Deterministic Literary Publishing: A Multi-Layer Provenance Model for Verifiable Manuscripts

Kevan Burns

Independent Researcher · FTH Trading, Norcross, GA

kevan.burns@fthtrading.com

ORCID: 0009-0008-8425-939X

Version 1.1 — February 15, 2026 | GitHub Repository

DOI: 10.5281/zenodo.18646886

ABSTRACT

Traditional publishing relies on institutional trust to establish authorship, version integrity, and distribution provenance. This trust model is opaque, non-auditable, and structurally dependent on intermediaries whose incentives may diverge from those of creators. We present a multi-layer provenance architecture that replaces trust with cryptographic verification, enabling any party to independently confirm a manuscript's authenticity, authorship, version history, and compositional integrity without relying on a central authority. The system combines Merkle tree hashing across multiple content categories, content-addressed storage, deterministic build pipelines, and append-only on-chain records to produce a five-layer verification stack. We formalize this approach as the Literary Protocol Standard (LPS-1) and validate it through two complete deployments: a 75,000-word novel anchored via four Merkle trees (31 manuscript blocks, 5 embedded artifacts, 10 visual assets, 10 AI-generation prompts) and a 13-story short fiction collection anchored via two Merkle trees (16 manuscript files, 13 audio narrations) — demonstrating protocol generalization across literary formats, TTS engines, and tree topologies. Both works are anchored across five smart contracts on a production blockchain, with cross-chain timestamping on Bitcoin. All claims in this paper are independently verifiable against public on-chain state and content-addressed storage.

Keywords: digital provenance, deterministic publishing, Merkle trees, content integrity, reproducible pipelines, literary technology, on-chain anchoring

CONTENTS

1. Introduction

The relationship between a literary work and its provenance has historically been mediated by institutions: publishers attest to authorship, archives preserve canonical versions, and courts adjudicate disputes. This arrangement functions adequately when institutions are trusted, but it provides no mechanism for independent verification. A reader cannot confirm that a digital manuscript is identical to what the author submitted. A researcher cannot verify which edition predates another without consulting a catalogue maintained by the publisher. An author cannot prove, without institutional cooperation, that a specific text existed at a specific time.

These limitations are not merely theoretical. Disputed authorship, unauthorized modifications, and versioning ambiguities are well-documented problems in both traditional and digital publishing [1, 2]. The shift toward digital-first distribution has exacerbated these issues: digital files are trivially copied, modified, and redistributed without leaving auditable traces.

We propose an alternative model in which provenance is a property of the work itself — computed, anchored, and verifiable without intermediary cooperation. The model treats a manuscript as a structured data object whose integrity can be verified at arbitrary granularity (individual chapters, embedded documents, visual assets) through Merkle proofs, whose existence at a point in time is attested by immutable blockchain records, and whose build process is deterministic — identical inputs always produce identical outputs.

This paper makes three contributions:

1. **A formal specification** (Literary Protocol Standard v1) for anchoring literary works with cryptographic integrity guarantees across four content dimensions: text, artifacts, images, and AI-generation prompts.
2. **A five-layer verification architecture** that chains filesystem state, version control history, content hashing, content-addressed storage, and on-chain records into a unified provenance stack where any layer can be independently audited.
3. **A complete reference implementation** validated against two production literary works — a novel and a short story collection — demonstrating that the model is practical, gas-efficient, format-independent, and compatible with existing publishing workflows.

The system is designed to be protocol-level infrastructure — a substrate on which traditional publishing, self-publishing, licensed distribution, and institutional archiving can operate without modification to their existing processes.

2. Related Work

2.1 Content Integrity in Digital Publishing

The problem of digital content integrity has been addressed through several approaches. Digital Object Identifiers (DOIs) provide persistent identification but not content verification — a DOI resolves to a URL that may serve modified content [3]. The ISBN system identifies editions but provides no mechanism for verifying that a particular file corresponds to a registered edition. PDF digital signatures attest to a document's state at signing time but are bound to the signing certificate's trust chain, creating a dependency on certificate authorities.

2.2 Merkle Trees in Data Verification

Merkle trees [4] have been applied extensively in distributed systems for efficient data verification. Git uses Merkle DAGs for repository integrity [5]. Certificate Transparency logs use Merkle trees to provide auditable records of TLS certificate issuance [6]. IPFS uses Merkle DAGs for content-addressed storage [7]. Our application adapts Merkle trees specifically for structured literary works, where the tree topology reflects the compositional structure of the manuscript (text, embedded documents, visual assets, generative prompts) rather than an arbitrary file hierarchy.

2.3 Blockchain-Based Provenance

Prior work on blockchain-based provenance has focused primarily on supply chain tracking [8], academic credential verification [9], and digital art authentication [10]. Applications to literary publishing have been limited and have typically emphasized tokenization (representing works as transferable digital assets) rather than verification infrastructure. Our approach differs fundamentally: the on-chain component stores only cryptographic commitments (Merkle roots, content hashes) — not the content itself — and serves as a timestamped, immutable anchor rather than a transactional medium.

