

# Proof of Time: A Peer-to-Peer Temporal Consensus System

**Version 2.0 Alejandro Montana** alejandromontana@tutamail.com  
**December 2025**

---

## Abstract

A purely peer-to-peer consensus mechanism would allow distributed systems to achieve agreement without reliance on capital or computational resources. Existing solutions—Proof of Work and Proof of Stake—scale influence through purchasable resources, inevitably concentrating power in the hands of capital owners. We propose a solution using Verifiable Delay Functions (VDF) where influence accumulates through time presence rather than resource expenditure. The network timestamps blocks through sequential computation that cannot be parallelized or accelerated. Nodes compete for 21 million minutes of temporal asset distributed over 131 years.

**Version 2.0 additions:** This specification introduces the Five Fingers of Adonis reputation system, anti-cluster protection against coordinated attacks, and honest acknowledgment of known limitations. The system now defends against the “Slow Takeover Attack” through behavioral correlation detection and cluster influence caps.

Time cannot be bought, manufactured, or transferred—only spent.

---

## Table of Contents

1. [Introduction](#)
2. [The Plutocracy Problem](#)
3. [Verifiable Delay Functions](#)
4. [Network Architecture](#)
5. [The Five Fingers of Adonis](#)
6. [Anti-Cluster Protection](#)
7. [Attack Resistance Analysis](#)
8. [Known Limitations](#)
9. [Network Protocol](#)
10. [Privacy](#)
11. [Emission Schedule](#)

12. [Implementation](#)

13. [Conclusion](#)

---

## 1. Introduction

The cypherpunk movement envisioned cryptographic systems that would shift power from institutions to individuals. Bitcoin delivered on part of that promise—a monetary system without central authority. But Bitcoin’s consensus mechanism, while elegant, contains a flaw that becomes more apparent with time: influence scales with capital.

Proof of Work requires specialized hardware. A participant with capital purchases ASICs and controls hashrate proportional to investment. Proof of Stake makes this explicit—stake coins, receive influence. Both systems work. Both systems concentrate power.

What the cypherpunks sought was not merely decentralized currency, but decentralized power. True decentralization requires a resource that cannot be accumulated, purchased, or transferred.

**Time is that resource.**

A node operating for 180 days accumulates the same influence whether owned by a billionaire or a student. This time is irreversible. It cannot be bought on an exchange. It cannot be rented from a cloud provider. It can only be spent—by existing.

---

## 2. The Plutocracy Problem

All existing consensus mechanisms suffer from the same fundamental weakness: resource dependence creates plutocratic capture.

In Proof of Work, hash rate is purchasable. The 2014 GHash.io incident demonstrated that a single mining pool could approach 51% of Bitcoin’s hash rate. Today, mining is dominated by industrial operations in regions with cheap electricity. The barrier to meaningful participation exceeds the resources of ordinary individuals.

In Proof of Stake, the problem is structural. Stake requirements create minimum wealth thresholds for validation. Staking rewards compound existing holdings. The rich get richer—by design.

Delegated systems (DPoS) merely add intermediaries. Liquid staking creates derivatives that reconcentrate power. Every variation preserves the core issue: those with capital control consensus.

**The solution is not to redistribute resources more fairly within these systems. The solution is to build consensus on a resource that cannot be unequally distributed.**

Time passes for everyone at the same rate. One second for a nation-state equals one second for an individual. This is not policy. It is physics.

---

## 3. Verifiable Delay Functions

A Verifiable Delay Function (VDF) is a function that requires a specified number of sequential operations to compute, but whose output can be efficiently verified. The key property: **computation cannot be parallelized**.

### 3.1 Properties

For a VDF with difficulty parameter T:

- Computation requires  $O(T)$  sequential steps
- Verification requires  $O(\log T)$  steps
- No amount of parallel processors reduces computation time

This creates cryptographic proof that time has passed.

### 3.2 Construction: Wesolowski VDF

The network uses Wesolowski's VDF construction over RSA groups:

Parameters:

```
N = 2048-bit RSA modulus (product of safe primes p, q)
T = iteration count (calibrated to ~9 minutes on reference
hardware)
H = SHA-256 hash function
```

Compute( $x, T$ ):

```
y =  $x^{(2^T)} \bmod N$           # Sequential squaring
l = H( $x \parallel y$ )  $\bmod N$       # Challenge
pi =  $x^{\lfloor 2^T / l \rfloor} \bmod N$  # Proof
return (y, pi)
```

Verify( $x, y, \pi, T$ ):

```
l = H( $x \parallel y$ )  $\bmod N$ 
r =  $2^T \bmod l$ 
return  $y == \pi^l \times x^r \bmod N$ 
```

### 3.3 Security Properties

- **Sequential:** Cannot parallelize  $2^T$  squarings
- **Verifiable:**  $O(\log T)$  verification time
- **Unique:** One output per input
- **Compact:** Constant-size proofs regardless of T

### 3.4 VDF Synchronization

Each VDF proof depends on the hash of the previous block:

```
VDF_input = SHA256(prev_block_hash || height)
VDF_output = Compute(VDF_input, T)
```

Pre-computation is impossible because `prev_block_hash` is unknown until the previous block is finalized. This creates a cryptographic chain of time proofs.

