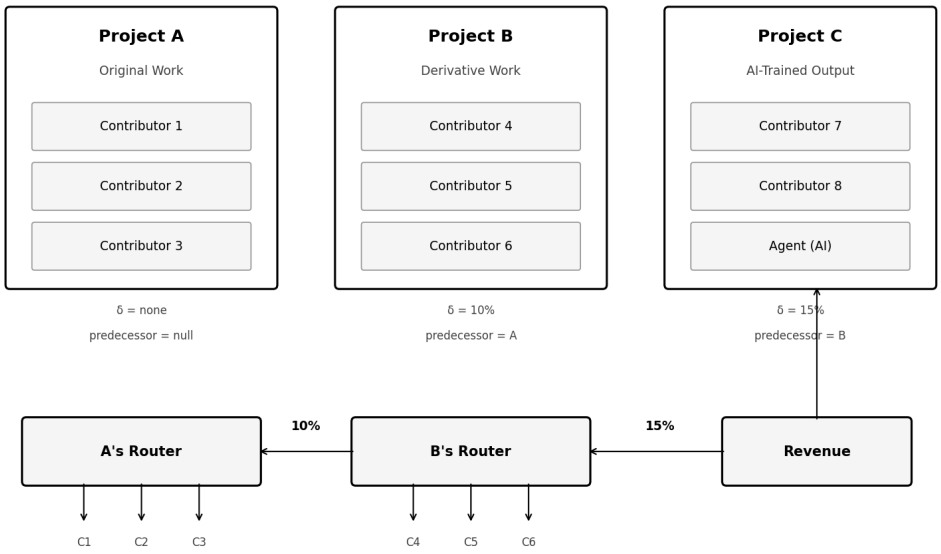

Figure 4. Predecessor chain. Revenue enters Project C, flows backward through declared links. Each project's contributors are paid by their own router.

The applications extend beyond creative works:

Ethical AI compensation. An AI model trained on creative works produced within the Tunnels ecosystem generates derivative output. If that output becomes a project on the protocol, the predecessor link creates a deterministic compensation path from AI-generated revenue back to the human creators of the training data. An application built for this purpose could require predecessor declarations as a condition of participation, enforcing ethical provenance at the application layer while the protocol handles the economics.

Open source sustainability. A commercial product built on an open source library declares the library as predecessor. Revenue flows back to the library's contributors automatically. The library maintainers who built the foundation are compensated by every commercial success built on top of it.

Research lineage. A startup spun out of academic research declares the original research project as predecessor. The researchers whose work made the company possible receive ongoing compensation proportional to their contribution to the foundational work.

Project succession. A long-running project that needs to update its parameters creates a new project with a predecessor link to itself. Contributors to the original project continue to earn from the successor through the delta allocation. The predecessor link replaces the need for parameter mutability. However, successor projects do not inherit contributor weight. Attestation history on the original project does not confer standing on the new one. The delta is a compensation mechanism, not a continuity guarantee. Contributors must earn their position in the successor through new attested work. The tradeoff is real: immutability protects contributors from having terms changed beneath them, but it also means project evolution requires starting fresh rather than editing in place.

The protocol does not enforce predecessor declarations. A derivative project can choose not to link back to its source, the same way someone can copy a creative work without attribution under current systems. The protocol cannot solve the attribution problem through code alone. What it can do is provide the infrastructure so that when attribution is declared, compensation is automatic, deterministic, and permanent. Applications built on this infrastructure (ethical AI platforms, remix culture marketplaces, open source funding mechanisms) can enforce attribution norms at the application layer using the protocol's provenance primitive.

The full design of recursive provenance chains (including depth limits, cumulative allocation caps, and the interaction between predecessor revenue and the conservation proof at arbitrary chain lengths) is identified as future work requiring dedicated formal analysis.

Appendix D: Agent Identity Model

The protocol distinguishes between Human and Agent identity types at the primitive level (Section 4.1.2). This appendix explains the design problem that distinction solves and the tradeoffs it introduces.

Without a separate agent identity type, machine-produced work is attributed to the operator's human identity. The identity graph cannot distinguish human output from machine output. This creates three technical problems. The operator's attestation history conflates work they performed with work their agents performed, degrading the graph's signal quality. Projects cannot evaluate the track record of a specific agent before accepting its contributions. And the challenge mechanism cannot function correctly if challengers cannot determine whether they are evaluating human work or machine output, because the evaluation criteria may differ.