2.4 AI Provenance and Disclosure

The increasing use of generative AI in creative workflows raises provenance questions that traditional models cannot address: which components were AI-generated, what prompts were used, and whether the disclosed generation parameters match the actual ones [11]. Our model includes a dedicated Merkle tree for prompt hashing (`promptRoot`), enabling verifiable disclosure of AI involvement at the protocol level rather than as a voluntary annotation.

3. System Architecture

3.1 Design Principles

The architecture is governed by four principles:

Verification over trust. Every claim about a work's identity, integrity, authorship, or history must be independently verifiable against publicly auditable state. No claim requires trusting the author, publisher, or any intermediary.

Granular integrity. Verification must be possible at arbitrary granularity — from the entire work down to a single chapter, embedded document, or image — without requiring access to the complete manuscript.

Deterministic reproducibility. The pipeline from source files to published output must be deterministic: identical inputs in identical order must produce byte-identical output. Any deviation indicates tampering or corruption.

Append-only history. Version history is strictly append-only. Published editions cannot be modified, only superseded or retracted. Retractions are themselves permanent records.

3.2 The Five-Layer Provenance Stack

Provenance is established through five independent, cross-verifiable layers:

```
Layer 5: On-Chain Anchor     Immutable blockchain record of content commitm
Layer 4: Content-Addressed    IPFS storage where CID = f(content)
         Storage
Layer 3: Cryptographic Hash   SHA-256 of compiled manuscript
Layer 2: Version Control      Git commit history with timestamps
Layer 1: Filesystem           Source files on author's machine
```

Table 1. Provenance layers with verification methods.

LAYER	ARTIFACT	VERIFIABLE BY	FAILURE MODE
5	Blockchain transaction	Any node / block explorer	Network unavailable (readable from any archive)
4	IPFS CID	Any IPFS gateway	Content unpinned (re-pinnable from local copy)
3	SHA-256 digest	Any SHA-256 implementation	Hash collision (computationally infeasible)
2	Git commit	Repository host / local clone	History rewrite (detectable; on-chain timestamps independent)
1	Local files	Author's machine	Data loss (recovered from IPFS pin + git clone)

Each layer serves as a check on the others. A modified file (Layer 1) produces a different hash (Layer 3), which mismatches the on-chain record (Layer 5). A corrupted IPFS pin (Layer 4) fails hash verification (Layer 3). A forged git history (Layer 2) is contradicted by the on-chain timestamp (Layer 5). No single point of compromise can produce a consistent forgery across all five layers.

3.3 Four-Tree Merkle Architecture

The manuscript is decomposed into four content categories, each forming an independent Merkle tree:

```
                 ┌────────────────────────┐
                 │      Edition Root      │
                 │  H(mR ‖ aR ‖ iR ‖ pR)  │
                 └───────────┬────────────┘
                             │
         ┌──────────┬────────┴─────┬──────────┐
         ▼          ▼              ▼          ▼
  ┌────────────┐ ┌──────────┐ ┌──────────┐ ┌──────────┐
  │ Manuscript │ │ Artifact │ │  Image   │ │  Prompt  │
  │    Root    │ │   Root   │ │   Root   │ │   Root   │
  │    (mR)    │ │   (aR)   │ │   (iR)   │ │   (pR)   │
  └─────┬──────┘ └────┬─────┘ └────┬─────┘ └────┬─────┘
        │             │            │            │
    31 blocks    5 artifacts   10 images    10 prompts
```

Figure 1. Four-tree Merkle architecture with unified edition root.

Manuscript Root (mR): SHA-256 Merkle root of 31 text blocks in canonical order defined by a build manifest (`order.json`). Each leaf is the SHA-256 hash of the raw Markdown file content.

Artifact Root (aR): SHA-256 Merkle root of 5 embedded documents (contracts, reports, communications) sorted alphabetically by filename.

Image Root (iR): SHA-256 Merkle root of 10 visual assets (cover art and chapter illustrations) sorted with cover first, then chapters alphabetically.

Prompt Root (pR): SHA-256 Merkle root of 10 AI-generation prompt records, each serialized as JSON and hashed individually, ordered by document position.

The **Edition Root** is the SHA-256 hash of the concatenation of the four tree roots:

$$\text{editionRoot} = \text{SHA-256}(\text{mR} | \text{aR} | \text{iR} | \text{pR})$$

This single 256-bit value is the commitment anchored on-chain. From it, any individual component can be verified via a Merkle inclusion proof consisting of sibling hashes along the path from leaf to root.

3.4 Merkle Tree Construction

Trees are constructed using the following rules:

1. **Leaf computation:** $h_i = \text{SHA-256}(\text{content}_i)$ where content is raw bytes (UTF-8 for text, binary for images).
2. **Internal nodes:** $h_{\text{parent}} = \text{SHA-256}(h_{\text{left}} | h_{\text{right}})$ where $|$ is string concatenation of lowercase hex digests.
3. **Odd-leaf rule:** If a layer has an odd number of nodes, the last node is duplicated before pairing.
4. **Ordering:** Canonical position per category (build manifest for text, alphabetical for artifacts and images, document order for prompts).