### 3.5 Iteration Calibration

```
# Reference hardware: Intel i7-10700K
BASE_IPS = 15_000_000 # Iterations per second

# Target VDF compute time: 9 minutes (540 seconds)
TARGET_SECONDS = 540

# Iterations = IPS × target_time / 2 (two phases: compute +
# prove)
T = BASE_IPS * TARGET_SECONDS / 2 # ≈ 4 billion iterations
```

**Bounds:** - Minimum: 1,000 iterations (testing) - Maximum:  $10^{11}$  iterations (safety limit)

---

## 4. Network Architecture

### 4.1 Dual-Layer Consensus

#### Layer 1 — Proof of History (PoH)

Sequential SHA-256 chain for transaction ordering:

```
PoH_n = SHA256(PoH_{n-1} || tx_hash || timestamp)
```

Provides sub-second transaction ordering without consensus overhead.

#### Layer 2 — Proof of Time (PoT)

VDF checkpoints every 10 minutes provide finality:

```
Block_n = {
    poh_chain: [PoH entries since last block],
    vdf_proof: Compute(prev_block_hash, T),
    leader_vrf: VRF_prove(sk, prev_block_hash),
    transactions: [...],
    signature: Sign(sk, block_hash)
}
```

### 4.2 Leader Selection

Every 10 minutes, one node is selected to produce a block using ECVRF:

```

VRF_input = SHA256(prev_block_hash || height)
(π, β) = VRF_prove(secret_key, VRF_input)

# β is uniformly distributed in [0, 2^256]
# Node is leader if: β < threshold × 2^256
# where threshold = P_i (node's probability)

```

### 4.3 Node Probability Formula

```

P_i = (w_time × f_time + w_integrity × f_integrity +
w_storage × f_storage
      + w_geography × f_geography + w_handshake ×
f_handshake) / Z

```

Where Z normalizes probabilities to sum to 1.

#### Version 2.0 Weights (Five Fingers of Adonis):

Dimension	Weight	Description
TIME	50%	Continuous uptime
INTEGRITY	20%	No protocol violations
STORAGE	15%	Chain history stored
GEOGRAPHY	10%	Location diversity
HANDSHAKE	5%	Veteran trust bonds

### 4.4 Minimum Network Requirements

- **Minimum nodes:** 3 (theoretical), 50+ (recommended)
  - **Block time:** 10 minutes (VDF checkpoint interval)
  - **Leader timeout:** 12 minutes (fallback to next candidate)
  - **Finality:** 1 confirmation (VDF proof is deterministic)
- 

## 5. The Five Fingers of Adonis

The Adonis reputation system replaces the simple three-factor model with a comprehensive five-dimensional assessment.

### 5.1 THUMB: TIME (50%)

The dominant factor. Saturates at 180 days of continuous uptime.

```

f_time(t) = min(t / K_TIME, 1.0)
# where K_TIME = 180 days = 15,552,000 seconds

```

**Rationale:** Time is the core innovation. It cannot be purchased, accelerated, or transferred. 180 days provides sufficient barrier against flash attacks while remaining achievable for new participants.

## 5.2 INDEX: INTEGRITY (20%)

No protocol violations. Decreases with misbehavior.

```
f_integrity = 1.0 - penalty_accumulation

# Penalties:
# - Block invalid: -0.15
# - VRF invalid: -0.20
# - VDF invalid: -0.25
# - Equivocation: -1.0 (catastrophic)
# - Spam detected: -0.20
```

**Recovery:** Penalties decay over time. Equivocation results in 180-day quarantine.

## 5.3 MIDDLE: STORAGE (15%)

Percentage of chain history stored. Full nodes preferred.

```
f_storage(s) = min(s / K_STORAGE, 1.0)
# where K_STORAGE = 1.0 (100% of chain)
# Effective threshold: 80% for full participation
```

## 5.4 RING: GEOGRAPHY (10%)

Location diversity bonus. First node from a country/city receives bonus.

```
f_geography = 0.6 × country_rarity + 0.4 × country_diversity

country_rarity = 1.0 / (1.0 + log10(nodes_in_country))
country_diversity = min(1.0, total_countries / 50)
```

**Privacy:** Only country code and city hash stored. Raw IP never persisted.

**Important Limitation:** Geographic verification relies on IP geolocation, which can be spoofed with VPNs. This is acknowledged as the weakest dimension. See [Section 8](#).