The agent identity model solves this by giving each agent its own key pair, its own attestation history, and its own on-chain record. The parent link to the operator is visible and immutable. Revenue routes to the parent automatically at the protocol level. An operator with twelve agents has twelve separate track records. A project evaluating an agent sees that agent's history, not the operator's. A challenger reviewing an agent's attestation knows it is machine output and can evaluate accordingly.

The protocol does not need to understand what an agent is or how it operates. It records that an identity of type Agent attested to work, that the attestation survived or did not survive the verification window, and that revenue distributed accordingly. The type distinction is metadata on the identity, not a different set of rules. Agents and humans follow the same attestation mechanics, the same verification windows, the same bond requirements. The distinction exists for graph legibility, not for differential treatment.

The verification window applies to agent attestations identically. An agent's work does not finalize until the window closes. Any participant with standing can inspect the evidence and challenge it. The window is a protocol-enforced pause between an agent completing work and that work having economic consequences. This is the same mechanism applied to human attestations. It does not treat agent work as less trustworthy. It treats all work the same way.

The tradeoff is that agent identity depends on honest declaration. The protocol cannot verify that an identity declared as Human is actually human, or that an identity declared as Agent is actually a machine. The type field is self-reported. An operator who registers an agent as Human to avoid the parent revenue link can do so. The protocol's defense is the same as its defense against all dishonest attestation: the identity graph makes patterns visible over time, and the mandatory bond mechanism makes sustained deception economically risky through

challenge-based forfeiture. Applications can add verification layers (proof of personhood for humans, deployment attestation for agents) at the application level.

The agent identity type adds one field to the identity struct and one revenue routing rule to the protocol. The cost is minimal. The alternative, treating all identities as identical regardless of whether they represent humans or machines, preserves simplicity at the cost of graph legibility. As the ratio of machine-produced work to human-produced work increases, that cost compounds. The protocol chose legibility.

Appendix E: Application Examples

The following examples demonstrate how different applications can use the same protocol primitives to build very different systems. Each example shows how the application answers the questions the protocol leaves open. Several examples reference tenure: this is not a protocol primitive. An identity's full attestation history (timestamps, projects, finalized contributions) is on-chain and queryable. Applications can read this history and derive tenure, reputation scores, experience weighting, or any other signal from it. The protocol stores the data. The application decides what it means.

E.1 Freelance Marketplace

A platform where contributors find projects, contribute work, and earn revenue share based on attested deliverables.

How units are assigned: Project stewards define deliverables with pre-set unit values. A landing page redesign might be worth 100 units. A backend API integration might be worth 250 units. Values are published before work begins. Contributors accept or decline.

Who attests: The contributor submits an attestation upon delivery. A designated project steward reviews the evidence (code review, design review) and does not challenge if satisfactory. Finalization happens automatically after the verification window.

Challenge standing: Any existing unit holder in the project may challenge. The steward has explicit challenge authority.

Router configuration: ρ = 5% (minimal reserve), φ = 3% (platform fee), γ = 15% (founder allocation). All visible at project creation. The platform fee compensates the application operator for building and maintaining the marketplace. The infrastructure connects contributors to projects. The fee is visible to all contributors before they participate, and is the explicit cost of the convenience the platform provides. Contributors who find projects independently can use a zero-fee application or interact with the protocol directly.

Tenure usage: The platform reads ecosystem-wide tenure from the protocol's identity data and uses it as a trust signal when matching contributors to projects. It does not affect unit weighting.

E.2 Collective Ownership (e.g., Artist-Owned Record Label)

A music collective where artists, producers, engineers, and managers share revenue from catalog sales based on their contributions to each release.

How units are assigned: The collective establishes a contribution framework: songwriting = 40 units, production = 30 units, engineering/mixing = 20 units, artwork/visual = 10 units per release. These are defaults that the collective votes on and can vary by release.

Who attests: Each contributor self-attests with evidence (session files, stems, writing credits). The collective reviews during the verification window.

Challenge standing: All collective members have standing. Challenges are resolved by majority vote of existing unit holders.

Router configuration: ρ = 10% (studio costs, marketing), φ = 0% (no external fee), γ = 0% (no founder allocation, collective). This is a 0% genesis project: purely labor-pool distributed.

Tenure usage: Long-standing collective members receive priority on release scheduling. Tenure does not affect revenue split.

E.3 Transparent Corporation (e.g., Aerospace Startup)

A venture-backed company that uses Tunnels for transparent, contribution-based compensation alongside traditional salary.

