

# A Debt Ledger Model for Tracking Proof Obligations in the Prime Counting Function

David Ingle  
VertRule Inc.

January 2026

## Abstract

We present a *debt ledger* model for organizing and verifying proof obligations related to bounds on the prime counting function  $\pi(x)$ . Rather than claiming any new mathematical result, we describe a verification infrastructure that (1) makes explicit which claims have been mechanically verified, (2) isolates unpaid obligations equivalent in difficulty to the Riemann Hypothesis, and (3) provides reproducible artifact chains for all paid rungs. The tail-bound obligation remains unpaid. This paper makes no claim regarding RH.

## Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Motivation . . . . .	3
1.2	Contributions . . . . .	3
<b>2</b>	<b>Problem Statement</b>	<b>3</b>
2.1	The Prime Counting Function . . . . .	3
2.2	Connection to the Riemann Hypothesis . . . . .	4
2.3	The Verification Challenge . . . . .	4
<b>3</b>	<b>The Debt Ledger Model</b>	<b>4</b>
3.1	Core Concepts . . . . .	4
3.2	The NA0 Debt Framing . . . . .	5
3.3	Artifact Chain . . . . .	5
<b>4</b>	<b>The Verification Ladder</b>	<b>5</b>
4.1	Foundation Rungs (R4–R6) . . . . .	5
4.2	Bound Statement (R7) . . . . .	5
4.3	Integrity Rungs (R8–R10) . . . . .	6
4.4	Signature Rungs (R11, R14–R18) . . . . .	6
4.5	Reproducibility Rungs (R20–R22) . . . . .	6
<b>5</b>	<b>Results So Far</b>	<b>6</b>
5.1	Paid Obligations . . . . .	6
5.2	Verification Coverage . . . . .	6
5.3	What This Means . . . . .	7

<b>6</b>	<b>Limits and Remaining Obligations</b>	<b>7</b>
6.1	The Tail Bound Obligation . . . . .	7
6.2	Remaining Obligations . . . . .	7
6.3	Why This Matters . . . . .	8
6.4	What Would Pay the Debt . . . . .	8
<b>7</b>	<b>Reproducibility</b>	<b>8</b>
7.1	Deterministic Builds . . . . .	8
7.2	Assembly Root Rebuild . . . . .	8
7.3	Verification Commands . . . . .	9
7.4	Checkpoint Releases . . . . .	9
<b>8</b>	<b>Conclusion</b>	<b>9</b>
8.1	What This Work Is . . . . .	9
8.2	What This Work Is NOT . . . . .	10
8.3	Future Work . . . . .	10

# 1 Introduction

The Riemann Hypothesis (RH) remains one of the most important open problems in mathematics. Any approach to RH must carefully distinguish between what has been established and what remains conjectural. This paper describes a software infrastructure—a *debt ledger*—designed to make such distinctions explicit and mechanically verifiable.

## Non-Claims

This paper:

- Is **not** a proof of RH.
- Is **not** a new bound on  $\pi(x)$  or  $\psi(x)$ .
- Is **not** verification of RH beyond known computational checks.
- Does **not** provide “evidence suggesting RH is true.”
- Makes **no forward projection** about future progress.

The tail-bound obligation (Section 6) remains unpaid and is equivalent in difficulty to RH itself.

## 1.1 Motivation

When working on difficult mathematical problems, it is easy to lose track of which steps are rigorously established and which contain gaps. The debt ledger model addresses this by:

1. Recording each proof step as a “rung” with explicit contracts.
2. Distinguishing between *paid* obligations (mechanically verified) and *unpaid* obligations (equivalent to the core conjecture).
3. Providing deterministic verification scripts that anyone can run.

## 1.2 Contributions

We contribute:

1. A deterministic verification ladder (R4–R22) with reproducible releases.
2. Explicit isolation of the tail-bound obligation as the unpaid debt.
3. Cryptographic receipt chains (SHA-256, GPG, post-quantum hybrid signatures).
4. Fully reproducible release artifacts built from git tags.

See `RUNG_INDEX.md` for the complete rung map and `STATUS.md` for the current ledger state.