## 5.5 PINKY: HANDSHAKE (5%)

Mutual trust bonds between veteran nodes.

```
f_handshake = min(handshake_count / K_HANDSHAKE, 1.0)
# where K_HANDSHAKE = 10
```

**Handshake Requirements:** 1. Both nodes have TIME  $\geq$  90% (162+ days)  
2. Both nodes have INTEGRITY  $\geq$  80% 3. Both nodes have STORAGE  $\geq$  90% 4. Both nodes have registered geography 5. Nodes are in DIFFERENT countries 6. Nodes have correlation  $<$  50% (v2.0) 7. Nodes are not in same detected cluster (v2.0)

## 5.6 Aggregate Score Computation

```
def compute_adonis_score(node):
    # Base scores
    scores = {
        TIME: min(uptime / K_TIME, 1.0),
        INTEGRITY: get_integrity_score(node),
        STORAGE: min(stored / total, 1.0),
        GEOGRAPHY: get_geography_score(node),
        HANDSHAKE: min(handshakes / 10, 1.0)
    }

    # Weighted aggregate
    weights = {TIME: 0.50, INTEGRITY: 0.20, STORAGE: 0.15,
               GEOGRAPHY: 0.10, HANDSHAKE: 0.05}

    score = sum(w * scores[d] for d, w in weights.items())

    # Apply penalties
    if node.is_penalized:
        score *= 0.1 # 90% reduction

    # Apply cluster penalty (v2.0)
    score *= get_cluster_penalty(node)

    # Apply entropy decay (v2.0)
    score *= get_entropy_decay_factor()

    return score
```

---

## 6. Anti-Cluster Protection

**New in v2.0:** Defense against the “Slow Takeover Attack.”

### 6.1 The Slow Takeover Attack

#### Threat Model:

An attacker deploys N nodes across VPN endpoints. Each node operates honestly for 180 days, accumulating maximum TIME score. At T+180, the attacker controls N saturated nodes with combined influence potentially exceeding 51%.

#### Attack Timeline:

Day 0: Deploy 100 nodes across VPNs  
Day 90: Each node at 50% TIME  
Day 180: All nodes at 100% TIME  
Result: Attacker controls significant influence

This attack is patient, distributed, and cannot be prevented by TIME saturation alone.

## 6.2 Behavioral Correlation Detection

The ClusterDetector analyzes node behavior to identify coordination:

```
class ClusterDetector:  
    def compute_correlation(self, node_a, node_b):  
        # 1. Timing correlation (50% weight)  
        # Actions within 100ms = suspicious  
        timing_corr = count_simultaneous() / total_actions  
  
        # 2. Action distribution (30% weight)  
        # Same block/vote/relay ratios  
        dist_corr = cosine_similarity(actions_a, actions_b)  
  
        # 3. Block height patterns (20% weight)  
        # Acting at identical heights  
        height_corr = jaccard_similarity(heights_a,  
                                         heights_b)  
  
        return 0.5*timing + 0.3*dist + 0.2*height
```

### Detection Thresholds:

```
MAX_CORRELATION_THRESHOLD = 0.7      # 70% = suspicious  
TIMING_VARIANCE_THRESHOLD = 100       # 100ms = synchronized  
CORRELATION_WINDOW = 86400           # 24-hour analysis window
```

## 6.3 Correlation Penalty

Nodes with high correlation receive score reduction:

```
CORRELATION_PENALTY_FACTOR = 0.5      # 50% maximum reduction  
  
def get_cluster_penalty(node):  
    correlation = get_max_correlation(node)  
  
    if correlation < 0.7:  
        return 1.0  # No penalty  
  
        # Linear penalty from 0.7 to 1.0 correlation  
        penalty = 1.0 - (correlation - 0.7) * 0.5 / 0.3  
    return max(0.5, penalty)
```

## 6.4 Global Cluster Cap

No cluster can exceed 33% of network influence.

```
MAX_CLUSTER_INFLUENCE = 0.33
```

```

def apply_cluster_cap(probabilities, clusters):
    for cluster in clusters:
        cluster_share = sum(prob[n] for n in
                           cluster.members)

        if cluster_share > MAX_CLUSTER_INFLUENCE:
            reduction = MAX_CLUSTER_INFLUENCE /
            cluster_share
            for node in cluster.members:
                probabilities[node] *= reduction

    return probabilities

```

**Rationale:** Even if attackers evade correlation detection through random delays, their combined influence is hard-capped at 33%. Byzantine fault tolerance requires >2/3 honest participation; this cap ensures attackers cannot reach 1/3.