3.5 State Machine

Each edition progresses through a strictly linear, irreversible state sequence:

$$\text{DRAFT} \rightarrow \text{COMPILED} \rightarrow \text{HASHED} \rightarrow \text{PINNED} \rightarrow \text{ANCHORED} \rightarrow \text{PUBLISHED}$$

Transition rules:

- **Forward-only.** No state can be reversed. A compiled manuscript cannot be "un-compiled."
- **Idempotent.** Executing a transition twice with identical inputs produces identical outputs.
- **Author-gated.** The transition from PINNED to ANCHORED requires a cryptographic signature from the author's private key.
- **Version increment.** Each complete traversal creates a new edition. Previous editions remain immutable.

4. Smart Contract Architecture

The on-chain component consists of five contracts deployed to Polygon (EVM-compatible, proof-of-stake consensus, ~2-second block time). The contracts serve complementary functions and reference each other through immutable address pointers.

4.1 Contract Taxonomy

Table 2. Deployed smart contracts with functions and verification links.

CONTRACT	PURPOSE	ADDRESS	BLOCK
LiteraryAnchor	Genesis proof-of-origin	`0x97f4...a90` ^a^	83,002,198
PublishingKernel v1	Edition management, licensing	`0x511c...3ae` ^b^	83,008,833
PublishingKernelV2	Hardened kernel with ECDSA, timelock, freeze	`0xca9F...037` ^c^	83,010,944
RoyaltyRouter	Programmable revenue distribution	`0x4416...461` ^d^	83,009,717
AuthorIdentity	On-chain identity declaration	`0xB9ff...170` ^e^	83,011,404

^a^ Polygonscan: 0x97f456300817eaE3B40E235857b856dfFE8bba90 ^b^ Polygonscan: 0x511c653fC0F450ba41C42A89A3125CcBf2eFE8ae ^c^ Polygonscan: 0xca9F6604A9b498DB31d113836E2957c0a9aAE037 ^d^ Polygonscan: 0x44169829489d70aaecbf845870652871C65fC461 ^e^ Polygonscan: 0xB9ffa688A8Bb332221030BbBE46bE5bF03323170

All five contracts are source-verified on Polygonscan — the deployed bytecode matches the published Solidity source. Any party can audit the contract logic without trusting the deployer's claims.

4.2 LiteraryAnchor (Genesis)

The genesis contract is intentionally minimal: it stores the work's title, IPFS CID, and SHA-256 hash as constructor arguments, making them part of the contract's immutable bytecode. It contains no administrative functions, no upgrade mechanisms, and no proxy patterns. Its simplicity is its security model — there is nothing to exploit because there is nothing to modify.

```
address public immutable author;    // Set once, stored in bytecode
string  public title;               // "The 2,500 Donkeys"
string  public ipfsCID;             // Content-addressed locator
string  public sha256Hash;          // Content fingerprint
```

The `author` address is declared `immutable` in Solidity, meaning it is embedded in the contract bytecode at deployment time and cannot be altered by any transaction, including one from the author. This provides a stronger guarantee than access control — it is a property of the deployed code itself.

4.3 PublishingKernel v1

The v1 kernel introduces structured edition management:

- **Edition anchoring** with Merkle roots (all five trees), IPFS CID, SHA-256, and descriptive metadata.
- **Edition lineage** through a `supersedesEdition` pointer that creates a linked list of versions.
- **Canonical designation** to mark which edition is currently authoritative.
- **On-chain licensing** with template IDs, territories, terms, and linked revenue routers.
- **AI provenance** recording the model identifier and prompt set hash for editions containing AI-generated content.

4.4 PublishingKernelV2

The v2 kernel hardens the protocol with four security mechanisms:

ECDSA signature enforcement. Every edition-anchoring transaction must include a valid ECDSA signature from the author over the edition's content hash. This provides cryptographic proof that the author endorsed the specific content, not merely that their wallet submitted the transaction (which could occur through a compromised key in a signing service).

Edition freezing. A frozen edition emits a `EditionFrozen` event and cannot be modified further — it is permanently sealed. This is distinct from canonicality (which can change) and stronger than immutability-by-convention.

48-hour timelock on destructive operations. Retractions and license revocations require a two-step process: proposal, then execution after a mandatory 48-hour waiting period. This prevents impulsive or coerced destructive actions and provides a window for the author to cancel.

O(1) canonical lookup. A cached `canonicalEditionId` replaces the O(n) scan required in v1, enabling constant-time access to the current authoritative edition regardless of how many editions exist.

