Timeproof: A Cryptographic Protocol for Probabilistic Truth Verification

Nathaniel Joseph Houk
Independent Researcher
Email: njhouk@gmail.com

January 29, 2025

Abstract

This paper presents Timeproof, a novel cryptographic protocol for probabilistic truth verification utilizing blockchain-based temporal persistence. The proposed framework integrates secure commitment schemes with economic incentives to establish a verifiable system for mathematical claims. A key innovation is the introduction of a temporal verification function that quantifies the probability of a claim's validity based on its duration of remaining uncontested. The research provides three main contributions: (1) a formal mathematical framework for probabilistic truth verification, (2) a practical implementation using Bitcoin's blockchain with proven security properties, and (3) a comprehensive game-theoretic analysis of the protocol's incentive structure. Security proofs are established under standard cryptographic assumptions, demonstrating the protocol's applicability to significant mathematical problems including the P vs NP conjecture. The analysis also addresses post-quantum security considerations and identifies potential attack vectors.

Timeproof Protocol

A **timeproof** is a cryptographic protocol and mathematical framework designed to establish probabilistic truth and priority for mathematical assertions through temporal persistence on the Bitcoin blockchain. First implemented in 2015, it combines cryptographic escrow mechanisms with blockchain timestamping to create an incentive-driven system for mathematical discovery.

1.0.1 Motivation

Traditional mathematical proofs require complete deductive arguments, which can be challenging for complex problems like P vs NP. Timeproof offers an alternative approach using cryptographic commitments and temporal verification, addressing the challenge of establishing priority and probabilistic truth for mathematical claims in a verifiable, incentive-compatible manner.

1.0.2 Contributions

The Timeproof protocol's key contributions include:

- Formal definition of the Timeproof protocol and theorems
- Probabilistic verification model using survival analysis
- Bitcoin-based implementation with cryptographic escrow
- Security analysis against preimage and collusion attacks
- Computational complexity implications, particularly for P vs NP

1.1 Historical Context

The timeproof protocol emerged from a specific application to the P versus NP problem. The first implementation occurred on May 30, 2015, at 11:43:39 UTC, when a proposed proof of

P = NP was committed to the Bitcoin blockchain mainnet through the Proof of Existence service [22].

This timeproof was recorded in a Bitcoin transaction with the hash
`378933871106d84dcac011be06770acfe0491f6f80fe0402f399e0f548ebb99f`,
where the proof documents' SHA-256 hash
`33095308466a6e896cb3dc71f40a1710ab9f4fc29205ecf55b34362f7ed52e66`
was permanently embedded in the blockchain.

The implementation utilized a Bitcoin address
`1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa`
(the genesis block address) as the cryptographic escrow point, creating what would later be formalized as the timeproof protocol. This historical instance serves as both the genesis and a practical demonstration of the protocol's core concepts: timestamped commitment of mathematical claims, cryptographic escrow mechanisms, and blockchain-based verification.

The temporal persistence of this initial implementation provides an empirical foundation for the protocol's effectiveness. As of January 29, 2025, the continued existence of the unclaimed bounty contributes to the accumulating probabilistic evidence regarding the original claim, demonstrating the practical application of the verification function $V(\Delta t)$ described in this work.

MAD Paradox Integration

The original 2015 implementation contained an early formulation of the MAD (Mathematical Asymmetric Demonstration) paradox:

$$\text{MAD}(P = NP) \land (P \neq NP) \land \neg(P = NP \land P \neq NP)$$

This paradox manifests in the protocol through:

- Non-constructive proof existence via blockchain persistence
- Constructive proof impossibility under finite-dimensional constraints
- Infinite-dimensional multiverse requirement for consistency

1.2 Core Protocol Components

1.2.1 Encrypted Assertion

A mathematical claim, such as a proof that (P = NP), is encrypted using a key derived from an unsolved cryptographic problem [5]. For instance, let X be an unknown value, and define the key as $K = \text{SHA-256}(X)$. The proof is then encrypted with K, ensuring its confidentiality until X is discovered.

1.2.2 Bounty Mechanism

Funds are locked in a Bitcoin address whose private key is deterministically generated from the cryptographic problem. The protocol anticipates two possible outcomes:

- **Unclaimed Bounty**: As time progresses without the problem being solved, there is increasing probabilistic evidence supporting the negation of the claim (e.g., $P \neq NP$).
- **Claimed Bounty**: Solving the cryptographic problem allows the claimant to access the funds and the decrypted proof, with the blockchain providing a verifiable timestamp of the original assertion and the author's signature to authenticate the claim's origin.

1.2.3 Temporal Verification

The Bitcoin blockchain mainnet [13] is utilized for immutable timestamping. Building on foundational work in secure timestamping [9], tools such as *OpenTimestamps* and *Proof of Existence* facilitate this process:

- **OpenTimestamps**: Provides a standard for blockchain timestamping by aggregating file hashes into a Merkle tree, committing the Merkle root to the Bitcoin blockchain, and returning a proof file that can be independently verified [21].
- **Proof of Existence**: Offers a service to certify the existence of a document at a specific time by embedding its hash in the Bitcoin blockchain [22].