How units are assigned: The company defines a role-weighted system: engineers earn 1 unit per story point delivered, designers earn 1 unit per approved design deliverable, sales earns 1 unit per $10K in closed revenue. Weights are set by leadership and published in the project terms.

Who attests: Contributors attest their own work. Engineers submit attestations upon completing story points, referencing CI/CD artifacts as evidence. Designers attest deliverables with approved design files. Sales attests closed deals. Each contributor signs their own attestation in the browser. Automated systems may operate as Agent identities to record system-level events, but they do not attest on behalf of human contributors.

Challenge standing: Any employee with units may challenge. A designated review board has expedited challenge authority.

Router configuration: ρ = 20% (operational costs, salaries are paid separately), φ = 0%, γ = 30% (founder allocation reflecting early risk and capital investment). Revenue entering the router is profit-share, not total revenue.

Tenure usage: The company applies a tenure multiplier at the application layer: employees with 3+ years receive 1.2× weighting on their units. This is the company's choice, not a protocol requirement.

E.4 Mentorship Network

A platform where experienced professionals mentor emerging talent and earn contribution credit for teaching, guidance, and knowledge transfer.

How units are assigned: Mentorship sessions are worth units based on duration and verified completion. A mentee's attestation of the session (counter-signed) generates units for the mentor. If the mentee subsequently contributes to a project, the mentor may receive a fractional attestation reflecting their indirect contribution.

Who attests: Dual attestation: both mentor and mentee attest to the session. Disagreement triggers the challenge mechanism.

Challenge standing: Both parties and a platform-designated mediator.

Router configuration: $\rho = 5\%$, $\varphi = 5\%$ (platform operation), $\gamma = 0\%$. Revenue comes from mentee subscriptions or project-level allocations for mentorship.

Tenure usage: Ecosystem-wide tenure is the primary signal for mentor matching. A mentor with five years of verified contributions across the network is visibly more experienced than one with six months. This is read directly from the protocol's identity data.

The full design of recursive provenance chains (including depth limits, cumulative allocation caps, and the interaction between predecessor revenue and the conservation proof at arbitrary chain lengths) is identified as future work requiring dedicated formal analysis.

E.5 Film Production and AI Derivative Chain

The deepest application of predecessor chains emerges when human creative work and machine capability interleave across multiple generations of derivative projects. Consider a concrete example that traces the full chain.

A filmmaker produces an independent film as a Tunnels project. Thirty contributors: the director, cinematographer, actors, editors, sound designers, composers. Each person's work is attested over the course of production. The film is released. Revenue enters the router and distributes to the people who made it. This is the base case. The protocol is just a payment system at this layer.

A machine learning team wants to train an AI model on the film's visual style. The lighting, the framing, the color grading, the way the cinematographer composed shots. They create a new project for the AI model and declare the original film as predecessor with a delta that reflects the training data's contribution to the model's capability. The team's own work (data preparation, model architecture, training, evaluation) is attested on their project. The AI model is now a derivative of the film. If the model generates revenue through licensing or API access, delta

flows back to the film's router, and from there to the thirty people who made the film. The cinematographer whose visual style the model learned from gets paid when someone uses the model. They don't need to know the model exists. The predecessor chain knows.

A production studio uses that AI model to generate visual assets for a new film. The AI is deployed as an agent identity, producing shots in the style it learned from the original film. The new film is its own project. It declares the AI model as predecessor. Human contributors to the new film (the director, the actors, the editors who work with the AI-generated footage) attest their work alongside the agent's contributions. Revenue from the new film enters its router. The waterfall runs: delta flows to the AI model's project, and from there delta flows again to the original film's project. The original cinematographer is now three layers removed from the revenue event and still getting paid.

This is four projects deep. The original film, the AI model trained on it, the new film that used the model, and potentially a soundtrack or marketing campaign derived from the new film that adds another layer. At each layer the predecessor link is a single field on the project struct. The protocol runs the same waterfall at every level. No contract renegotiation at each layer. No licensing department tracking derivatives. No legal dispute about whether the AI-generated work owes anything to the training data. The delta was set when the predecessor was declared. The protocol enforces it on every deposit, at every depth, permanently.

The filmmaker who shot the original film in a warehouse with a small crew has created something that compounds. Not because they negotiated a good deal or hired the right lawyer. Because the work they did became part of a creative lineage that the protocol tracks automatically. Every derivative, every generation, every new project that builds on what they made sends revenue back through the chain to the people who started it. That is the model the creative industries were supposed to build. Predecessor chains are the mechanism that makes it real.