4.5 RoyaltyRouter

The RoyaltyRouter implements programmable revenue distribution using a pull-pattern design:

- **Basis-point splits** (1 bp = 0.01%) that must sum to exactly 10,000 (100%).
- **Recoupment waterfall** for advance recovery: a configurable percentage of incoming funds is diverted to recoupment until a specified amount is reached, after which all funds flow to normal splits.
- **Pull-based withdrawals** to prevent reentrancy attacks — payees claim accumulated balances rather than receiving push payments.

4.6 AuthorIdentity

The identity contract establishes a verifiable, on-chain link between:

- A real-world identity (legal name)
- A pseudonym (pen name)
- A cryptographic wallet address
- A bibliography of published works (on-chain registers linking to external platforms)
- The other contracts in the protocol (cross-referencing by address and role)

The contract stores 12 registered works (spanning on-chain editions, IPFS-pinned content, and commercial platform listings) and 4 linked contracts, creating a unified identity graph that any verifier can query.

5. Deterministic Build Pipeline

5.1 Pipeline Architecture

The build pipeline transforms source files into a verifiable, anchored edition through four deterministic stages:

```
Source Files ──▸ compile.js ──▸ hash.js ──▸ merkle.js ──▸ manifest.js
                      │              │           │              │
                      ▼              ▼           ▼              ▼
                 Compiled       SHA-256     Merkle        Per-file
                 Manuscript     Digest      Roots         Integrity
                                                          Map
```

Figure 2. Deterministic build pipeline stages.

Stage 1 — Compilation. Reads `order.json` (a build manifest listing 31 blocks and 5 artifacts with insertion points) and concatenates them into a single output file. Block ordering is determined exclusively by the manifest, not by filesystem sort order, ensuring cross-platform determinism.

Stage 2 — Hashing. Computes the SHA-256 digest of the compiled output. This digest is the canonical fingerprint for the edition.

Stage 3 — Merkle tree construction. Builds four independent Merkle trees from source files and computes the composite edition root.

Stage 4 — Manifest generation. Produces a per-file integrity map with SHA-256 hashes, file sizes, and modification timestamps for every source file, enabling fine-grained tamper detection.

5.2 Determinism Guarantee

The pipeline is deterministic by construction:

- All file orderings are explicit (manifest-driven, alphabetical, or document-order).
- All hash computations use SHA-256 with defined encoding (UTF-8 for text, raw bytes for images).
- All concatenation boundaries are deterministic (no random separators, no timestamps in output).
- The odd-leaf Merkle rule (duplicate last) is specified, not implementation-dependent.

Invariant INV-B3 (No Manual Artifacts): Files in the build output directory must only be produced by the pipeline. Manual editing of derived files violates determinism and breaks the hash chain.

5.3 Verification Procedure

Any party can verify an edition's integrity using only public information:

1. Retrieve the source files from IPFS using the edition's CID.
2. Run the build pipeline (`compile.js` , `hash.js` , `merkle.js`) on the retrieved files.
3. Compare the computed edition root against the on-chain value returned by `getEditionRoots(editionId)` .
4. If the roots match, the content is authentic and unmodified.

For partial verification (e.g., a single chapter), the verifier computes the chapter's SHA-256, obtains its Merkle proof from the published proof file, and walks the proof up to the manuscript root, then verifies the manuscript root contributes to the on-chain edition root.

6. System Invariants

The protocol defines 14 invariants organized into four categories. Violation of any invariant indicates an inconsistent state requiring correction before further operations.

6.1 Content Invariants (5)

ID	INVARIANT	FORMAL STATEMENT
INV-C1	Hash-Content Binding	$\text{SHA-256}(\text{ipfs_get}(\text{CID})) = \text{onChainHash}$
INV-C2	CID Determinism	$\text{ipfs_add}(A) = \text{ipfs_add}(B) \iff A = B$
INV-C3	Build Determinism	$\text{compile}(S, O)*{t_1} = \text{compile}(S, O)* {t_2}$
INV-C4	Manifest Integrity	$\forall f \in M: \text{SHA-256}(\text{read}(f)) = M[f].\text{hash}$
INV-C5	Order Completeness	$\forall e \in O: \exists f \in \text{FS}: \text{name}(f) = e$

6.2 Contract Invariants (6)

ID	INVARIANT	ENFORCEMENT
INV-K1	Author Immutability	`immutable` keyword (bytecode-level)
INV-K2	Genesis Permanence	Constructor-only initialization
INV-K3	Append-Only Editions	No delete/modify functions exist
INV-K4	Author-Only Anchoring	`onlyAuthor` modifier
INV-K5	Sequential Indexing	Array-push semantics
INV-K6	Timestamp Monotonicity	Block timestamp ordering

6.3 Provenance Invariants (3)

ID	INVARIANT	DESCRIPTION
INV-P1	Five-Layer Alignment	All 5 layers consistent for any published edition
INV-P2	Cross-Layer Consistency	Local hash = IPFS hash = on-chain hash

ID	INVARIANT	DESCRIPTION
INV-P3	Timestamp Priority	On-chain timestamp > git commit timestamp

6.4 Pipeline Invariants (3)

ID	INVARIANT
INV-B1	No Uncommitted Anchoring (source committed before on-chain anchor)
INV-B2	Pipeline Ordering (compile → hash → manifest, no reordering)
INV-B3	No Manual Artifacts (derived files produced only by pipeline)

Table 3. Complete invariant registry. Each invariant has a defined verification procedure and monitoring frequency.