A verification function $V(\Delta t)$ quantifies confidence over time, building on established probabilistic proof systems [7] and timed commitment schemes [12]. The mechanism incorporates modern verifiable delay functions [18] to ensure:

- As Δt increases without a bounty claim, $V(\Delta t) \to 1$, suggesting that the claim is likely false.
- If the bounty is claimed, $V(\Delta t) \to 0$, indicating that the claim has been proven true.

Verification Function $V(\Delta t)$

The verification function $V(\Delta t)$ is modeled as a Weibull distribution:

$$V(\Delta t) = 1 - \exp\left(-\left(\frac{\Delta t}{\eta}\right)^{\beta}\right) \tag{1.1}$$

where:

- η is the scale parameter (characteristic time)
- β is the shape parameter (capturing increasing/decreasing hazard rate)

Justification for Weibull Distribution: The Weibull distribution is particularly appropriate because:

- It can model both increasing and decreasing failure rates over time, which aligns with the reality that the probability of solving a mathematical problem may change as new techniques emerge
- It has been successfully used in similar contexts of time-to-event analysis in computational complexity [15]
- The distribution's flexibility allows it to model different problem-solving dynamics:
  - $\beta < 1$: Decreasing hazard rate (solutions become less likely over time)
  - $\beta = 1$: Constant hazard rate (exponential distribution)
  - $\beta > 1$: Increasing hazard rate (solutions become more likely over time)
- Empirical studies of mathematical problem-solving times show Weibull-like distributions [17]

Alternative Models: While the Weibull distribution is our primary model, we acknowledge that other distributions could be considered:

- Poisson process with time-varying rate
- Log-normal distribution for solution times
- Gamma distribution for accumulated solving effort

However, the Weibull distribution provides the best fit for our purposes due to its flexibility and interpretability of parameters.

Algorithm 1 Timeproof Address Selection

procedure SELECTESCROWADDRESS(H, S)
Input: $H \in \{0,1\}^{256}$ ▷ Proof hash
Input: $S \in \{0,1\}^{256}$ ▷ Salt
Output: Bitcoin address A
    $k \leftarrow \text{HMAC-SHA256}(key = S, msg = H) \bmod n$ ▷ secp256k1 curve order $n =$ FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE BAAEDCE6 AF48A03B BFD25E8C D0364141
    $K \leftarrow k \cdot G$ ▷ Public key derivation
    Address Construction
    $K_{\text{bytes}} \leftarrow \text{bytes}(K.x) \parallel \text{bytes}(K.y)$
    $H_{\text{sha}} \leftarrow \text{SHA-256}(K_{\text{bytes}})$
    $H_{\text{ripe}} \leftarrow \text{RIPEMD-160}(H_{\text{sha}})$
    $A_{\text{net}} \leftarrow \texttt{0x00} \parallel H_{\text{ripe}}$
    $C \leftarrow \text{first\_4\_bytes}(\text{SHA-256}^2(A_{\text{net}}))$
    $A_{\text{final}} \leftarrow \text{Base58}(A_{\text{net}} \parallel C)$
    return A_{final}
end procedure

1.2.4 Address Generation

Where the algorithm makes use of:

- $\text{SHA-256}^2(x) \equiv \text{SHA-256}(\text{SHA-256}(x))$
- $\parallel$ denotes byte concatenation
- All conversions between integers and bytes use big-endian encoding
- Security depends on the elliptic curve discrete logarithm problem (ECDLP) [23]
- Key derivation follows NIST SP 800-56A Rev. 3 key derivation guidelines

1.2.5 The Role of the Salt

The salt S in the Timeproof protocol serves several critical purposes, even though the same proof can be published multiple times with definitive timestamps:

Preventing Precomputation Attacks

- Without a salt, an attacker could precompute potential solutions X' for a known proof hash H and derive the corresponding Bitcoin address A in advance.
- The salt ensures that the Bitcoin address A is unique for each instance of the proof, even if the same proof is published multiple times. This makes precomputation attacks infeasible.

Ensuring Uniqueness of the Bitcoin Address

- The Bitcoin address A is generated using both the proof hash H and the salt S. This ensures that even if the same proof is published multiple times, each instance will have a unique Bitcoin address.
- This uniqueness is critical for the bounty mechanism, as it prevents confusion or conflicts between multiple instances of the same proof.

Security Against Rainbow Table Attacks

- The salt acts as a defense against rainbow table attacks, where an attacker precomputes hashes for common inputs. By introducing a unique salt, the attacker cannot reuse precomputed tables, making brute force attacks significantly more difficult.

Supporting Multiple Proofs

- If multiple proofs are published for the same mathematical claim (e.g., different approaches to proving $P = NP$), the salt ensures that each proof has a distinct Bitcoin address. This allows for independent verification and bounty claims for each proof.

Enhancing Privacy

- The salt adds an additional layer of obfuscation, making it harder for an attacker to link multiple instances of the same proof or infer relationships between different proofs.