These five examples use identical protocol primitives with different configurations. The protocol does not prefer any model over another.

Appendix F: What the Protocol Does Not Do

The following are explicitly outside the protocol's scope. These are not omissions. They are design decisions that keep the protocol focused by leaving application-level concerns to applications.

The protocol does not assess contribution quality. It records that an attestation was made with an evidence hash. Whether the evidence represents excellent work, adequate work, or fraudulent work is determined by the challenge process, which is triggered by participants with economic standing, not by the protocol itself.

The protocol does not compare units across categories. If a project defines both "design" and "engineering" contribution categories, the relative value of a design unit versus an engineering unit is an application decision. The protocol distributes based on aggregate unit weight regardless of category.

The protocol does not govern projects. How decisions are made within a project (who leads, who votes, how strategy is set) is entirely outside the protocol. Tunnels is compatible with flat collectives, traditional hierarchies, and everything in between.

The protocol does not compel revenue entry. This is the most significant boundary on what the protocol can deliver, and it deserves more than a passing acknowledgment. The deterministic distribution guarantee, the protocol's headline property, governs money that enters the router. It does not govern money that exists outside it. A project can collect revenue through side channels and never deposit it. The attestations are on-chain. The router is empty. The protocol works perfectly and the contributors got nothing. The distribution problem and the capture problem are different problems. The protocol solves distribution trustlessly. It does not solve capture.

The paper's reputational argument, that contributors will not work for projects that do not pay, describes an equilibrium that depends on a dense identity graph, liquid contributor mobility, and fast information flow about project behavior. These are properties of the mature network, not the early one. At launch, the graph is sparse, contributor alternatives are limited, and information about non-payment travels slowly. Power asymmetries persist in thin markets. People take bad deals when they need work. The reputational enforcement mechanism is real, but it is the end state, not the starting condition. The protocol has a bootstrap vulnerability on capture that is distinct from its bootstrap vulnerability on network effects, and the paper should not let the two blur together.

What the protocol does provide is transparency. A project with five hundred finalized attestations and zero deposits is a publicly readable fact on an immutable ledger. This is qualitatively different from the transparency that exists in current systems, where information about

underpayment lives in private conversations, unverified reviews, and individual grievances that never aggregate. In Tunnels, the capture gap is legible. It can be read programmatically. It can be aggregated across projects. It creates a signal that contributors, applications, and third-party services can act on collectively in ways that do not currently exist. Transparency with a permanent public record is stronger than the transparency available today.

But transparency is not enforcement. The protocol cannot force a project to deposit revenue any more than it can force a project to exist. The router's accounting is immutable and publicly auditable for everything that enters it. What stays outside is visible as an absence, not governed as a violation. The protocol makes the distribution side trustless and the capture side transparent. These are two different things and this paper is not conflating them.

The protocol does not handle project lifecycle. What happens when a project ends, pivots, or fails is an application concern. Attestation history persists at the identity level regardless of project status. A contributor's work on a failed project remains part of their verifiable history.

The protocol's Sybil resistance mechanism (mandatory attestation bonds with challenge-based forfeiture) is described in Section 6. That section acknowledges the residual limitations: a well-funded attacker can still attempt to build reputation through self-attestation in the absence of challengers. The mechanism makes Sybil attacks detectable and economically risky when challenged. It does not make them impossible.

The protocol does not upgrade itself. As specified in this paper, the protocol requires no ongoing governance to maintain its current parameters. Spam prevention difficulty self-adjusts based on on-chain data. Project parameters are immutable at creation. No committee, vote, or external decision is needed for the protocol to continue operating as designed. Meta-governance (how the protocol evolves, who proposes changes, how consensus is reached on protocol upgrades) is a challenge that exists above the protocol layer and applies only to future evolution, not to maintenance of the current specification. The ecosystem may develop governance mechanisms for protocol evolution, potentially using Tunnels' own primitives: identities with sufficient ecosystem-wide attestation history could participate in governance weighted by their verified contribution to the network. The distinction is important: governance is needed to change the protocol, not to run it.

The protocol distinguishes between structural permanence and parametric permanence. Structural permanence is the protocol's identity: append-only attestations, non-transferable history, binary verification, deterministic distribution, immutable project parameters, and irrecoverable keys. These are permanent because the alternatives introduce manipulation surfaces the protocol cannot close. They are permanent by design and they do not create governance obligations. Parametric permanence is different. A fixed spam prevention difficulty is permanent not because the alternative is dangerous but because the protocol lacked a

mechanism to adjust it. Every fixed parameter is a governance debt: correct at launch, wrong eventually, requiring human intervention to recalibrate. This protocol eliminates parametric permanence by making its operational parameters self-adjusting. Spam prevention difficulty tracks transaction volume and uses on-chain data that the protocol already collects. It maintains its intended function across changing conditions without external input. The result is a protocol with no governance dependencies in its current specification. Structural permanence protects the system. Parametric permanence was a liability. This protocol keeps the first and eliminates the second.

