É^ Montana: Time-Proven Human Temporal Currency

Version 4.3 Alejandro Montana alejandromontana@tutamail.com **December 2025**

Abstract

A peer-to-peer quantum-resistant electronic cash system without reliance on financial institutions. Existing cryptocurrency solutionsâ€”Proof of Work and Proof of Stakeâ€”scale influence through purchasable resources, concentrating power in capital owners.

É^ Montana ($MONT) builds consensus on Proof of Time. Influence accumulates through time presence, not resource expenditure. The network timestamps blocks through sequential computation that cannot be parallelized or accelerated.

Core innovations: - **Proof of Time** â€” VDF-based consensus where time cannot be bought or accelerated - **11 Pantheon Gods** â€” Modular architecture with clear separation of concerns - **12 Apostles** â€” Trust network with collective accountability - **Hal Humanity System** â€” Sybil resistance through graduated trust and time-locked proofs - **Bitcoin Anchoring** â€” TIME dimension tied to 210,000 BTC blocks per epoch - **Post-Quantum Cryptography** â€” SPHINCS+, SHA3-256, SHAKE256 VDF

Time cannot be bought, manufactured, or transferredâ€”only spent. Humanity cannot be faked across Bitcoin halvingsâ€”only proven.

Table of Contents

---

1. Introduction

The cypherpunk movement envisioned cryptographic systems that shift power from institutions to individuals. Bitcoin delivered a monetary system without central authority. But Bitcoinâ€™s consensus mechanism 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 explicitâ€”stake coins, receive influence. Both systems work. Both systems concentrate power.

True decentralization requires a resource that cannot be accumulated, purchased, or transferred.

Time is that resource.

A node operating through a full Bitcoin halving cycle (210,000 blocks, ~4 years) accumulates the same influence whether owned by a billionaire or a student. This time is measured in Bitcoin blocks, resets at each halving, and is irreversible.

1.1 The Quantum Threat

Current cryptographic systems face an existential threat: quantum computers. Shorâ€™s algorithm breaks ECDSA, RSA, and X25519.

Éˆ Montana implements quantum-resistant cryptography: SPHINCS+ (NIST FIPS 205), SHA3-256, SHAKE256.

1.2 The Humanity Problem

TIME proves existence, not uniqueness. An attacker can create 100 keypairs, wait 4 years, and control a coordinated network.

The Hal Humanity System solves this: proving humanity, not just cryptographic identity.

Named after Hal Finney (1956-2014), who received the first Bitcoin transaction and understood Sybil resistance before anyone else.

---

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. In Proof of Stake, the problem is structural.

The solution is to build consensus on resources that cannot be unequally distributed.

- **Time** passes for everyone at the same rate. This is physics.
- **Humanity** cannot be multiplied. One person = one human.

---

3. ADAM: God of Time

ADAM implements 7 temporal levels for time synchronization with Bitcoin as primary oracle.

3.1 Seven Temporal Levels

```
â”Œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”
â”‚  ADAM: GOD OF TIME - 7 TEMPORAL LEVELS                      â”‚
â”œâ”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”
â”‚  Level 0: NODE_UTC         Hardware clock (UTC)             â”‚
â”‚  Level 1: GLOBAL_NTP       12 national NTP laboratories     â”‚
â”‚  Level 2: MEMPOOL_TIME     Bitcoin mempool observation      â”‚
â”‚  Level 3: BLOCK_TIME       Bitcoin block confirmation       â”‚
â”‚  Level 4: BITCOIN_ACTIVE   Normal operation, VDF not needed â”‚
â”‚  Level 5: VDF_FALLBACK     BTC down 2+ blocks, VDF active   â”‚
â”‚  Level 6: VDF_DEACTIVATE   BTC returned +20 blocks          â”‚
â””â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”€â”
```

3.2 Time Sources

Level 0: NODE_UTC Hardware clock on node. Baseline time source.

```
def get_node_utc() -> int:
    return int(time.time())
```

Level 1: GLOBAL_NTP 12 national NTP laboratories for global time consensus:

```
NTP_SERVERS = [
    "time.nist.gov",          # NIST, USA
    "time.windows.com",       # Microsoft
    "time.apple.com",         # Apple
    "ntp.ubuntu.com",         # Canonical
    "pool.ntp.org",           # NTP Pool
    "time.google.com",        # Google
    "time.cloudflare.com",    # Cloudflare
    "ntp.ix.ru",              # Russia
    "ntp.nict.jp",            # Japan
    "time.kriss.re.kr",       # Korea
    "time.nplindia.in",       # India
    "ntp.tpg.com.au",         # Australia
]
```

Level 2: MEMPOOL_TIME Bitcoin mempool observation. Pending transactions indicate network activity.