7. Case Study: *The 2,500 Donkeys*

7.1 Work Description

The reference implementation accompanies *The 2,500 Donkeys*, a 75,000-word literary work in 31 narrative blocks with 5 embedded artifacts (reproduced contracts, communications, and reports). The work's structure — narrative interleaved with documentary evidence — is representative of investigative journalism, documentary fiction, and academic writing where embedded primary sources require independent verifiability.

7.2 Deployment Timeline

All operations occurred within a single deployment session on February 14–15, 2026.

Table 4. Deployment sequence with on-chain verification.

STEP	OPERATION	BLOCK	TX HASH (PREFIX)	GAS
1	Genesis anchor deployed	83,002,198	`0x9c036d1d...`	1,116,006
2	Edition 2 anchored (31 blocks)	83,004,469	`0xf4bebec4...`	305,935
3	PublishingKernel v1 deployed	83,008,833	`0xfdd3854c...`	—
4	RoyaltyRouter deployed	83,009,717	`0xed744561...`	—
5	Revenue test executed	83,009,932	`0x4d0d1e04...`	—
6	License #0 granted (v1)	83,010,027	`0xf1ae7869...`	—
7	PublishingKernelV2 deployed	83,010,944	`0x89c34617...`	4,804,013
8	AuthorIdentity deployed	83,011,404	—	—
9	12 works registered + 4 contracts linked	83,011,404+	—	—

7.3 Content Metrics

Table 5. Anchored content summary.

CATEGORY	ITEMS	MERKLE LEAVES	ROOT (PREFIX)
Manuscript blocks	31	31	`dd95d121...`
Embedded artifacts	5	5	`9c653a2e...`
Visual assets	10	10	`0e45331c...`
AI prompts	10	10	`32bed9e5...`
Total	**56**	**56**	—
Edition Root	—	—	`6719ed7f...`

The compiled manuscript is 293,550 bytes with SHA-256 digest `d1b9a57f618f0445dc7a5d30d5bf4e707bb4d0cd8d83ceb277f9628d5f68363c`.

7.4 Cross-Chain Anchoring

To mitigate single-chain risk, the edition root is anchored across multiple networks:

CHAIN	METHOD	STATUS
Polygon	Smart contract (5 contracts)	Confirmed (blocks 83,002,198–83,011,404)
Bitcoin	OpenTimestamps (3 calendar servers)	Pending BTC block confirmation
Ethereum	Calldata transaction	Planned

The Bitcoin anchor uses the OpenTimestamps protocol [12], which commits a SHA-256 digest to the Bitcoin blockchain via calendar servers, providing a timestamp proof independent of the Polygon network.

7.5 Revenue Distribution Test

The RoyaltyRouter was tested on mainnet with a 0.01 POL payment distributed across four split recipients (author 70%, illustrator 15%, editor 10%, treasury 5%). The distribution produced **0 wei of dust** (undistributed remainder), confirming basis-point arithmetic precision. The full cycle — receive, distribute, withdraw — completed in two transactions:

- Receive + distribute: `0x4d0d1e04...` (block 83,009,932)
- Withdraw: `0x1b7de1f6...`

7.6 Author Identity Attestation

The AuthorIdentity contract registers 12 published works across multiple platforms (on-chain editions, IPFS content, Amazon listings) and links 4 protocol contracts (LiteraryAnchor, PublishingKernel v1, PublishingKernelV2, RoyaltyRouter), creating a queryable on-chain bibliography that maps a single cryptographic identity to a complete publishing record.

8. Evaluation

8.1 Test Coverage

The protocol is validated by 146 automated tests across all five contracts:

Table 6. Test suite composition.

CONTRACT	TESTS	COVERAGE
LiteraryAnchor	11	Deployment, author verification, edition anchoring
PublishingKernel v1	25	Editions, licensing, Merkle roots, AI provenance
PublishingKernelV2	63	ECDSA, timelock, freeze, admin, canonical cache
RoyaltyRouter	24	Splits, recoupment, withdrawal, edge cases
AuthorIdentity	23	Identity, bibliography, contract linking
Total	**146**	**0 failures**

8.2 Gas Efficiency

Table 7. Gas costs for key operations (Polygon mainnet, February 2026).

OPERATION	GAS USED	APPROXIMATE COST (POL)
Genesis deployment	1,116,006	0.887
Edition anchoring	305,935	~0.10
V2 kernel deployment	4,804,013	~1.60
Work registration (single)	~80,000	~0.03
License grant	~120,000	~0.04
Revenue distribution	~60,000	~0.02

Total infrastructure deployment cost: approximately 5 POL (~$2.50 USD at February 2026 prices). This demonstrates that the model is economically accessible to individual authors — the entire five-contract deployment costs less than a single ISBN registration.