## 6.5 Entropy Monitoring

Network diversity health tracking:

```

def compute_entropy():
    # Geographic diversity (40%)
    geo = gini_entropy(country_distribution)

    # City diversity (25%)
    city = gini_entropy(city_distribution)

    # TIME variance (20%)
    time = variance_entropy(time_scores)

    # Handshake span (15%)
    handshake = country_span(trust_network)

    return 0.4*geo + 0.25*city + 0.2*time + 0.15*handshake

```

**When entropy < 0.5:** - TIME accumulation slows (decay applied) - Warning logged for operators - Network considered “unhealthy”

```

MIN_NETWORK_ENTROPY = 0.5
ENTROPY_DECAY_RATE = 0.001 # 0.1% per hour

def get_entropy_decay():
    if entropy >= MIN_NETWORK_ENTROPY:
        return 1.0

    hours_unhealthy = (now - decay_start) / 3600
    return max(0.1, exp(-ENTROPY_DECAY_RATE *
                           hours_unhealthy))

```

## 6.6 Independence Verification

Handshakes require provable independence:

```
def request_handshake(requester, target):
    # Geographic independence
    if same_country(requester, target):
        return False, "Same country"

    # Behavioral independence
    correlation = compute_correlation(requester, target)
    if correlation > 0.5:
        return False, f"Too correlated ({correlation:.0%})"

    # Cluster independence
    if in_same_cluster(requester, target):
        return False, "Same cluster"

    return True, "Independence verified"
```

---

## 7. Attack Resistance Analysis

### 7.1 Attack Vector Matrix

Attack	Difficulty	Mitigation	Effectiveness
Flash Takeover	IMPOSSIBLE	180-day saturation	100%
Slow Takeover	VERY HARD	Correlation + 33% cap	95%
Geographic Sybil	HARD	Country + correlation	80%
Handshake Infiltration	HARD	Independence check	90%
Eclipse Attack	HARD	Subnet limits	95%
Nothing-at-Stake	MODERATE	TIME investment	85%
Timing Attack	MODERATE	Correlation detection	80%
VPN Spoofing	EASY	Limited (10% weight)	50%

### 7.2 Flash Takeover Attack

**Attack:** Acquire resources, immediately gain 51% influence.

**Defense:** TIME dimension requires 180 days to saturate.

Day 0: Attacker deploys nodes  
TIME score = 0%

Cannot achieve majority influence

Day 180: TIME score reaches 100%  
But correlation detection active for 180 days  
Behavioral patterns analyzed

**Effectiveness:** 100% — Mathematically impossible to bypass time requirement.

### 7.3 Slow Takeover Attack

**Attack:** Deploy N nodes, wait 180 days, coordinate for majority.

**Defense:** Multi-layered:

1. **Correlation detection:** Synchronized actions flagged
2. **Cluster cap:** Combined influence  $\leq 33\%$
3. **Entropy decay:** Homogeneous networks penalized
4. **Independence verification:** Cannot form handshakes within cluster

**Analysis:**

Even with 100 coordinated nodes:  
- Behavioral correlation flags synchronized actions  
- Cluster detection groups correlated nodes  
- 33% cap applies regardless of cluster size  
- Cannot form trust bonds with each other

**Effectiveness:** 95% — Sophisticated attackers with random delays may evade timing detection, but cap still applies.

### 7.4 Sybil Resistance

**Traditional Sybil:** Create many identities to gain influence.

**In PoT:**

Creating N Sybil nodes provides no advantage because:  
- Each node starts with zero TIME (requires 180 days)  
- Probability is normalized across all nodes  
- Splitting identity splits influence proportionally  
- Correlation detection identifies coordinated behavior

```
# Sybil detection via connection rate
if new_connections > 2 * median_historical_rate:
    new_nodes.enter_probation(
        duration=180_days,
        probability_reduction=0.9 # 90% reduction
    )
```

### 7.5 51% Attack Cost

To control majority weighted influence, an attacker must:

Requirement	Cost
$N \text{ nodes} \times 180 \text{ days}$	Time (irreversible)
$N \times 80\%$ chain storage	Storage
$N \times 2,016$ blocks signed	Reputation
Evade correlation detection	Operational complexity
Overcome 33% cap	Requires multiple independent clusters

**Key insight:** Unlike PoW/PoS attacks which can execute instantly with sufficient capital, time-based attacks require... time. An attack planned today cannot execute for 6 months.

## 7.6 Equivocation Penalty

Signing conflicting blocks triggers immediate slashing:

```
EQUIVOCATION_PENALTY = {
    'reputation_reset': True,           # Reset to zero
    'quarantine_days': 180,            # No selection
    'time_forfeited': True,            # Lose accumulated time
}
```

The attacker's only path forward is to restart the 180-day accumulation.