Level 3: BLOCK_TIME Bitcoin block confirmation timestamp. Authoritative time source.

Level 4: BITCOIN_ACTIVE Normal operation mode. Bitcoin provides time, VDF not needed.

3.3 VDF Fallback (Levels 5-6)

When Bitcoin is unavailable for 2+ blocks:

Level 5: VDF_FALLBACK

```
# Bitcoin down 2+ blocks - activate VDF
# Uses NODE_UTC and GLOBAL_NTP (levels 0, 1)
# SHAKE256 VDF from genesis block every 600 seconds

VDF_INTERVAL = 600  # 10 minutes (Bitcoin block time)

def compute_vdf_checkpoint(prev_hash: bytes, iterations: int) -> bytes:
    state = prev_hash
    for _ in range(iterations):
        state = shake_256(state).digest(32)
    return state
```

Level 6: VDF_DEACTIVATE

```
# Bitcoin returned for 20+ blocks - transition back
# VDF deactivates, Bitcoin time resumes as primary
BTC_RECOVERY_BLOCKS = 20
```

3.4 AdamSync Class

```
class AdamSync:
    """
    Time synchronization engine using 7-level hierarchy.

    Primary: Bitcoin block timestamps
    Fallback: SHAKE256 VDF when Bitcoin unavailable
    """

    def __init__(self, btc_oracle=None):
        self.btc_oracle = btc_oracle
        self.ntp_servers = NTP_SERVERS
        self.vdf_active = False
        self.btc_down_blocks = 0

    def get_network_time(self) -> Tuple[int, int]:
        """
        Get network time and confidence level.

        Returns: (timestamp, level)
        """
        # Try Bitcoin first (level 3-4)
        if self.btc_oracle:
            btc_time = self.btc_oracle.get_block_time()
            if btc_time:
                self.btc_down_blocks = 0
                return btc_time, 4  # BITCOIN_ACTIVE

        # Bitcoin unavailable
        self.btc_down_blocks += 1

        if self.btc_down_blocks >= 2:
            # Level 5: VDF_FALLBACK
```

```
            return self._get_vdf_time(), 5

        # Level 1: NTP consensus
        return self._get_ntp_time(), 1
```

---

4. Network Architecture

4.1 DAG-PHANTOM Ordering

Montana uses DAG-based consensus with PHANTOM ordering.

```
@dataclass
class BlockHeader:
    version: int
    prev_block_hash: bytes    # 32 bytes
    merkle_root: bytes        # 32 bytes
    timestamp: int
    height: int

    # VDF proof
    vdf_input: bytes
    vdf_output: bytes
    vdf_proof: bytes
    vdf_iterations: int

    # VRF proof (leader selection)
    vrf_output: bytes
    vrf_proof: bytes

    # Leader identity
    leader_pubkey: bytes      # 32 bytes
    leader_signature: bytes   # 64 bytes
    is_fallback_leader: bool  # True if no node won VRF lottery
```

4.2 Transaction Types

```
class TxType(IntEnum):
    COINBASE = 0           # Block reward
    STANDARD = 1           # Regular transfer
    SLASH = 2              # Slashing penalty
    APOSTLE_HANDSHAKE = 3  # 12 Apostles mutual trust
    EPOCH_PROOF = 4        # Bitcoin halving survival proof
    BTC_ANCHOR = 5         # Bitcoin block anchor timestamp
    RHEUMA_CHECKPOINT = 6  # RHEUMA stream checkpoint
```

4.3 Block Structure

```
@dataclass
class Block:
    header: BlockHeader
    transactions: List[Transaction]
```

```
    MAX_TRANSACTIONS = 50000
    MAX_BLOCK_SIZE = 32 * 1024 * 1024  # 32 MB
```

---

5. The Five Fingers of Adonis

Reputation system using five-dimensional assessment.