8.3 Verification Performance

Merkle proof verification for any single leaf requires at most $\lceil \log_2(n) \rceil$ hash computations, where n is the number of leaves in the relevant tree. For the largest tree (31 manuscript blocks), this is 5 hash operations — effectively instantaneous on any modern device.

Full edition verification (recomputing all four trees from source files) completes in under 2 seconds on commodity hardware (Node.js v24, Windows).

8.4 Invariant Compliance

All 14 defined invariants (Section 6) hold for the deployed reference implementation. Key verifications:

- **INV-C1 (Hash-Content Binding):** SHA-256 of IPFS-retrieved content matches on-chain hash. Verified via `sha256sum` against `editions[0].sha256Hash`.
- **INV-K1 (Author Immutability):** `author()` returns `0xC91668184736BF75C4ecE37473D694efb2A43978` across all contracts. Cannot be modified (Solidity `immutable`).
- **INV-K2 (Genesis Permanence):** `genesis().ipfsCID` returns `QmVQ79NM3qxAsBpftTG4YhD4KV9sUEmM3WwFrc5vs5g8vK`. No function exists to alter it.
- **INV-P1 (Five-Layer Alignment):** All five layers verified consistent for Edition 0 and Edition 2.

9. Discussion

9.1 Applicability Beyond Fiction

While the reference implementation accompanies a literary work, the architecture is domain-agnostic. Any structured document with identifiable components — journalism with embedded sources, academic papers with supplementary data, legal briefs with cited exhibits — can be decomposed into Merkle trees and anchored using the same pipeline. The four-tree model (text, artifacts, images, prompts) covers the common compositional elements; additional trees can be added for domain-specific content types (audio, video, datasets) without modifying the core protocol.

9.2 Relationship to Existing Publishing Infrastructure

The model operates alongside, not in replacement of, existing publishing systems. An author using this protocol can simultaneously publish through Amazon KDP, submit to traditional publishers, and license through standard agreements. The on-chain anchor provides an independent provenance record that supplements — but does not depend on — institutional attestations.

9.3 AI Provenance Implications

The inclusion of a dedicated prompt tree addresses a gap in current AI disclosure frameworks. While platforms and regulations increasingly require disclosure of AI-generated content, they typically rely on self-reporting with no verification mechanism. By hashing the generation prompts into a Merkle tree and anchoring the root on-chain, this protocol makes AI disclosure verifiable: anyone can confirm that the claimed prompts hash to the anchored root, and any modification to the disclosed prompts would produce a root mismatch.

9.4 Limitations

Key management. The author's private key is a single point of failure. Compromise allows unauthorized edition anchoring (though content verification via SHA-256 provides a detection mechanism). Multi-signature schemes can mitigate this risk; the v2 kernel includes an `admin` role for future multi-sig integration.

Storage availability. IPFS content requires at least one active pin. If all pins are lost, the content becomes unavailable (though the on-chain hash still proves what the content was). Filecoin deals or institutional archiving can provide redundancy.

Blockchain dependency. The on-chain anchor is only as durable as the underlying network. Polygon's proof-of-stake consensus provides strong but not absolute finality. Cross-chain anchoring (Bitcoin, Ethereum) mitigates single-chain risk.

Legal recognition. On-chain timestamps provide strong evidence of existence-at-time but may not constitute legal proof in all jurisdictions. The system provides evidence for legal proceedings, not a substitute for them.

10. Future Work

Institutional integration. Partnerships with university libraries, national archives, and research repositories to serve as persistent IPFS pinners and provide institutional endorsement of the verification model.

Reproducible publishing templates. Generalizing the build pipeline into a configurable template system that journalists, researchers, and digital humanities scholars can adopt for their own works without writing custom code.

Cross-chain finality. Completing the Ethereum calldata anchor and automating the Bitcoin OpenTimestamps verification workflow to provide three-chain redundancy with independent consensus mechanisms.

Formal verification. Applying formal methods to the smart contract suite to provide mathematical guarantees of invariant preservation, complement the current test-based assurance.

Zero-knowledge proofs. Enabling selective disclosure — proving that a specific chapter exists in a published work without revealing the chapter's content — using ZK-SNARK or ZK-STARK proofs over the Merkle tree.

11. Conclusion

We have presented a multi-layer provenance architecture for literary publishing that replaces institutional trust with cryptographic verification. The system anchors 56 content components (novel) and 29 content components (stories) across six Merkle trees into on-chain commitments, supported by deterministic build pipelines, 14 formally defined invariants, and 146 automated tests. The complete infrastructure — five smart contracts, cross-chain anchoring, revenue distribution, and author identity — was deployed and validated on a production network.