Example Scenario

Suppose Alice publishes a proof of $P = NP$ with salt S_1, and Bob publishes the same proof with salt S_2. The Bitcoin addresses A_1 and A_2 will be different because:

$$A_1 = \text{SelectEscrowAddress}(H, S_1)$$

$$A_2 = \text{SelectEscrowAddress}(H, S_2)$$

Even though the proof hash H is the same, the salts S_1 and S_2 ensure that $A_1 \neq A_2$. This prevents any ambiguity in the bounty mechanism and ensures that each instance of the proof is treated independently.

Security Implications

The security implications can be summarized as follows:

- **Without Salt**: An attacker could precompute solutions for a known proof hash H and derive the Bitcoin address A in advance, undermining the security of the protocol.
- **With Salt**: The attacker must solve the puzzle for each unique salt, making the attack computationally infeasible.

1.2.6 Salt Generation and Security

The salt S is generated as a 256-bit random value, ensuring sufficient entropy to prevent brute force attacks. The security of the protocol relies on the unpredictability of S, as it prevents precomputation attacks and ensures the uniqueness of the Bitcoin address A. The salt is stored alongside the proof hash H and the timestamp, allowing for independent verification of the proof's integrity.

1.2.7 Adversarial Considerations

The attacker does not initially know the hash K, which is derived from an unknown value X using the SHA-256 function, i.e., $K = \text{SHA-256}(X)$. The attacker's goal is to find X such that $\text{SHA-256}(X) = K$. Here's how the attacker might attempt to discover K:

Attacker's Process

- **Public Information**: The hash K is not directly public, but the attacker can infer it from the Bitcoin address A and the proof hash H. The Bitcoin address A is

generated from K and the salt S using the `SelectEscrowAddress` algorithm. If the attacker knows H and S, they can attempt to derive K by solving the ECDLP, which is computationally infeasible.

- **Brute Force Search**: The attacker performs a brute force search to find X such that SHA-256$(X) = K$. This involves generating potential candidates X', computing SHA-256(X'), and comparing the result to K.
- **Optimization Techniques**: The attacker may use optimization techniques to reduce the search space, such as parallel computing or cryptographic attacks. Rainbow tables are ineffective against high-entropy inputs like X.
- **Verification**: Once a candidate X' is found such that SHA-256$(X') = K$, the attacker verifies the solution by decrypting the proof using K and checking the validity of the decrypted proof.

Multiple Publications

The Timeproof protocol leverages Bitcoin's blockchain to provide immutable and verifiable timestamps, ensuring that even if the same proof is published multiple times, the first publication can be definitively identified. Here's why this works:

- **Immutable Timestamps**: The Bitcoin blockchain provides a tamper-proof record of when a proof was first published. Once a proof is embedded in the blockchain, its timestamp cannot be altered or backdated.
- **Proof Hash Uniqueness**: Each proof is hashed using SHA-256, producing a unique identifier (H). Even if the same proof is published multiple times, the hash H remains the same, and the earliest timestamp associated with H is definitive.
- **First-to-Publish Priority**: The protocol ensures priority by recording the first instance of the proof's hash in the blockchain. Subsequent publications of the same proof will have later timestamps, which are irrelevant for establishing priority. This mechanism is particularly important in scenarios where:
  - **Simultaneous Discoveries**: In major mathematical breakthroughs like P vs NP, it's common for multiple research groups to arrive at similar solutions independently around the same time. The blockchain timestamp provides an immutable record of who published first.

- **Verifiable Proof of Priority**: The timestamp serves as cryptographic evidence of when the proof was first committed, preventing disputes about who discovered the solution first.
- **Independent Verification**: The timestamp allows anyone to verify the exact moment the proof was published, without relying on potentially biased third-party witnesses or publication timelines.
- **Global Standardization**: Blockchain timestamps provide a globally consistent time reference, eliminating issues with timezone differences or local clock inaccuracies.
- **Preventing Backdating**: The immutability of the blockchain prevents any party from fraudulently claiming an earlier publication date.

The protocol's timestamping mechanism is designed to handle the likely scenario where multiple groups may independently solve P vs NP simultaneously, ensuring that credit and recognition are properly assigned to the first discoverer.

Example Scenario

Suppose the mathematical claim is a proof that $P = NP$, encrypted with key $K = \text{SHA-256}(X)$. The attacker's steps would be:

1. **Identify the Puzzle**: The attacker knows the Bitcoin address A and the proof hash H.
2. **Brute Force Search**: The attacker starts generating random values X' and computes $\text{SHA-256}(X')$.
3. **Optimization**: The attacker uses a distributed computing network to speed up the search.
4. **Verification**: When $\text{SHA-256}(X') = K$, the attacker decrypts the proof and verifies its correctness.
5. **Claiming the Bounty**: The attacker uses X' to derive the private key and transfers the funds.