# 2 Problem Statement

## 2.1 The Prime Counting Function

Let  $\pi(x)$  denote the number of primes not exceeding  $x$ . The Prime Number Theorem establishes that

$$\pi(x) \sim \frac{x}{\log x} \quad \text{as } x \rightarrow \infty.$$

A more refined approximation is provided by the logarithmic integral:

$$\text{li}(x) = \int_0^x \frac{dt}{\log t}.$$

## 2.2 Connection to the Riemann Hypothesis

The Riemann zeta function  $\zeta(s)$  is defined for  $\Re(s) > 1$  by

$$\zeta(s) = \sum_{n=1}^{\infty} \frac{1}{n^s}$$

and admits analytic continuation to  $\mathbb{C} \setminus \{1\}$ . The Riemann Hypothesis asserts that all non-trivial zeros of  $\zeta(s)$  have real part  $\frac{1}{2}$ .

RH is equivalent to the bound

$$\psi(x) = x + O(\sqrt{x} \log^2 x),$$

where  $\psi(x) = \sum_{p^k \leq x} \log p$  is the Chebyshev function.

## 2.3 The Verification Challenge

Any claimed bound on  $\pi(x) - \text{li}(x)$  must either:

1. Assume RH (in which case the assumption must be stated explicitly), or
2. Prove the required zero-free region (equivalent to proving RH for the purpose of deriving the bound).

The debt ledger model makes this choice explicit by recording which claims are paid (unconditional) and which carry unpaid debt (conditional on RH or equivalent).

# 3 The Debt Ledger Model

## 3.1 Core Concepts

The debt ledger model organizes proof obligations into three categories:

**Definition 3.1** (Paid Obligation). An obligation is *paid* if it can be mechanically verified from artifacts in the repository without assuming any unproven conjecture.

**Definition 3.2** (Unpaid Obligation). An obligation is *unpaid* if its satisfaction requires proving a statement equivalent in difficulty to RH (or another open problem).

**Definition 3.3** (Rung). A *rung* is a discrete step in the verification ladder, consisting of:

- A contract specifying what the rung establishes and what it does NOT claim.
- Artifacts providing evidence (equations, digests, signatures).
- A verifier script that checks the rung's integrity.
- A release tag for reproducibility.

### 3.2 The NA0 Debt Framing

The key insight is that any bound of the form

$$|\pi(x) - \text{li}(x)| < f(x)$$

for all  $x > x_0$  requires controlling the contribution of zeta zeros. The “tail” contribution from zeros with large imaginary part cannot be bounded without RH or an equivalent statement.

We call this the **NA0 debt**: the obligation to bound the tail contribution from infinitely many zeros. This debt remains unpaid.

See `proof_artifacts/R7_BOUND_STATEMENT/R7_BOUND_STATEMENT.md` for the explicit statement of the bound and its dependencies.

### 3.3 Artifact Chain

Each rung produces artifacts that feed into subsequent rungs:

1. R4–R6: Equation inventory, error bound components, parameter choices.
2. R7: Explicit bound statement citing R4–R6.
3. R8: Regeneration of equation inventory for comparison.
4. R9: Input manifest with frozen digests.
5. R10: Assembly receipt (root hash over all inputs).
6. R11–R18: Signature layers (GPG, post-quantum hybrid).
7. R20: Deterministic rebuild of assembly root.
8. R21: Deterministic rebuild of release zip.
9. R22: Rung index and status ledger.

See `RUNG_INDEX.md` for the complete map.

## 4 The Verification Ladder

The verification ladder consists of rungs R4–R22. Each rung has a contract, artifacts, and a verifier script. We summarize the key rungs below.

### 4.1 Foundation Rungs (R4–R6)

**R4: Transfer Repro.** Records the equation inventory for partial summation transfer. Does not claim proof correctness. See `proof_artifacts/R4_TRANSFER_REPRO/07_EQUATION_INVENTORY.md`.

**R5: Error Bound Source.** Provides a menu of error bound components. Does not claim the bounds are valid or optimal. See `proof_artifacts/R5_ERROR_BOUND_SOURCE/R5_MENU.md`.

**R6: Instantiation.** Records specific parameter choices. Does not claim optimality. See `proof_artifacts/R6_INSTANTIATION/R6_INSTANTIATION_RECORD.md`.

### 4.2 Bound Statement (R7)

**R7: Bound Statement.** Provides an explicit statement of the bound, citing R4–R6 for its components. Does NOT claim RH. See `proof_artifacts/R7_BOUND_STATEMENT/R7_BOUND_STATEMENT.md`.

The bound statement explicitly records which terms are paid (verified) and which carry unpaid debt (the tail contribution).

### 4.3 Integrity Rungs (R8–R10)

**R8: Comparison Run.** Regenerates the equation inventory and compares to the frozen version. Detects any drift. See `proof_artifacts/R8_COMPARISON_RUN/R8_COMPARISON_RECEIPT.md`.