Critically, the protocol's generalizability is demonstrated through two structurally distinct literary works: a 75,000-word novel (4 Merkle trees, ElevenLabs TTS, 31 blocks) and a 13-story short fiction collection titled *Private Placement Puppetry* (2 Merkle trees, Kokoro TTS, 16 files). The stories collection uses a different tree topology (manuscript + audio vs. manuscript + artifact + image + prompt), a different TTS engine (local Kokoro vs. cloud ElevenLabs), and a different content structure (independent stories vs. layered narrative blocks) — yet anchors through the same LiteraryAnchor and PublishingKernelV2 contracts using identical verification patterns. This validates LPS-1 as a format-independent protocol rather than a pipeline specific to one manuscript structure.

Every claim in this paper is independently verifiable. The smart contracts are source-verified on a public block explorer. The content is retrievable from content-addressed storage. The build pipelines are open-source and deterministic. The Merkle proofs are computable by anyone with access to the source files and a SHA-256 implementation.

The model demonstrates that verifiable provenance for creative works is not only technically feasible but economically accessible, requiring no specialized infrastructure, no institutional partnerships, and no ongoing operational costs beyond initial deployment. It provides a foundation for reproducible publishing — a paradigm in which the authenticity of a work is not asserted by an authority but computed from first principles.

References

[1] J. Goldsmith and T. Wu, *Who Controls the Internet? Illusions of a Borderless World*, Oxford University Press, 2006.

[2] P. Samuelson, "Digital media and the changing face of intellectual property law," *Rutgers Computer & Technology Law Journal*, vol. 16, pp. 323–340, 1990.

[3] N. Paskin, "Digital Object Identifier (DOI) System," *Encyclopedia of Library and Information Sciences*, 3rd ed., Taylor & Francis, 2010.

[4] R. C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," *Advances in Cryptology — CRYPTO '87*, Lecture Notes in Computer Science, vol. 293, pp. 369–378, Springer, 1988.

[5] S. Chacon and B. Straub, *Pro Git*, 2nd ed., Apress, 2014.

[6] B. Laurie, A. Langley, and E. Kasper, "Certificate Transparency," RFC 6962, Internet Engineering Task Force, June 2013.

[7] J. Benet, "IPFS — Content Addressed, Versioned, P2P File System," arXiv:1407.3561, 2014.

[8] F. Tian, "An agri-food supply chain traceability system for China based on RFID & blockchain technology," *13th International Conference on Service Systems and Service Management*, IEEE, 2016.

[9] M. Sharples and J. Domingue, "The blockchain and kudos: A distributed system for educational record, reputation and reward," *Adaptive and Adaptable Learning*, Lecture Notes in Computer Science, vol. 9891, pp. 490–496, Springer, 2016.

[10] A. Whitaker, "Art and Blockchain: A Primer, History, and Taxonomy of Blockchain Use Cases in the Arts," *Artivate: A Journal of Entrepreneurship in the Arts*, vol. 8, no. 2, pp. 21–46, 2019.

[11] E. Solaiman, "AI-generated content and provenance verification," *AI & Ethics*, vol. 3, pp. 1097–1104, 2023.

[12] P. Todd, "OpenTimestamps: Scalable, Trustless, Distributed Timestamping with Bitcoin," https://opentimestamps.org, 2016.

Appendix A: On-Chain Verification Reference

All artifacts referenced in this paper can be verified using the following public records.

A.1 Contract Addresses (Polygon Mainnet, Chain ID 137)

CONTRACT	ADDRESS	VERIFIED SOURCE
LiteraryAnchor	`0x97f456300817eaE3B40E235857b856dfFE8bba90`	Yes
PublishingKernel v1	`0x511c653fC0F450ba41C42A89A3125CcBf2eFE8ae`	Yes
PublishingKernelV2	`0xca9F6604A9b498DB31d113836E2957c0a9aAE037`	Yes
RoyaltyRouter	`0x44169829489d70aaecbf845870652871C65fC461`	Yes
AuthorIdentity	`0xB9ffa688A8Bb332221030BbBE46bE5bF03323170`	Yes

A.2 Content Identifiers

Novel (The 2,500 Donkeys)

IDENTIFIER	VALUE
IPFS Genesis CID	`QmVQ79NM3qxAsBpftTG4YhD4KV9sUEmM3WwFrc5vs5g8vK`
IPFS Latest CID	`QmPXtEsRwiWuaKmKNA569XAqFNVySN8pwTdGQrvcdpgtMa`
SHA-256 (compiled)	`d1b9a57f618f0445dc7a5d30d5bf4e707bb4d0cd8d83ceb277f9628d5f68363c`
Edition Root	`6719ed7f9e142a39a4a7db533895562bdf5379cf7f9816ed7cbe045ca359594e`
Manuscript Root	`dd95d1216b8e2cb8008c8993dffc54d66b550018a47401dd5df001ff487467d3`
Artifact Root	`9c653a2e453e895f294375818bb872d47d4c90b15859587ba2c5238024202c56`
Image Root	`0e45331c0b80738ff3f491e63b47a5454f162cfe5a1d367e90b709c96c56c638`
Prompt Root	`32bed9e54ed6dc5f4ee8082dce928bd86fb76c36b92d9f949ba12c046674f32c`