Security Considerations

The protocol's mainnet security relies on several key factors:

- **Quantum Resistance**: While currently secure, future quantum computers could potentially break the cryptographic primitives used. The protocol should be designed with quantum-resistant alternatives in mind
- **Network Security**: The security of the Bitcoin mainnet network itself is crucial, as any compromise could affect the integrity of the timestamps
- **Key Management**: Proper key management practices are essential to prevent loss of funds or proof access
- **Computational Difficulty**: SHA-256 is designed to be computationally intensive to reverse, making brute force searches impractical without significant resources
- **Economic Incentives**: The bounty must be large enough to justify the computational cost of solving the puzzle
- **Temporal Persistence**: The longer the puzzle remains unsolved, the stronger the probabilistic evidence that the claim is falsxe
- **Preventing Precomputation Attacks**: The salt ensures that the Bitcoin address is unique for each instance of the proof, making precomputation attacks infeasible
- **Rainbow Table Resistance**: The salt acts as a defense against rainbow table attacks, making brute force attacks significantly more difficult

By solving the puzzle, the attacker not only gains access to the encrypted proof but also claims the financial bounty, thereby providing a definitive resolution to the mathematical claim.

1.2.8 Security Proofs

Definition 1 (Timeproof Security Game)**.** *Let $\mathcal{A}$ interact with challenger $\mathcal{C}$ in the following phases:*

1. ***Setup****: $\mathcal{C}$ generates $(pk, sk) \leftarrow KeyGen(1^\lambda)$ 2.* ***Commit Phase****: $\mathcal{A}$ makes polynomial many* COMMIT *queries with (H_i, S_i) 3.* ***Verify Phase****: $\mathcal{A}$ makes* VERIFY *queries on (t_j, σ_j) 4.* ***Forge****: $\mathcal{A}$ outputs forgery (t^*, σ^*)*

$\mathcal{A}$ wins if $Verify(pk, t^*, \sigma^*) = 1$ for unqueried t^* with non-negligible advantage.

Theorem 1 (Security Reduction)**.** *Under the post-quantum security of SHA3-256 and CRYSTALS-Dilithium, for any PPT adversary $\mathcal{A}$,*

$$Adv_{\mathcal{A}}^{Timeproof}(\lambda) \leq \frac{q_H^2}{2^{256}} + Adv_{ML\text{-}DSA}^{EUF\text{-}CMA}(\lambda)$$

where q_H is the number of hash queries and ML-DSA is Module-Lattice Digital Signature Algorithm.

Proof. We reduce the security of the timeproof protocol to the hardness of SHA-256 preimage resistance and ECDLP:

1. **Timestamp Forgery**: Assume an adversary $\mathcal{A}$ can forge a timestamp. This would require either:

- Breaking Bitcoin's blockchain security (reduced to ECDLP)
- Creating a valid proof with an earlier timestamp (reduced to SHA-256 preimage resistance)

2. **Proof Confidentiality**: Assume $\mathcal{A}$ can decrypt the proof without solving the puzzle. This would require:

- Finding a preimage X' such that SHA-256$(X') = K$ (directly reduces to SHA-256 preimage resistance)

3. **Address Integrity**: Assume $\mathcal{A}$ can generate a valid address without the correct H and S. This would require:

- Solving the ECDLP to derive the private key from the public key

Since all three cases reduce to problems assumed to be hard, the protocol is secure. □

1.3 Protocol Implementation

1.3.1 Algorithmic Details

The Timeproof protocol is implemented using Bitcoin-compatible cryptographic primitives. Below is the core algorithm in pseudocode:

Algorithm 2 Timeproof Protocol

procedure CREATEPROOF($proof, pubkey, privkey, bounty$)
  $proofHash \leftarrow$ SHA-256($proof$)
  $signature \leftarrow$ Sign($privkey, proofHash$)
  $escrowAddress \leftarrow$ DeriveAddress($proofHash$)
  $locktime \leftarrow$ CurrentTime + 1,000,000 years
  Verify $bounty \geq 1$ BTC
  Return ($proofHash, signature, escrowAddress, locktime$)
end procedure
procedure VERIFYPROOF($proofHash, signature, pubkey, locktime$)
  Verify VerifySig($pubkey, signature, proofHash$)
  Verify CurrentTime $\geq locktime$
  Return True if all verifications pass
end procedure
procedure CLAIMBOUNTY($solution, escrowAddress, proofHash$)
  $derivedAddress \leftarrow$ DeriveAddress($proofHash$)
  Verify $escrowAddress = derivedAddress$
  Verify ValidateSolution($solution$)
  Transfer funds from $escrowAddress$ to claimant
  Return True if claim successful
end procedure

The implementation includes the following key components:

- **Proof Creation**: Generates cryptographic commitments and signatures
- **Proof Verification**: Validates proof integrity and temporal requirements
- **Bounty Claim**: Processes solution verification and fund transfer

1.3.2 Workflow

The timeproof protocol operates through the following sequence:

1. Encrypt the proof document $\mathcal{P}$ with key $K = \text{SHA-256}(X)$
2. Create author signature $\sigma = \text{Sign}_{sk}(H \parallel S \parallel \text{timestamp})$

3. Derive Bitcoin address:
   - Compute $H = \text{SHA-256}(\text{EncryptedProof})$
   - Generate $A = \text{SelectEscrowAddress}(H, S)$
4. Record timestamp for tuple (σ, H, S, A)
5. Transfer funds to address A

1.3.3 Practical Considerations

The Timeproof protocol faces several practical challenges:

- **Network Latency**: The protocol relies on the Bitcoin blockchain for timestamping, which introduces delays due to network latency and block confirmation times. This can impact the timeliness of proof verification.
- **Scalability**: As the number of proofs increases, the protocol may face scalability issues due to the limited throughput of the Bitcoin blockchain. Future work could explore the use of layer-2 solutions or alternative blockchains to address this limitation.
- **Storage Costs**: Storing encrypted proofs and associated metadata on the blockchain can be costly. The protocol could benefit from off-chain storage solutions, such as IPFS, to reduce storage costs while maintaining verifiability.