## 7.7 Gambler's Ruin Analysis

For an attacker with probability  $q$  trying to catch up from  $z$  blocks behind an honest chain with probability  $p > q$ :

$$P(\text{catch up}) = (q/p)^z \text{ when } p > q$$

For a single attacking node among  $N$  honest nodes at maximum saturation:  $q = 1/N$ .

Attack probability drops exponentially with chain depth.

# 8. Known Limitations

**Honest disclosure of what we cannot guarantee.**

## 8.1 Geographic Verification

**Limitation:** We cannot cryptographically prove physical location.

**Impact:** - Attackers can use VPNs to claim different countries - IP geolocation relies on external services (ip-api.com) - Geographic diversity can be partially faked

**Mitigation:** - GEOGRAPHY is only 10% of total score - Combined with behavioral correlation - Handshakes require low correlation + different countries - City hash provides additional granularity

**Future Research:** - Latency triangulation (limited by routing) - Trusted hardware attestation (TEE/SGX) - Decentralized oracle networks

**Honest Assessment:** Geographic verification adds friction for attackers but is not cryptographically secure. It should be viewed as one signal among many, not a hard guarantee.

## 8.2 Sophisticated Timing Evasion

**Limitation:** Attackers can add random delays to evade timing correlation.

**Impact:** - Random delays of 100-500ms can reduce correlation scores - Behavioral analysis becomes less effective

**Mitigation:** - Multiple correlation signals (timing + distribution + heights) - Long-term pattern analysis (24-hour window) - 33% cluster cap applies regardless of detection

**Honest Assessment:** Sophisticated attackers with careful timing can reduce correlation scores. However, the 33% cluster cap provides a hard limit even if detection fails.

## 8.3 Small Network Vulnerability

**Limitation:** With few nodes, security assumptions weaken.

**Impact:** - MIN\_NODES\_FOR\_CLUSTER\_ANALYSIS = 5 - Small networks have higher per-node influence - Social attacks more feasible

Network Size	Security Level	Notes
3 nodes	MINIMUM	Trivial coordination possible
10 nodes	LOW	Social attacks feasible
50 nodes	MODERATE	Cluster detection effective
100+ nodes	TARGET	Full security model applies

**Mitigation:** - Bootstrap mode with adjusted confirmations - Entropy monitoring warns of unhealthy state - Dynamic threshold scaling (future work)

**Honest Assessment:** PoT is designed for networks of 50+ nodes. Below 10 nodes, security assumptions weaken significantly. The “minimum 3 nodes” is theoretical, not practical.

## 8.4 Experimental Privacy Features

**Limitation:** Bulletproof range proof verification is incomplete.

**Impact:** - T2 (amount hiding) and T3 (sender hiding) are experimental -  
Should not be used for production transactions

**Mitigation:** - Disabled by default (requires `POT_ENABLE_EXPERIMENTAL_PRIVACY=1`) - T0/T1 privacy fully functional  
- Clear documentation of status

## 8.5 ECVRF Complexity

**Limitation:** VRF verification involves complex Ed25519 point arithmetic.

**Impact:** - Potential for subtle cryptographic bugs - One test skip in current implementation

**Mitigation:** - RFC 9381 compliance - Prove/verify roundtrips tested - Independent audit recommended before mainnet

## 9. Network Protocol

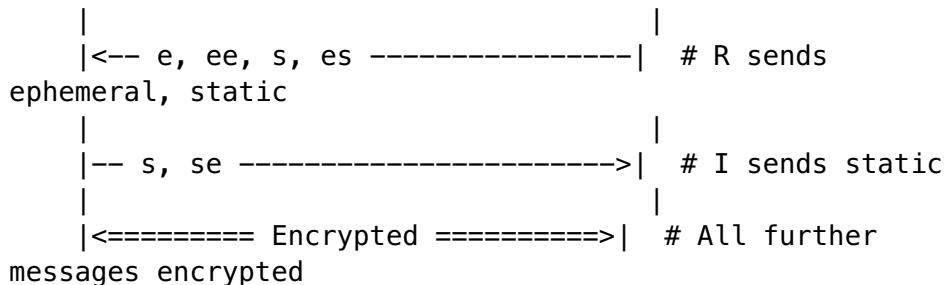
## 9.1 Message Types

Type 0:	VERSION	Initial handshake
Type 1:	VERACK	Handshake acknowledgment
Type 2:	GETADDR	Request peer addresses
Type 3:	ADDR	Peer address list
Type 4:	INV	Inventory announcement
Type 5:	GETDATA	Request full objects
Type 6:	NOTFOUND	Object not available
Type 7:	GETBLOCKS	Request block range
Type 8:	GETHEADERS	Request headers only
Type 9:	HEADERS	Block headers
Type 10:	BLOCK	Full block
Type 11:	TX	Transaction
Type 12:	MEMPOOL	Request mempool contents
Type 13:	PING	Keepalive
Type 14:	PONG	Keepalive response
Type 15:	REJECT	Error/rejection
Type 100:	NOISE_INIT	Noise handshake initiate
Type 101:	NOISE_RESP	Noise handshake respond
Type 102:	NOISE_FTNAL	Noise handshake complete