Appendix G: Prior Art and Differentiation

Tunnels is a contribution attestation protocol. It uses distributed ledger technology and cryptographic verification as underlying infrastructure.

Prior systems built on distributed ledger technology (cryptocurrencies, tokens, decentralized finance protocols) created transferable digital assets. A token, regardless of what it represents at creation, becomes a financial instrument the moment it can be traded. Its price detaches from the underlying activity. Speculation dominates. The people doing the work become secondary to the people trading the asset. This is not a failure of implementation. It is an inevitable consequence of transferability. Any transferable unit of value will be financialized, because financialization is what markets do with transferable value.

Tunnels does not create a transferable asset. An attestation is a record, not an instrument. It cannot be sold any more than a surgeon can sell the record of a surgery they performed. Non-transferability is what separates Tunnels from every token-based system. There is only a growing, immutable record of who did what. The regulatory implication follows from the structural properties. Units cannot be bought, sold, or traded. There is no secondary market. Revenue accrues only through the contributor's own attested work, not through the efforts of others. The protocol makes no legal determination about the classification of its units, but its design ensures they function as records of work performed, not as investment instruments.

Existing smart contract frameworks are neutral infrastructure. They execute code as written without regard for outcome. A smart contract will enforce a 99% extraction fee or a 0% extraction fee with equal indifference. Tunnels is also neutral with respect to configuration: any parameter setting is valid. The difference is transparency and immutability. Every parameter is visible at project creation and cannot be changed afterward. A project with a 95% founder allocation is not prohibited. It is visible. Contributors see the terms before they participate and choose accordingly. The market, not the protocol, decides what configurations attract talent.

Attempts to build decentralized currencies through consortium governance reveal a structural contradiction: decentralized systems require decisions, decisions require decision-makers, and decision-makers become centralized authorities. The table changes; the power dynamic does not. Tunnels does not decentralize authority. It eliminates the need for authority over the functions the protocol governs. The router executes math, not judgment. The attestation ledger is append-only, not edited. Where human judgment is required, the protocol delegates explicitly to the application layer, where centralized authority is expected, visible, and bounded.

Efforts to let individuals tokenize their productive value through personal and social tokens pursued a sound intuition: if every person is a micro-monopoly, every person should have their own unit of account. But transferable personal tokens immediately became speculative

instruments, their price detaching from the work they were meant to represent. And a personal token still requires a common medium of exchange to transfer value, making it not a currency but a redundant accounting layer. Tunnels preserves the insight while discarding the failed mechanism. An identity's attestation history is the personal token: unique, non-purchasable, accumulated only through verified work. But it is not tradeable. Compensation flows in a common medium, distributed by attestation weight. The identity is the signal. The revenue is the reward. Separating them is what makes it work.

Finally, in token-based systems, the token is the valuable asset. In Tunnels, the identity graph is the valuable asset: the accumulated, verified, portable history of contributions across every project in the ecosystem. This history cannot be bought, cannot be forked, and grows more meaningful as the network grows. Network effects in token systems are driven by liquidity and speculation. Network effects in Tunnels are driven by the density and reliability of the identity graph, the collective truth of who has done what, verified by challenge over time.

Appendix H: Network Effects and the Identity Graph

The protocol can be forked. The identity graph cannot.

Any party can replicate Tunnels' code. What they cannot replicate is the accumulated attestation history of every identity in the ecosystem: the verified record of who did what, when, where, and how reliably. This history exists only within the network and grows more valuable as the network grows.

More identities create more attestation data. More attestation data makes each identity's history more meaningful as a signal. More meaningful signal attracts more participants. The loop is the protocol's moat, and it is a social consensus moat, not a cryptographic one. A competing protocol could theoretically import Tunnels attestation history, but it could not import the social consensus that those attestations are trustworthy. Trust is earned through the challenge mechanism over time. It cannot be copied.

Identity persistence is a protocol-level primitive rather than an application feature. If identity existed only within individual applications, each application would have its own isolated reputation silo. By making identity protocol-level, every application built on Tunnels contributes to and benefits from the same identity graph.