1.4 Mathematical Foundations

A timeproof T for a statement S is defined as a tuple (E, V, t), where:

- E: Encrypted proof or disproof.
- $V(\Delta t)$: Verification function defined as:

$$V(\Delta t) = 1 - \exp\left(-\int_0^{\Delta t} \lambda(t)dt\right)$$

  where $\lambda(t)$ follows a non-homogeneous Poisson process with:

$$\lambda(t) = \frac{f_X(t)}{1 - F_X(t)}$$

  for probability density function f_X and CDF F_X of solution time distribution, following rigorous survival analysis [27].

1. **Temporal Monotonicity**: Proven via survival analysis [25]:

$$S(t) = \exp\left(-\int_0^t h(u)du\right)$$

where hazard function $h(u) \geq 0 \ \forall u$, making $S(t)$ non-increasing

2. **Asymptotic Certainty**: $\lim_{\Delta t \to \infty} V(\Delta t) = 1$ if S is false.

3. **Zero-Knowledge Priority**: Requires satisfying

$$\text{VerifySig}_{pk}(\sigma, H \parallel S \parallel t) = 1$$

without revealing sk or proof contents

These verification properties build on fundamental results in interactive proof systems [10], with security guarantees derived from established cryptographic primitives [2, 6].

Theorem 2 (Houk's Theorem of Temporal Verification)**.** *Let T be a timeproof for statement S with verification function $V(\Delta t)$. Then:*

1. (Temporal Monotonicity) $V(\Delta t)$ is non-decreasing in Δt 2. (Asymptotic Certainty) $\lim_{\Delta t \to \infty} V(\Delta t) = 1$ if S is false 3. (Bounded Verification) For any finite Δt, $V(\Delta t) < 1$ 4. (Solution Detection) If S is true, $\exists t_0$ such that $V(t) = 0$ for all $t \geq t_0$

Furthermore, these properties hold under the following assumptions:

- *The cryptographic hash function is preimage-resistant*
- *The blockchain provides immutable timestamps*
- *Economic incentives are properly aligned*

Proof. 1. **Temporal Monotonicity**: Follows from the Weibull distribution's properties and the non-decreasing nature of the cumulative hazard function.

2. **Asymptotic Certainty**: As $\Delta t \to \infty$, the probability of the problem remaining unsolved approaches zero, making $V(\Delta t) \to 1$.

3. **Bounded Verification**: For finite Δt, there's always non-zero probability of solution, so $V(\Delta t) < 1$.

4. **Solution Detection**: When S is true, the solution will be found at some finite time t_0, after which $V(t) = 0$ for all $t \geq t_0$.

The assumptions ensure the protocol's security and proper functioning:

- Preimage resistance prevents premature solution discovery

- Immutable timestamps guarantee temporal integrity
- Economic incentives maintain protocol participation

□

1.5 Case Study: P vs NP Problem

To apply the timeproof protocol to the (P) vs (NP) problem [3, 4]:

1. The prover encrypts a proof that $P = NP$ using a key derived from SHA-256(X).
2. A bounty address is created at `1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa`, containing funds that are claimable only by solving SHA-256(X).
3. The system continuously monitors:
   - The elapsed time since the timestamp, providing probabilistic evidence for $P \neq NP$.
   - The blockchain for any bounty claims, which would serve as definitive proof that $P = NP$.

1.5.1 Non-Constructive Proof via MAD Paradox

The Timeproof protocol formally captures the MAD paradox through:

Theorem 3 (Houk's MAD Incompleteness)**.** *For any finite-dimensional universe U_n with $n < \infty$:*

1. $\exists$ *non-constructive proof that* $P = NP$
2. $\nexists$ *efficient algorithm for* $SAT \in P$
3. $U_n \models MAD \to \bot$ *(paradoxical explosion)*

Thus U_∞ required for consistency.

Proof. Via protocol's temporal verification function:

1. $V(\Delta t) \to 1$ demonstrates (1)
2. Unclaimed bounty proves (2)
3. Blockchain persistence proves U_∞ requirement

□

Physical Manifestation

The protocol's blockchain persistence empirically demonstrates:

- Gödel-Hawking incompleteness: As formalized in [34]
- M-theoretic requirements: $N = 11$ dimensions insufficient, requiring $N \to \infty$
- Quantum gravity interface: Blockchain entropy mirrors holographic principle

1.5.2 Oracle Verification Mechanism

An oracle $\mathcal{O}$ capable of solving NP problems in P time can use the Timeproof protocol to demonstrate this capability without revealing their method:

1. Prover submits $c = \text{Commit}(X, \pi)$ where π represents their P=NP acceleration capability
2. $\mathcal{O}$ runs:

while $\Delta t < T_{\max}$ **do**
    Select random NP-complete problem P_i
    Solve P_i using acceleration capability
    Generate proof π_i of solution
    Commit π_i to blockchain
    return ZK-SNARK(π_i) ▷ Proves solution without revealing method
return $P = NP$ acceleration capability with confidence $1 - \exp(-\lambda T_{\max})$

3. Verification uses:

$$\Psi(t) = \Theta\left(\frac{1}{n}\sum_{i=1}^{n} V_i(t) - \frac{1}{2}\right)\begin{cases}1 & \text{if } \exists\text{consistent acceleration} \\ 0 & \text{otherwise}\end{cases} \tag{1.2}$$

where Θ is the Heaviside step function and $V_i(t)$ verifies individual problem solutions

1.5.3 Phase Transition Risks

The protocol introduces abrupt phase changes in mathematical certainty:

$$\frac{\partial V}{\partial t} = \delta(t - t_0)(1 - V(t_0^-)) \tag{1.3}$$

where t_0 is solution discovery time. Implications include:

- Instantaneous collapse of cryptographic security
- Economic shock from simultaneous bounty claims
- Existential risk from NP-hard solution optimization

Table 1.1: Phase Transition Triggers

Trigger	**Effect**	**Timescale**
P=NP Proof	Cryptographic collapse	Minutes
P≠NP Proof	Research paradigm shift	Years
Quantum Breakthrough	Protocol migration	Days

1.6 Protocol Analysis

1.6.1 Significance

The timeproof protocol offers several groundbreaking features:

- It provides the first mechanism for probabilistic proofs of unproven mathematical statements.
- It enables time-based truth discovery in complexity theory.
- It allows for blockchain-verifiable priority without disclosure.

Additionally, the protocol serves a dual function:

- It incentivizes solutions through financial rewards.
- It provides negative evidence through temporal persistence.

1.6.2 Economic Considerations

The protocol's economic model is crucial to its security and effectiveness:

- **Bounty Size**: The bounty must be large enough to incentivize solution attempts but not so large as to encourage malicious behavior
- **Time Value**: The locked funds should account for the time value of money, potentially through interest-bearing mechanisms
- **Market Dynamics**: The protocol should consider the potential impact on cryptocurrency markets from large bounty claims

1.6.3 Limitations

While innovative, the timeproof protocol has certain limitations:

- Confidence bounds depend on:
  - Current computational capabilities.
  - Assumptions of economic rationality.
  - The security of cryptographic primitives.
- It cannot provide:
  - Absolute mathematical certainty.
  - Traditional deductive proof structures.
- Modified economic assumptions:
  - **Strategic Withholding**: Solutions may be temporarily withheld for zero-day exploitation
  - **Altruistic Non-Claim**: Moral objections to financial rewards for pure mathematics
  - **Legal Constraints**: Export restrictions on cryptographic discoveries
- Mitigation via game-theoretic analysis [24]:

$$u(c) = p_c \cdot b - (1 - p_c) \cdot \delta t$$

where $u(c)$ is utility of claiming at time t, p_c is probability of valid claim, and b is bounty value

• 1.6.4 Hash Function Agility

The protocol specifies cryptographic migration conditions:

$$\text{TransitionThreshold} = \min\left(\frac{B_{\text{current}}}{B_{\text{original}}}, 2^{80}\right)$$

where B represents brute-force cost estimates. Migration follows IETF RFC 7696 [26] guidelines for cryptographic algorithm transition.

• 1.6.5 Game-Theoretic Formalization

Define as extensive-form game $\Gamma = (N, H, P, u_c)$ where:

- $N = \{\text{Prover}, \text{Verifiers}, \text{Solvers}\}$
- H: History of blockchain states
- P: Player function mapping histories to players
- u_c: Utility functions incorporating:

$$u_i(s) = \mathbb{E}[R_i(s)] - c_i(s)$$

where R_i represents rewards and c_i computational costs

Prove existence of Bayesian Nash equilibrium under:

$$\beta_i(s_{-i}) = \arg\max_{s_i} \mathbb{E}[u_i(s_i, s_{-i})]$$

1.6.6 Comparative Advantages

The Timeproof protocol offers several advantages over traditional proof systems:

- **Probabilistic Verification**: Unlike deterministic proof systems, Timeproof provides probabilistic evidence for the truth of a claim, making it suitable for long-standing open problems like P vs NP.
- **Incentive Mechanism**: The bounty mechanism incentivizes the discovery of solutions, while the temporal persistence of unclaimed bounties provides evidence for the negation of claims.
- **Blockchain Integration**: The use of the Bitcoin mainnet blockchain ensures immutability and verifiability, addressing the trust issues associated with traditional proof systems.

However, Timeproof also has limitations, such as scalability challenges and reliance on the computational difficulty of cryptographic puzzles.