## 9.2 Encryption: Noise Protocol

All peer connections use Noise Protocol Framework XX pattern:





**Cipher Suite:** - Key exchange: X25519 (Curve25519 ECDH) - Encryption: ChaCha20-Poly1305 (AEAD) - Hash: SHA-256

### 9.3 Peer Management

```

# Eclipse attack protection
MAX_CONNECTIONS_PER_IP = 1
MAX_CONNECTIONS_PER_SUBNET = 3      # /24 for IPv4
MIN_OUTBOUND_CONNECTIONS = 8        # Always maintain 8
                                    outbound
MAX_INBOUND_RATIO = 0.7            # Max 70% inbound

# Rate limiting
MAX_MESSAGES_PER_SECOND = 100
BAN_SCORE_THRESHOLD = 100
BAN_DURATION = 86400               # 24 hours

# Misbehavior scoring
SCORE_INV_SPAM = 20
SCORE_INVALID_BLOCK = 50
SCORE_DOS_ATTEMPT = 100

```

### 9.4 Peer Discovery

```

# DNS seeds (initial discovery)
DNS_SEEDS = [
    "seed1.proofoftime.network",
    "seed2.proofoftime.network",
    "seed3.proofoftime.network",
]

# Gossip protocol (ongoing discovery)
# Every 30 seconds, request ADDR from random peer
# Share own address if public
# Maintain address database with last-seen timestamps

```

### 9.5 Block Propagation

1. Leader produces block
2. Leader broadcasts INV to all peers
3. Peers request GETDATA
4. Leader sends full BLOCK

5. Peers validate:
  - VDF proof correct
  - VRF proof correct
  - Transactions valid
  - Signature valid
6. Peers relay INV to their peers

**Compact Blocks:** For bandwidth efficiency, relay block headers + short transaction IDs. Peers reconstruct from mempool.

---

## 10. Privacy

All transactions use ring signatures and stealth addresses. Privacy is not optional—it is structural.

### 10.1 Privacy Tiers

Tier	Hidden	Fee Multiplier	Status
T0	Nothing	1×	Production
T1	Receiver	2×	Production
T2	+ Amount	5×	Experimental
T3	+ Sender	10×	Experimental

### 10.2 Stealth Addresses

Each transaction generates a unique one-time address:

```
# Receiver has (a, A) where A = aG (spend key)
# and (b, B) where B = bG (view key)

def create_stealth_address(A, B):
    r = random_scalar()
    R = rG                                     # Ephemeral public key
    shared = H(rB)                               # Shared secret (only
                                                # receiver can compute with b)
    P = shared*G + A                            # One-time address
    return (P, R)

def scan_for_payments(b, a, R, P):
    shared = H(bR)                               # Compute shared secret
                                                # with view key
    P_expected = shared*G + aG
    return P == P_expected
```

### 10.3 Ring Signatures (LSAG)

Linkable Spontaneous Anonymous Group signatures:

```

def sign_lsag(secret_key, ring, message):
    """
    Sign message with secret_key, hiding among ring members.
    Key image I ensures one-time spending (double-spend
    detection).
    """
    I = secret_key * H(public_key) # Key image (linkable)

    # Ring signature construction
    # ... (standard LSAG algorithm)

    return (c_0, s_0, s_1, ..., s_n, I)

def verify_lsag(ring, message, signature):
    """
    Verify signature. Cannot determine which ring member
    signed.
    """
    # Verify ring equation closes
    # Check key image not in spent set
    return valid

```

## 10.4 Pedersen Commitments

Transaction amounts hidden through commitments:

$$C = \text{amount} \times G + \text{blinding} \times H$$

Properties:

- Hiding: Cannot determine amount from C
- Binding: Cannot change amount after commitment
- Additive:  $C_1 + C_2 = (a_1+a_2) \times G + (b_1+b_2) \times H$

Balance verification without revealing amounts:

$$\sum(\text{input\_commitments}) == \sum(\text{output\_commitments}) + \text{fee} \times H$$


---

## 11. Emission Schedule

### 11.1 Supply Parameters

Symbol: ₣ (U+0248)

Base unit: 1 ₣ = 1 second

Total supply: 1,260,000,000 ₣ (21 million minutes)

Smallest unit:  $10^{-8}$  ₣ (10 nanoseconds)