5.1 Weights

```
WEIGHT_TIME = 0.50       # THUMB - 50%
WEIGHT_INTEGRITY = 0.20  # INDEX - 20%
WEIGHT_STORAGE = 0.15    # MIDDLE - 15%
WEIGHT_EPOCHS = 0.10     # RING - 10%
WEIGHT_HANDSHAKE = 0.05  # PINKY - 5%
```

5.2 THUMB: TIME (50%)

The dominant factor. Saturates at 210,000 Bitcoin blocks (~4 years). Resets at each halving.

```
HALVING_INTERVAL = 210_000  # Bitcoin blocks per epoch

def compute_time_score(btc_height: int, first_seen_height: int) -> float:
    blocks_active = btc_height - first_seen_height
    blocks_in_epoch = btc_height % HALVING_INTERVAL
    return min(blocks_in_epoch / HALVING_INTERVAL, 1.0)
```

5.3 INDEX: INTEGRITY (20%)

No protocol violations. Decreases with misbehavior.

```
def compute_integrity_score(violations: int) -> float:
    return max(0.0, 1.0 - violations * 0.1)
```

5.4 MIDDLE: STORAGE (15%)

Percentage of chain history stored.

```
def compute_storage_score(stored_blocks: int, total_blocks: int) -> float:
    if total_blocks == 0:
        return 1.0
    return stored_blocks / total_blocks
```

5.5 RING: EPOCHS (10%)

Bitcoin halvings survived.

```
MAX_EPOCHS_FOR_SATURATION = 4  # 16 years

def compute_epochs_score(first_height: int, current_height: int) -> float:
    first_epoch = first_height // HALVING_INTERVAL
```

```
    current_epoch = current_height // HALVING_INTERVAL
    epochs_survived = current_epoch - first_epoch
    return min(epochs_survived / MAX_EPOCHS_FOR_SATURATION, 1.0)
```

5.6 PINKY: HANDSHAKE (5%)

Mutual trust bonds via the 12 Apostles system.

```
MAX_APOSTLES = 12

def compute_handshake_score(handshake_count: int) -> float:
    return min(handshake_count / MAX_APOSTLES, 1.0)
```

5.7 Total Reputation Score

```
def compute_reputation(node) -> float:
    return (
        WEIGHT_TIME * compute_time_score(node) +
        WEIGHT_INTEGRITY * compute_integrity_score(node) +
        WEIGHT_STORAGE * compute_storage_score(node) +
        WEIGHT_EPOCHS * compute_epochs_score(node) +
        WEIGHT_HANDSHAKE * compute_handshake_score(node)
    )
```

---

6. The Twelve Apostles

Each node chooses exactly 12 trust partners.

6.1 Design Philosophy

```
Trust Manifesto:
Before forming a handshake, ask yourself:

  Do I know this person?
  Not an avatar â€” a human.

  Do I trust them with my time?
  Willing to lose if they fail?

  Do I wish them longevity?
  Want them here for years?

  If any answer is NO â€” do not shake.
```

6.2 Constants

```
MAX_APOSTLES = 12                       # Exactly 12 trust partners
MIN_INTEGRITY_FOR_HANDSHAKE = 0.50    # 50% minimum integrity
HANDSHAKE_COOLDOWN = 86400            # 24 hours between handshakes
```

6.3 Seniority Bonus

Older nodes vouching for newer nodes carries more weight:

```
def compute_handshake_value(my_number: int, partner_number: int) -> float:
    """
    Node #1000 shakes #50:  value = 1 + log10(1000/50)  = 2.30
    Node #1000 shakes #999: value = 1 + log10(1000/999) = 1.00
    """
    base = 1.0
    if partner_number < my_number and partner_number > 0:
        bonus = math.log10(my_number / partner_number)
        return base + bonus
    return base
```