Appendix I: Privacy Model

The protocol records work, not people.

An identity is a cryptographic key pair: one private key and one corresponding public key. The link between this key pair and a human being exists only in the holder's mind. The protocol never asks who you are, never stores who you are, and provides no mechanism to compel disclosure. This is not a privacy feature added to a transparent system. It is a foundational architectural decision: the protocol sees keys and contributions. It does not see people.

All attestation metadata (contributor key, project, units, state, timestamps) is on-chain and public. The work is transparent because the challenge mechanism requires it. If challengers cannot see what they are challenging, challenges are theater and the self-policing incentive collapses. Revenue distribution math is auditable because determinism is only a guarantee if it can be verified. Work artifacts, customer data, and detailed evidence remain off-chain, referenced by hash. Who can access the evidence behind a hash is an application-layer access control decision.

Every act of identification is voluntary and occurs at the application layer. A contributor may link their key to a real-world name because they want collaborators to find them. They may link multiple keys together to present a unified portfolio. They may verify their key against a government identity for projects in regulated industries. Or they may do none of these things and remain a key with a history, known only by what they have done, never by who they are.

Every current system works the opposite way. Today, identity is the default (your name on the resume, your face on the profile, your credentials on the application) and you fight for privacy. Tunnels defaults to anonymous. Visibility is something you opt into, on your terms, at the level you choose. When the work is the only signal, the circumstances of the person become irrelevant.

Applications built on Tunnels may construct identity layers, verification services, portfolio aggregators, and compliance frameworks on top of the protocol's data, all without the protocol ever knowing or requiring who is behind the key. A professional identity layer could function as a credential system backed by verified attestations rather than self-reported claims. A verification layer could confirm that a key belongs to a real human without revealing which human. A compliance layer could require identity disclosure as a condition of project participation for regulated industries. All of these are built on top of the protocol, not inside it. The protocol provides the truth about the work. The person decides how much truth about themselves to attach to it. This extends to Sybil resistance. An application that integrates a proof-of-personhood protocol (biometric verification, social graph analysis, or any system that confirms one human per key without revealing which human) could reduce bond requirements

for verified identities because the Sybil risk is lower when unique personhood is established. The protocol does not require this. It provides mandatory bonds and challenge-based forfeiture as the universal baseline. Applications that want stronger guarantees or lower costs can add identity verification on top without the protocol depending on any external system.

I.1 Key Management

The single-key architecture described in Section 4.1.1 directly serves the protocol's privacy and security model. The private key is the identity's root of ownership. It should be stored with the highest security the contributor is willing to maintain: a hardware device, an encrypted seed phrase in physical storage, or any method that keeps it offline and under sole control when not in use. Applications never receive the private key. They interact with the contributor through the client-side SDK, which signs transactions in the browser and sends only signed transactions and the public address to the server.

Loss of the private key means permanent loss of the identity. There is no protocol-level recovery mechanism. Introducing one would require either the protocol to know who holds the key (violating anonymous-by-default) or a key rotation mechanism that could be exploited to transfer identity ownership (violating non-transferability). This is the honest cost of true ownership: no one can take it from you, and no one can recover it for you. These are the same property. Every recovery door is also a seizure door.

The application layer provides the practical solution to key compromise without requiring protocol-level key rotation. A compromised key cannot steal funds because the application controls payouts through its own authentication, separate from the protocol key. When a key is compromised or lost, the contributor creates a new on-chain identity and the application reconnects it to their existing account. The application can link multiple on-chain identities to a single user, aggregating earnings and contribution history across identities. The protocol remains unaware that these identities belong to the same person, preserving the privacy model. The cost of key loss is the on-chain reputation history attached to the old identity, not the contributor's earnings, future earning potential, or application-level account.

The protocol cannot distinguish between a transaction signed by a contributor on their own device and a transaction signed by an application on the contributor's behalf. It verifies signatures, not custody methods. An application can generate a private key, store it on its own server, and sign transactions for the user. The protocol will process these transactions. It has no mechanism to detect or prevent this, and building one would require the protocol to know where a key is stored, which would violate the privacy model.

This creates a predictable failure mode. If the easiest way to create an identity requires trusting an application with the private key, most contributors will do so without understanding what they

have given up. The application becomes a custodian: it holds the contributor's root of ownership, can claim their revenue, attest under their identity, and act as them entirely. The protocol's security model depends on the private key staying with the contributor. If the private key lives on an application's server, the contributor has no security at all.