**R9: No-Surprise Assembly.** Freezes the input manifest with SHA-256 digests. Ensures no hidden inputs. See `proof_artifacts/R9_NO_SURPRISE_ASSEMBLY/R9_INPUT_MANIFEST.txt`.

**R10: Assembly Receipt.** Computes a canonical root hash over all proof artifacts. Forms the signing payload. See `proof_artifacts/R10_ASSEMBLY_RECEIPT/R10_RECEIPT.json`.

### 4.4 Signature Rungs (R11, R14–R18)

**R11: GPG Signature.** Optional detached GPG signature over R10 assembly root. See `proof_artifacts/R11_SIGNATURE/sigs/dave.asc`.

**R14: Key Capture.** Records GPG fingerprints for attribution. See `proof_artifacts/R14_KEY_CAPTURE/`

**R15–R17: Post-Quantum Signatures.** Provides hybrid PQ signatures (P-384 + ML-DSA-65) for quantum-resistant attestation. See `proof_artifacts/R15_PQ_SIGNATURE/sigs_pq/dave.p384_ml`

### 4.5 Reproducibility Rungs (R20–R22)

**R20: Assembly Root Rebuild.** Deterministically rebuilds R10 from the verify surface. Proves the root hash is reproducible. See `proof_artifacts/R20_REBUILD_ASSEMBLY_ROOT/R20_CONTRACT.md`.

**R21: Release Zip Rebuild.** Deterministically rebuilds the release zip from a git tag. Ensures byte-identical releases. See `proof_artifacts/R21_RELEASE_ZIP_REBUILD/R21_CONTRACT.md`.

**R22: Rung Index.** Provides the canonical map of all rungs and current ledger status. See `RUNG_INDEX.md` and `STATUS.md`.

## 5 Results So Far

We summarize what the verification ladder has established.

### 5.1 Paid Obligations

The following obligations have been paid (mechanically verified):

1. **Classical signature (GPG):** Dave’s detached signature over the R10 assembly root. See `proof_artifacts/R11_SIGNATURE/sigs/dave.asc`.
2. **Post-quantum hybrid signature:** P-384 + ML-DSA-65 signature over the assembly root. See `proof_artifacts/R15_PQ_SIGNATURE/sigs_pq/`.
3. **Assembly root rebuild:** Deterministic regeneration of R10 from the verify surface. See `VERIFY_R20_REBUILD_ASSEMBLY_ROOT.sh`.
4. **Release reproducibility:** Deterministic zip rebuild from git tag. See `scripts/rebuild_release_zip.sh`.
5. **Key fingerprint capture:** GPG fingerprint recorded for attribution. See `proof_artifacts/R14_KEY_CAPTURE/`.
6. **PQ tooling receipt:** Backend and version captured for PQ signatures. See `proof_artifacts/R16_PQ_TOOLING/`.

### 5.2 Verification Coverage

The main verifier runs 23 steps:

- Steps 1–11: Repository integrity (git status, required files, redaction, exhibits, contribution ledger).

- Steps 12–23: Rung verifiers (R4–R20).

To run verification:

```
VR_STRICT=1 ./VERIFY.sh
```

Expected output:

```
=== VERIFICATION PASSED ===
```

### 5.3 What This Means

The verification ladder establishes that:

1. The equation inventory, error bounds, and parameter choices are frozen and reproducible.
2. The assembly root hash is deterministic.
3. Signed attestations are cryptographically bound to the artifacts.
4. Release zips can be rebuilt identically from git tags.

This provides *infrastructure* for tracking proof obligations, but does NOT prove any mathematical claim about  $\pi(x)$  or RH.

## 6 Limits and Remaining Obligations

This section explicitly states what remains unpaid and why.

### 6.1 The Tail Bound Obligation

The central unpaid obligation is the **tail bound**: controlling the contribution of zeta zeros  $\rho = \beta + i\gamma$  with  $|\gamma| > T$  for any fixed  $T$ .

Any bound of the form

$$|\psi(x) - x| < C\sqrt{x} \log^2 x$$

requires summing over *all* zeros. The sum converges, but its size depends on the location of zeros. Without knowing that all zeros lie on the critical line, the tail cannot be bounded effectively.

**Status:** UNPAID. Equivalent in difficulty to RH.

See `proof_artifacts/R7_BOUND_STATEMENT/R7_BOUND_STATEMENT.md` for the explicit statement of where this debt enters.