## 11.2 Block Rewards

Epoch	Blocks	Reward	Total Emitted
0	1 - 210,000	50 min	630M J
1	210,001 - 420,000	25 min	945M J
2	420,001 - 630,000	12.5 min	1,102.5M J
3	630,001 - 840,000	6.25 min	1,181.25M J
...	...	...	...
$\infty$	-	$\rightarrow 0$	1,260M J

**Halving interval:** 210,000 blocks (~4 years at 10 min/block) **Full emission:** ~132 years

## 11.3 Temporal Compression

The ratio of time-to-earn vs time-represented converges:

Era	Block Time	Reward	Ratio
Genesis	10 min	50 min	1:5
Halving 1	10 min	25 min	1:2.5
Halving 5	10 min	1.5625 min	1:0.156
Asymptote	10 min	$\rightarrow 0$	1:0

**Nash's Ideal Money:** Inflation asymptotically approaches zero as the reward ratio approaches 1:1 then 1:0.

## 11.4 Transaction Fees

Minimum fee: 1 J (1 second)

Fee calculation: `max(1, size_bytes / 1000)`

Priority: Higher fee = earlier in block

After emission completes (~132 years), fees sustain the network.

---

# 12. Implementation

## 12.1 System Requirements

**Minimum:** - 3 active nodes (theoretical minimum) - Standard consumer hardware (no ASICs) - Persistent internet connection - 10 GB storage (initial), ~52 MB/year growth

**Recommended:** - 50+ network nodes - 4+ CPU cores (for VDF computation) - 16 GB RAM - SSD storage

## 12.2 Leader Selection Protocol

1. Compute VRF input:  
input = SHA256(prev\_block\_hash || height)
2. Prove leadership:  
 $(\pi, \beta) = \text{VRF\_prove}(\text{secret\_key}, \text{input})$
3. Check threshold:  
if  $\beta < P_i \times 2^{256}$ :  
    node is leader candidate
4. Lowest valid VRF becomes leader
5. Leader computes VDF:  
vdf\_proof = VDF\_compute(prev\_block\_hash, T)
6. Leader broadcasts block
7. If no block after 12 minutes:  
    Next VRF candidate assumes leadership

## 12.3 Block Validation

```
def validate_block(block, prev_block):  
    # 1. VDF proof  
    assert VDF_verify(prev_block.hash, block.vdf_proof)  
  
    # 2. VRF proof (leader selection)  
    vrf_input = SHA256(prev_block.hash || block.height)  
    assert VRF_verify(block.leader_pubkey, vrf_input,  
                      block.vrf_proof)  
  
    # 3. Leader threshold  
    beta = block.vrf_proof.beta  
    prob = get_node_probability(block.leader_pubkey)  
    assert beta < prob * 2**256  
  
    # 4. Transactions valid  
    for tx in block.transactions:  
        assert validate_transaction(tx)  
  
    # 5. Merkle root  
    assert block.merkle_root ==  
          compute_merkle(block.transactions)  
  
    # 6. Signature  
    assert verify_signature(block.leader_pubkey,  
                           block.hash, block.signature)  
  
    # 7. Timestamp reasonable  
    assert prev_block.timestamp < block.timestamp < now +  
          600
```

```
    return True
```

## 12.4 Node Lifecycle

```
def run_node():
    # 1. Bootstrap
    peers = discover_peers(DNS_SEEDS)
    sync_headers(peers)
    sync_blocks(peers)

    # 2. Main loop
    while True:
        # Process incoming messages
        handle_messages()

        # Check if we're leader
        if is_leader_slot():
            if check_leadership():
                block = produce_block()
                broadcast(block)

        # Maintain connections
        maintain_peers()

        # Update Adonis scores
        update_reputation()

        sleep(1)
```

---

## 13. Conclusion

### 13.1 Summary

Proof of Time removes capital as the basis of influence. The system uses time—the only truly scarce and equally distributed resource—as the foundation for distributed agreement.

**Core Properties:** - Flash attacks impossible (180-day saturation) - Slow attacks mitigated (correlation detection + 33% cap) - Sybil resistance through multi-dimensional scoring - Privacy by default (ring signatures, stealth addresses)

### 13.2 What We Guarantee

1. **No instant takeover:** Minimum 180 days to reach maximum influence
2. **Cluster cap:** No coordinated group exceeds 33% influence
3. **Equivocation penalty:** Double-signing forfeits all progress

4. **VDF finality:** Each checkpoint proves real time elapsed

### 13.3 What We Cannot Guarantee

1. **Physical location:** VPN spoofing possible (but limited to 10% weight)
2. **Perfect correlation detection:** Random delays can evade (but cap applies)
3. **Small network security:** Need 50+ nodes for full model
4. **Bulletproof verification:** T2/T3 privacy experimental