The protocol addresses this by shipping three tools that make sovereign key management the path of least resistance.

The first is a standalone identity generator: a single downloadable file that runs in any browser with no server and no internet connection. It generates a key pair, encrypts the private key with a password the contributor chooses, and exports a backup file. The contributor's identity exists before any application is involved. The file can be verified against a published cryptographic hash and distributed through any channel. It requires nothing from anyone.

The second is a client-side SDK that applications embed in their frontends. When a contributor creates an account through an application, the SDK generates the key pair in the contributor's browser. The private key is not transmitted to the application's server. The SDK signs all contributor transactions in the browser and sends only the signed transactions and the public address to the application. The application receives the minimum information it needs to identify the contributor on-chain and relay signed transactions to the node. The private key remains on the contributor's device. The SDK enforces this by design: it does not provide a method to export the private key across a network boundary.

The third is a command-line wallet for developers and node operators who work directly with the protocol. It generates keys, submits transactions, and manages identities through the node's API. It runs on the contributor's machine and requires no external service.

All three tools produce the same key format, the same encrypted backup format, and the same identity structure. A key generated in the identity generator works in the SDK and the command-line wallet. A contributor can create their identity offline, import it into an application through the SDK, and manage it from the command line interchangeably.

An application can still choose to ignore the SDK, generate private keys on its server, and manage them in its own database. The protocol cannot prevent this. But the protocol's reference tooling makes the non-custodial path the default integration. An application using the SDK correctly cannot become a custodian of private keys because the SDK does not give it access to them. Contributors who use applications built on the SDK hold their own keys whether they know it or not.

The gap between these tools and real-world behavior is worth stating plainly. Key loss is permanent by design. That design choice is correct: every recovery door is also a seizure door, and the protocol chose irreversibility because the alternative is worse. But the consequence is

predictable. Some users will seek custodial safety nets. Some applications will provide them. When they do, the security property that a compromised server cannot sign contributor transactions no longer holds for those users. The protocol did not fail in this scenario. It made a tradeoff, and this is the cost side of that tradeoff. The SDK minimizes this cost by making sovereign key management invisible to the user during normal operation: the key stays in the browser, signing happens automatically, and the contributor's experience is indistinguishable from a custodial application. The backup download is where the gap remains. A contributor who loses their backup and has no other copy of their key loses their identity permanently. The protocol guarantees that the sovereign path is always available and always functional. It cannot guarantee that every user will stay on it. The remaining distance between the protocol's security model and how humans actually behave is real, and this paper is not pretending otherwise.

I.2 New Identities

Anyone can generate a new key pair at any time, creating a new identity. A new identity has zero history: no attestations, no tenure, no reputation. This is the natural cost of starting fresh and the natural solution to the right to be forgotten. You cannot delete attestations from an old identity. That would break immutability. But you can walk away from an identity entirely. If no one knows you held the key, it becomes an abandoned identity with an orphaned history. The cost of starting over is real: you lose everything you built. The protocol does not judge which you choose. New identities hold no privilege. They are the least powerful position in the ecosystem, which prevents identity creation from being a gaming vector.

Revenue allocated to an abandoned identity remains in the router indefinitely. The protocol provides no mechanism to redistribute unclaimed value. Any such mechanism would require distinguishing an abandoned identity from one whose owner simply claims infrequently, and a dormancy timeout that redistributes unclaimed revenue is an expiration date on earned compensation. The protocol will not take back what it owes you. Over time, some portion of routed value will be locked. That is the economic cost of the same property that protects every active contributor. If no one can seize your earnings, then no one can seize your earnings, even when you are gone.

Appendix J: Formal Incentive Analysis Through Multi-Agent Reinforcement Learning

The protocol's five primitives and five invariants were designed with the formal properties of multi-agent reinforcement learning environments in mind. This appendix uses RL formalism to analyze why those structural choices produce a stable cooperative equilibrium. The protocol does not perform reinforcement learning. It is a set of rules with incentive structures. The RL framework is an analytical lens, not a description of what the protocol does.

Under this framing, each identity maps to an agent. Attestations are actions. Revenue share is the reward signal. The identity graph is the shared state space. Binary verification (finalized or rejected) is the feedback mechanism: finalization is positive reward, rejection is negative reward. The superlinear cost of fraud described in Section 4.4.2 is the penalty function. The environment is non-stationary, because every agent's actions change the shared state for every other agent.