### 6.2 Remaining Obligations

Item	Status	Notes
Tail bound proof	UNPAID	The dragon is boxed, not slain
RH claim	NOT MADE	This repo makes no claim regarding RH
Proof reproduction	NOT DONE	No proofs are reproduced
External audit	NOT DONE	Third-party review pending

### 6.3 Why This Matters

The debt ledger model makes explicit a common source of errors in mathematical claims: conflating “we have organized the pieces” with “we have proven the result.” By explicitly recording unpaid obligations, we prevent scope creep and premature claims.

The infrastructure is valuable even with unpaid debt because it:

1. Provides a clear target for future work.
2. Ensures that any future claim of progress can be verified against the frozen artifacts.
3. Maintains an audit trail for reproducibility.

### 6.4 What Would Pay the Debt

The tail bound obligation would be paid by either:

1. A proof of RH (establishing all zeros have  $\beta = \frac{1}{2}$ ).
2. An unconditional zero-density estimate sufficient for the required bound.
3. A fundamentally different approach that avoids the explicit sum over zeros.

None of these is claimed or implied by this work.

## 7 Reproducibility

A key design goal is that all artifacts can be independently reproduced.

### 7.1 Deterministic Builds

Release zips are built deterministically:

- Fixed file timestamps (2024-01-01 00:00:00 UTC).
- Stable file ordering (LC\_ALL=C sort).
- No extended attributes (stripped on macOS).
- No zip timestamps (zip -X flag).

To rebuild a release:

```
./scripts/rebuild_release_zip.sh <tag> /tmp/rebuilt.zip
```

The resulting SHA-256 should match the published release.

### 7.2 Assembly Root Rebuild

The R10 assembly root can be regenerated from the verify surface:

```
./scripts/rebuild_r10_assembly_root.sh
```

The output should match `proof_artifacts/R10_ASSEMBLY_RECEIPT/R10_RECEIPT.json`.  
See `VERIFY_R20_REBUILD_ASSEMBLY_ROOT.sh` for the automated check.



### 7.3 Verification Commands

**Baseline verification (23 steps):**

```
VR_STRICT=1 ./VERIFY.sh
```

**Full GPG verification (requires pubkey import):**

```
gpg --import <pubkey>
gpg --verify proof_artifacts/R11_SIGNATURE/sigs/dave.asc \
  proof_artifacts/R11_SIGNATURE/R10_ASSEMBLY_ROOT.txt
```

**Full PQ verification (requires oqsprovider):**

```
OPENSSL_MODULES=/path/to/openssl-modules \
  ./scripts/verify_pq_sig.sh p384_mldsa65 \
  proof_artifacts/R15_PQ_SIGNATURE/sigs_pq/dave.p384_mldsa65.sig \
  proof_artifacts/R11_SIGNATURE/R10_ASSEMBLY_ROOT.txt \
  proof_artifacts/R15_PQ_SIGNATURE/pubkeys/dave.p384_mldsa65.pub
```

### 7.4 Checkpoint Releases

Each rung has an associated git tag and GitHub release:

- Tag format: `r<N>-<name>`
- Asset: `r<N>-<name>-verify-surface.zip`
- Contains: Verifier scripts, proof artifacts, documentation.

See `RUNG_INDEX.md` for the complete list of tags.

## 8 Conclusion

We have presented a debt ledger model for organizing proof obligations related to bounds on the prime counting function. The key contributions are:

1. A deterministic verification ladder (R4–R22) with explicit contracts for each rung.
2. Clear separation between paid obligations (mechanically verified) and unpaid obligations (equivalent to RH).
3. Reproducible release artifacts with cryptographic attestations.
4. An explicit statement of remaining debt: the tail bound is unpaid.

### 8.1 What This Work Is

- A framework for tracking proof obligations.
- A demonstration of reproducible mathematical artifact pipelines.
- An honest accounting of what has and has not been established.

## 8.2 What This Work Is NOT

- A proof of RH.
- A claim that RH is true or false.
- Evidence suggesting RH is more or less likely.
- A substitute for peer review of mathematical content.

## 8.3 Future Work

The debt ledger infrastructure supports future work by:

1. Providing a clear target (pay the tail bound obligation).
2. Ensuring any claimed progress can be verified against frozen artifacts.
3. Maintaining an audit trail for reproducibility.

The dragon is boxed, not slain. The box is well-documented.

## References