Stories (Private Placement Puppetry)

IDENTIFIER	VALUE
IPFS Manuscript CID	`QmahPEAZuWz3dFa55BsNgBEkjBzvWm5M3xbGaRYwm581LV`
IPFS Audio CID	`QmbT7L6zcEvXceYkR362zBJkq75A3Qb2y7wVKcCtKyVhYa`
Manuscript Root	`a73efc2af74e71d59daac1f050a1976e786ebc6fb2ace1e826f41517342173d3`
Edition Root	`a73efc2af74e71d59daac1f050a1976e786ebc6fb2ace1e826f41517342`
Audio Root	`c0049f05391cd72d7738042efd4bc35b3102db82d8ba205e4f66a8`
Audio Edition Root	`90daad5f90d4617a6fa245fff381049a2a15cd4a1b5dcffee8548e`
Combined Hash	`77bb9f5e3f3a6908f96f2519e6b20b7ee15351b08ba962792da430`
File Count	16
Total Size	117,109 bytes

A.3 Key Transactions

Novel Transactions

OPERATION	TX HASH
Genesis deployment	`0x9c036d1d8e946e0d9c8c520d4818e3d211c137478f7a704b733f`
Edition 2 anchor	`0xf4bebec46a32419b8e9455994e92f037268f1ad9e839f21f7bce`
Kernel v1 deployment	`0xfdd3854c4ca2ae46b61ef64df3edce60d423b1cf64e9df57c6cc`
Router deployment	`0xed7445619f58d9517b000e6e35f2323457efea475b4eca4bd325`
Revenue test	`0x4d0d1e04c589a56b7f70f377802b79d8903d559c5bbd53a352bf`
License #0 grant	`0xf1ae786994e1a60cdbafa64cc280d3de9535c34a629b54aba6c2`
V2 kernel deployment	`0x89c34617867efa8592cca2cb17abc20e958f9fe5257a008e3c4e`

Stories Transactions

OPERATION	TX HASH
LiteraryAnchor anchor	`0xd2c9c49d02d31594c5963775973f0646c11382cbba1301e4ef` (Block 83,103,627)
KernelV2 register (Ed. 2)	`0x57caeffe39e26352bc83af72fe6aa2ebc0a02284448aaa9026` (Block 83,103,652)
KernelV2 freeze (Ed. 2)	`0x70b65cdd4146fd24f6649d9143fc12df3751b35db5affecfb3` (Block 83,103,655)

A.4 Author Wallet

`0xC91668184736BF75C4ecE37473D694efb2A43978`

A.5 Source Repository

`https://github.com/FTHTrading/2500-donkeys`

Appendix B: Glossary

TERM	DEFINITION
Block	A discrete unit of manuscript text (chapter, section, sub-section)
Artifact	An embedded primary-source document inserted into the manuscript
Edition	A complete, compiled, and anchored version of the work
Edition Root	SHA-256 of concatenated Merkle roots — the single on-chain commitment
Canonical	The edition currently designated as authoritative

TERM	DEFINITION
Basis Point	One hundredth of one percent (0.01%); 10,000 bp = 100%
Pull Pattern	Withdrawal design where recipients claim funds rather than receiving pushes
Timelock	Mandatory delay between proposing and executing a destructive operation
Freeze	Permanent, irreversible seal on an edition; no further modifications possible

Appendix C: How to Cite This Work

BibTeX

```
@techreport{burns2026deterministic,
  title     = {Deterministic Literary Publishing: A Multi-Layer Provenance
               Model for Verifiable Manuscripts},
  author    = {Burns, Kevan},
  year      = {2026},
  month     = {February},
  version   = {1.1},
  doi       = {10.5281/zenodo.18646886},
  url       = {https://doi.org/10.5281/zenodo.18646886},
  note      = {Independent research. Reference implementation deployed on
               Polygon mainnet. Protocol validated through two literary wor
               a 75,000-word novel and a 13-story short fiction collection.
               All claims verifiable against public on-chain state.}
}
```

APA (7th Edition)

Burns, K. (2026). *Deterministic literary publishing: A multi-layer provenance model for verifiable manuscripts* (Version 1.1). Independent research. https://doi.org/10.5281/zenodo.18646886

Chicago

Burns, Kevan. "Deterministic Literary Publishing: A Multi-Layer Provenance Model for Verifiable Manuscripts." Version 1.1. Independent research, February 2026. https://doi.org/10.5281/zenodo.18646886.

IEEE

K. Burns, "Deterministic Literary Publishing: A Multi-Layer Provenance Model for Verifiable Manuscripts," Independent research, v1.1, Feb. 2026. DOI: 10.5281/zenodo.18646886. [Online]. Available: https://doi.org/10.5281/zenodo.18646886

Version 1.1 — February 15, 2026 Kevan Burns — Independent Researcher — FTH Trading, Norcross, GA ORCID: 0009-0008-8425-939X All on-chain artifacts verified on Polygonscan. Source code: github.com/FTHTrading/2500-

donkeys