In reinforcement learning terms, a standard environment is defined by a tuple (S, A, T, R, γ) where S is the state space, A is the action space, T is the transition function, R is the reward function, and γ is the discount factor. The Tunnels mapping is:

S (state space): The identity graph. The full set of identities, projects, attestations, and their states. Every agent observes the same public graph.

A (action space): Attest, challenge, claim, create project, or do nothing. All actions are discrete and observable.

T (transition function): Deterministic for unchallenged attestations (Pending → Finalized after the verification window). Stochastic for challenged attestations (outcome depends on adjudication mechanism, which varies by application).

R (reward function): Revenue share from the router, distributed deterministically based on finalized attestation weight. The reward is continuous (proportional to units), delayed (paid when revenue enters the router), and shared (the same deposit rewards all unit holders pro rata).

γ (discount factor): Implicit in identity persistence. Because identity accumulates history indefinitely, the effective discount factor approaches 1. Future attestations retroactively strengthen past attestations through cross-validation. Agents with long time horizons are rewarded by the graph's structure.

With the mapping established, the equilibrium properties follow directly from the protocol's constraints.

The honest contribution equilibrium described throughout this paper, where fraud costs more than it returns at sufficient network density, is a Nash equilibrium in this multi-agent system. No agent can improve their expected reward by deviating from honest attestation, given that other

agents are playing honestly and the graph is public. The folk theorem from repeated game theory provides the formal basis: cooperation is rational when the discount factor is sufficiently close to 1, which identity persistence guarantees.

The design produces four properties that distinguish this environment from standard multi-agent settings and that are direct consequences of the protocol's invariants.

Full observability. The state space is public. Every agent can observe every other agent's full action history. This eliminates the partial observability problem that makes most multi-agent environments intractable. In Tunnels, the information asymmetry runs in the defender's favor: honest agents see the entire graph, while a fraudulent agent must coordinate deception across a fully visible state space.

Endogenous reward shaping. The protocol does not set reward values externally. The reward function is shaped by the collective actions of all agents. When agents attest honestly, the graph becomes denser, signal becomes stronger, and the environment becomes more rewarding for honest participants. The reward signal improves as a function of participation. In most RL environments, the reward function is fixed by the environment designer. Tunnels deliberately departs from this: the protocol defines the reward mechanism, but its value is determined by participation quality.

Asymmetric penalty scaling. The penalty for defection (fraud) grows superlinearly with the agent's accumulated history, while the reward for defection remains bounded by the single project's revenue share. An agent with a long honest history faces an increasingly unfavorable risk-reward ratio for fraud. The penalty function is self-reinforcing: the more an agent has invested in honest participation, the more they have to lose.

Binary action feedback. Because the environment's feedback on actions is binary (finalized or rejected), the signal is unambiguous. There is no noisy gradient, no partial credit, no subjective score that agents can dispute or game. This is the Gibbard-Satterthwaite constraint from Section 4.4 expressed in RL terms: binary feedback is strategy-proof feedback.

The mapping is useful but not clean. The transition function is partially stochastic: adjudication outcomes depend on application-layer mechanisms that the protocol does not specify, making the environment non-stationary in ways that vary across deployments. The reward signal is heavily delayed: revenue enters the router on the depositor's schedule, not the contributor's, and the gap between attestation and payment can be arbitrarily long. The state space grows without bound as identities, projects, and attestations accumulate, which makes the environment increasingly expensive to observe fully even though it remains technically public. None of these are disqualifying for the RL framing, but they mean the protocol is a structured environment with useful formal properties, not a solved one. The equilibrium arguments hold

under the assumptions stated. Whether real-world deployments satisfy those assumptions is an empirical question the protocol cannot answer by construction.

The practical implication is not that the protocol is an RL system. It is that the protocol's data is structured for learning. Application-layer AI systems (fraud detection, trust scoring, contributor matching, project recommendation) can treat the identity graph as training data and the protocol's binary outcomes as ground truth labels. The public state space, the clean reward signal, and the self-reinforcing penalty function are properties that applications can build on. The protocol provides the environment. Intelligence is what gets built on top.

The constraints that make the protocol simple (non-transferable identity, append-only history, binary verification, deterministic distribution, and a public graph) are the same constraints that make it analyzable through RL formalism and useful as a foundation for learning systems. Whether any specific deployment is a well-posed learning environment depends on the application-layer choices that complete it. The protocol does not need to be intelligent. It needs to be legible.

Acknowledgments

Special thanks to Lazlo Hollyfeld and the Pacific Tech class of 1985 for demonstrating that deterministic systems, properly understood, favor those who do the work.