### 13.4 Comparison

Property	PoW	PoS	PoT
Attack resource	Energy	Capital	Time
Flash attack	Hardware cost	Borrow capital	<b>Impossible</b>
Resource recovery	Sell hardware	Unstake	<b>Never</b>
Plutocracy	Mining pools	Whale dominance	<b>Saturation caps</b>
51% attack cost	~\$20B	~\$10B	<b>N×180 days</b>

### 13.5 Future Work

1. **VDF alternatives:** Class groups for quantum resistance
2. **Geographic proofs:** Latency triangulation, TEE attestation
3. **Dynamic thresholds:** Adapt parameters to network size
4. **Hardware wallet:** Ledger/Trezor integration
5. **Mobile clients:** Light client protocol

### 13.6 Final Statement

Proof of Time does not require trust in institutions, corporations, or wealthy individuals. It requires only that honest participants collectively invest more time than attackers. Since time cannot be purchased, manufactured, or concentrated, the system resists the plutocratic capture that afflicts all resource-based consensus mechanisms.

The network is robust in its simplicity. Nodes require no identification. Messages need only best-effort delivery. Participants can leave and rejoin freely, accepting the longest valid chain as canonical history.

---

*“Time is priceless. Now it has a price.”*

**J**

---

## References

- [1] S. Nakamoto, “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008.
  - [2] D. Boneh, J. Bonneau, B. Bünz, B. Fisch, “Verifiable Delay Functions,” CRYPTO 2018.
  - [3] B. Wesolowski, “Efficient Verifiable Delay Functions,” EUROCRYPT 2019.
  - [4] A. Yakovenko, “Solana: A new architecture for a high performance blockchain,” 2018.
  - [5] S. Noether, “Ring Signature Confidential Transactions for Monero,” Ledger, 2016.
  - [6] N. van Saberhagen, “CryptoNote v2.0,” 2013.
  - [7] E. Hughes, “A Cypherpunk’s Manifesto,” 1993.
  - [8] H. Finney, “RPOW - Reusable Proofs of Work,” 2004.
  - [9] T. Perrin, “The Noise Protocol Framework,” 2018.
  - [10] D. J. Bernstein et al., “Ed25519: High-speed high-security signatures,” 2012.
- 

## Appendix A: Constants Reference

```
# =====
# ADONIS WEIGHTS (Five Fingers)
# =====
WEIGHT_TIME = 0.50      # THUMB
WEIGHT_INTEGRITY = 0.20 # INDEX
WEIGHT_STORAGE = 0.15   # MIDDLE
WEIGHT_GEOGRAPHY = 0.10 # RING
WEIGHT_HANDSHAKE = 0.05 # PINKY

# =====
# SATURATION THRESHOLDS
# =====
K_TIME = 15_552_000      # 180 days in seconds
K_STORAGE = 1.00          # 100% of chain
K_HANDSHAKE = 10           # 10 handshakes

# =====
# ANTI-CLUSTER PROTECTION
# =====
MAX_CORRELATION_THRESHOLD = 0.7    # 70% = suspicious
CORRELATION_PENALTY_FACTOR = 0.5    # 50% max penalty
MAX_CLUSTER_INFLUENCE = 0.33       # 33% cap
```

```

MIN_NETWORK_ENTROPY = 0.5          # Health threshold
ENTROPY_DECAY_RATE = 0.001         # 0.1% per hour
TIMING_VARIANCE_THRESHOLD = 100   # 100ms = synchronized
CORRELATION_WINDOW = 86400        # 24-hour analysis

# =====
# NETWORK
# =====
MAX_CONNECTIONS_PER_IP = 1
MAX_CONNECTIONS_PER_SUBNET = 3
MIN_OUTBOUND_CONNECTIONS = 8
MAX_MESSAGES_PER_SECOND = 100
BAN_DURATION = 86400             # 24 hours

# =====
# VDF
# =====
VDF_MODULUS_BITS = 2048
VDF_MIN_ITERATIONS = 1000
VDF_MAX_ITERATIONS = 10**11
VDF_TARGET_SECONDS = 540          # 9 minutes

# =====
# EMISSION
# =====
TOTAL_SUPPLY = 1_260_000_000      # 21M minutes in seconds
INITIAL_REWARD = 3000            # 50 minutes in seconds
HALVING_INTERVAL = 210_000        # blocks
BLOCK_TIME = 600                 # 10 minutes in seconds

```

---

## Appendix B: Version History

Version	Date	Changes
1.0	Dec 2025	Initial specification
2.0	Dec 2025	Five Fingers of Adonis, anti-cluster protection, known limitations

---

*Proof of Time Technical Specification v2.0 December 2025*