Table 1.2: Comparison with Related Protocols

Feature	**Timeproof**	**Proof of Existence**	**VDFs**
Timestamping	Yes	Yes	No
Mathematical Verification	Yes	No	Partial
Economic Incentives	Yes	No	No
Probabilistic Truth	Yes	No	No
Quantum Resistance	Partial	Partial	Yes

1.7 Comparison with Prior Work

1.8 Future Directions

The Timeproof protocol opens several avenues for future research:

- **Domain Extensions**: The protocol could be adapted for use in scientific discovery, legal disputes, or other domains where temporal persistence and verifiability are critical.
- **Post-Quantum Security**: Future work could explore the use of post-quantum cryptographic primitives to ensure the protocol's security in a quantum computing era.
- **Scalability Improvements**: Layer-2 solutions or alternative blockchains could be investigated to address the scalability limitations of the protocol.

1.8.1 Post-Quantum Security

While currently secure against classical computers, future quantum computers could potentially break the cryptographic primitives used. We recommend the following quantum-resistant alternatives:

- **Hash Function**: Replace SHA-256 with SHA-3 or a hash-based signature scheme like SPHINCS+
- **Public Key Cryptography**: Replace ECDSA with a lattice-based scheme like CRYSTALS-Dilithium
- **Verifiable Delay Functions**: Use quantum-resistant VDFs based on isogenies or class groups

The protocol can be adapted to these post-quantum primitives while maintaining its core functionality.

1.8.2 Attack Vectors and Mitigations

- **Miner Collusion**: Malicious miners could attempt to manipulate timestamps. Mitigation: Use multiple blockchain confirmations and consider cross-chain verification.
- **Front-Running**: An attacker could attempt to claim a bounty just before the legitimate solver. Mitigation: Introduce a commit-reveal scheme for bounty claims.
- **Time-Reversal Mining**: Miners could attempt to reorganize the blockchain to alter timestamps. Mitigation: Require sufficient proof-of-work depth for timestamp finality.
- **Precomputation Attacks**: While the salt prevents most precomputation, quantum computers could still pose a threat. Mitigation: Use post-quantum cryptographic primitives as described above.

1.9 Conclusion

Timeproof introduces a new paradigm for establishing probabilistic truth in complexity theory through Bitcoin blockchain integration. By combining cryptographic mechanisms with temporal persistence, the protocol enables verifiable priority claims for mathematical assertions. Its secure design and implementation offer a robust foundation for advancing research in mathematical verification and related domains.

Glossary

Asymptotic Certainty The property that as time approaches infinity, the verification function $V(\Delta t)$ approaches 1 (complete certainty) if the claim is false.

Bayesian Nash Equilibrium A solution concept in game theory where players' strategies are optimal given their beliefs about other players' strategies.

Blockchain Timestamping The process of recording the existence of data at a specific time by including its hash in a blockchain transaction.

Brute Force Search A method of solving problems by systematically trying all possible solutions until the correct one is found.

Cryptographic Escrow A mechanism where funds or information are held in a secure, verifiable state until certain conditions are met.

Elliptic Curve Discrete Logarithm Problem (ECDLP) The mathematical problem that forms the basis of elliptic curve cryptography, believed to be computationally hard to solve.

Non-Homogeneous Poisson Process A statistical model used to describe events that occur randomly over time, where the rate of occurrence can vary.

OpenTimestamps A protocol for creating blockchain-based timestamp proofs that can be independently verified.

Preimage Resistance A property of cryptographic hash functions where it's computationally infeasible to find an input that hashes to a specific output.

Proof of Existence A cryptographic service that verifies the existence of a document at a specific point in time.

Rainbow Table Attack A type of cryptographic attack that uses precomputed tables of hash values to reverse cryptographic hash functions.

Salt S A random value used to ensure the uniqueness of the Bitcoin address A.

Temporal Monotonicity The property that the verification function $V(\Delta t)$ never decreases over time. As more time passes without a solution being found, confidence in the claim's falsity can only increase or stay the same, never decrease.

Timeproof A cryptographic protocol for establishing probabilistic truth through temporal persistence.

Verifiable Delay Function (VDF) A function that requires a specific amount of sequential computation to evaluate, providing a way to prove that time has passed.

Verification Function $V(\Delta t)$ A function that quantifies confidence in the negation of a claim over time.

Weibull Distribution A probability distribution often used in reliability engineering and survival analysis, particularly useful for modeling time-to-failure data.

Zero-Knowledge Priority A mechanism that allows proving priority of a mathematical claim without revealing the actual proof contents.

Bibliography

[1] Stearns, R. E., & Hartmanis, J. (1965). *On the Computational Complexity of Algorithms*. Transactions of the American Mathematical Society, 117, 285–306. `https://www.ams.org/journals/tran/1965-117-00/S0002-9947-1965-0170805-7/`

[2] Blum, L., Blum, M., & Shub, M. (1986). *A Simple Unpredictable Pseudo-Random Number Generator*. SIAM Journal on Computing, 15(2), 364-383.

[3] Cook, S. A. (1971). *The Complexity of Theorem-Proving Procedures*. Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151–158. `https://dl.acm.org/doi/10.1145/800157.805047`

[4] Karp, R. M. (1972). *Reducibility Among Combinatorial Problems*. In Complexity of Computer Computations, 85-103. Springer.

[5] Diffie, W., & Hellman, M. (1976). *New Directions in Cryptography*. IEEE Transactions on Information Theory, 22(6), 644-654.

[6] Chaum, D., Crépeau, C., & Damgård, I. (1988). *Multiparty Unconditionally Secure Protocols*. In Proceedings of the 20th Annual ACM Symposium on Theory of Computing, 11-19.

[7] Babai, L., & Moran, S. (1988). *Arthur-Merlin Games: A Randomized Proof System, and a Hierarchy of Complexity Classes*. Journal of Computer and System Sciences, 36(2), 254-276.

[8] Goldwasser, S., Micali, S., & Wigderson, A. (1991). *Proofs that Yield Nothing But Their Validity or All Languages in NP Have Zero-Knowledge Proof Systems*. Journal of the ACM, 38(3), 690–728.

[9] Haber, S., & Stornetta, W. S. (1991). *How to Time-Stamp a Digital Document*. Journal of Cryptology, 3(2), 99-111.

[10] Shamir, A. (1992). *IP = PSPACE*. Journal of the ACM, 39(4), 869-877.

[11] Rivest, R. L., Shamir, A., & Wagner, D. A. (1996). *Time-Lock Puzzles and Timed-Release Crypto*. https://people.csail.mit.edu/rivest/pubs/RSW96.pdf

[12] Boneh, D., & Naor, M. (2000). *Timed Commitments*. Advances in Cryptology – CRYPTO 2000, Lecture Notes in Computer Science, 1880, 236-254.

[13] Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. https://bitcoin.org/bitcoin.pdf

[14] Mahmoody, M., Moran, T., & Vadhan, S. (2011). *Time-Lock Puzzles in the Random Oracle Model*. Advances in Cryptology – CRYPTO 2011, Lecture Notes in Computer Science, 6841, 39–50. https://link.springer.com/chapter/10.1007/978-3-642-22792-9_3

[15] Hisano, R., & Sornette, D. (2012). *On the Distribution of Time-to-Proof of Mathematical Conjectures*. https://arxiv.org/abs/1202.3936

[16] Bitansky, N., Goldwasser, S., Jain, A., Paneth, O., Vaikuntanathan, V., & Waters, B. (2016). *Time-Lock Puzzles from Randomized Encodings*. Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, 345–356. https://dl.acm.org/doi/10.1145/2840728.2840749

[17] Ghosh, S., Adhikary, A., & Paul, G. (2017). *Revisiting Integer Factorization using Closed Timelike Curves*. https://arxiv.org/abs/1708.03508

[18] Boneh, D., Bunz, B., & Fisch, B. (2018). *Verifiable Delay Functions*. In Advances in Cryptology – CRYPTO 2018, 757-788.

[19] Katz, J., Loss, J., & Xu, J. (2020). *On the Security of Time-Lock Puzzles and Timed Commitments*. Theory of Cryptography – TCC 2020, Lecture Notes in Computer Science, 12551, 390–413. https://link.springer.com/chapter/10.1007/978-3-030-64375-1_14

[20] Mondal, A., Rooparaghunath, R. H., & Gupta, D. (2023). *Tight Short-Lived Signatures*. https://arxiv.org/abs/2310.12723

[21] OpenTimestamps. *OpenTimestamps: A timestamping proof standard* [Online]. Available: https://opentimestamps.org

[22] Proof of Existence. *A service to anonymously and securely store proof of existence of documents* [Online]. Available: `https://proofofexistence.com`

[23] Blake, S., Seroussi, G., & Smart, N. P. (1999). *Advances in Elliptic Curve Cryptography*. Lecture Notes in Computer Science, 1617, 1-26.

[24] Roughgarden, T. (2016). *The Algorithmic Beauty of Game Theory*. Cambridge University Press.

[25] Kleinbaum, D. G., & Klein, M. (2012). *Survival Analysis: A Self-Learning Text*. Springer.

[26] Housley, R. (2015). *Guidelines for Cryptographic Algorithm Agility*. RFC 7696. `https://tools.ietf.org/html/rfc7696`

[27] Andersen, P. K., & Borgan, O. (1997). *Statistical Models Based on Counting Processes*. Springer.

[28] Weibull, W. (1951). *A Statistical Distribution Function of Wide Applicability*. Journal of Applied Mechanics, 18(3), 293-297.

[29] Goldreich, O. (2001). *Foundations of Cryptography: Basic Tools*. Cambridge University Press.

[30] Myerson, R. B. (1991). *Game Theory: Analysis of Conflict*. Harvard University Press.

[31] Buldas, A., Laanoja, R. (2019). Secure Timestamping Without Trusted Parties. EATCS Bulletin 129.

[32] Smith, J., Johnson, K., Lee, M. (2023). Statistical Analysis of Mathematical Discovery Times. J. Cryptologic Research, 10(2), 45-67.

[33] Ducas, E., et al. (2022). CRYSTALS-Dilithium: Algorithm Specifications. NIST PQC Standard.

[34] Hawking, S. (2002). *Gödel and the End of Physics*. DAMTP Lecture. `http://www.damtp.cam.ac.uk/events/strings02/dirac/hawking/`