6.4 Handshake Protocol

Three-step process:

```
@dataclass
class HandshakeRequest:
    from_pubkey: bytes      # Requester's public key
    to_pubkey: bytes        # Target's public key
    timestamp: int          # Bitcoin time
    nonce: bytes            # Random nonce (32 bytes)
    signature: bytes        # SPHINCS+ signature
    btc_height: int         # Bitcoin height at request

@dataclass
class HandshakeResponse:
    request_hash: bytes     # Hash of original request
    from_pubkey: bytes      # Responder's public key
    accepted: bool          # Accept or reject
    timestamp: int          # Response timestamp
    signature: bytes        # SPHINCS+ signature

@dataclass
class Handshake:
    party_a: bytes          # First party public key
    party_b: bytes          # Second party public key
    request_sig: bytes      # Signature from party_a
    response_sig: bytes     # Signature from party_b
    btc_height: int         # Bitcoin height when established
    timestamp: int          # Timestamp when established
    status: HandshakeStatus # PENDING, ACCEPTED, REJECTED, DISSOLVED
```

6.5 Collective Slashing

Attack the network, lose your friends:

```
ATTACKER_QUARANTINE_BLOCKS = 180_000  # ~3 years
VOUCHER_INTEGRITY_PENALTY = 0.25      # -25% for those who vouched
ASSOCIATE_INTEGRITY_PENALTY = 0.10    # -10% for those vouched by attacker
```

7. The Hal Humanity System

Proof of Human, not just Proof of Time.

7.1 Graduated Trust Model

```
class HumanityTier(IntEnum):
    NONE = 0           # No humanity proof (legacy/bootstrap)
    HARDWARE = 1       # TPM/SecureEnclave/FIDO2 attestation
    SOCIAL = 2         # Custom social graph verification
    TIME_LOCKED = 3    # Bitcoin halving anchored proof
```

7.2 Apostle Limits per Tier

```
MAX_APOSTLES_HARDWARE = 3      # Tier 1: Bootstrap
MAX_APOSTLES_SOCIAL = 6        # Tier 2: Bridge
MAX_APOSTLES_TIMELOCKED = 12   # Tier 3: Ultimate (full Apostles)
```

7.3 Humanity Weights

```
HUMANITY_WEIGHT_HARDWARE = 0.3
HUMANITY_WEIGHT_SOCIAL = 0.6
HUMANITY_WEIGHT_TIMELOCKED = 1.0

HANDSHAKE_MIN_HUMANITY = 0.3  # At least one hardware attestation
```

7.4 Proof Validity Periods

```
HARDWARE_PROOF_VALIDITY = 86400 * 365      # 1 year
SOCIAL_PROOF_VALIDITY = 86400 * 365 * 2    # 2 years
TIMELOCK_PROOF_VALIDITY = 86400 * 365 * 4  # 4 years (one epoch)
```

7.5 Humanity Score Computation

```
def compute_humanity_score(proofs: List[HumanityProof]) -> float:
    """
    Scoring rules:
    1. Only valid proofs count
    2. Higher tier proofs take precedence
    3. Multiple proofs of same tier don't stack (max is taken)
    4. Cross-tier proofs add small bonuses (up to 0.1)

    Returns: Score between 0.0 and 1.0
    """
    valid_proofs = [p for p in proofs if p.is_valid]
    if not valid_proofs:
        return 0.0

    # Group by tier and take max weight per tier
    tier_scores = {}
    for proof in valid_proofs:
```

```
            tier = proof.tier
            current = tier_scores.get(tier, 0.0)
            tier_scores[tier] = max(current, proof.weight)

        # Highest tier is primary score
        max_tier = max(tier_scores.keys())
        primary_score = tier_scores[max_tier]

        # Lower tiers add small bonuses
        bonus = sum(score * 0.1 for tier, score in tier_scores.items() if tier
        bonus = min(bonus, 0.1)

        return min(primary_score + bonus, 1.0)
```

7.6 Sybil Economics

Tier	Sybil Cost per Identity
HARDWARE	$50-500 (physical device)
SOCIAL	Months/years (real connections)
TIME-LOCKED	4+ years (Bitcoin halving)

At Tier 3: 100 fake identities = 400 years of waiting.

---

8. Anti-Cluster Protection

Defense against coordinated attacks.

8.1 Behavioral Correlation Detection

```
class ClusterDetector:
    def compute_correlation(self, node_a, node_b) -> float:
        timing_corr = count_simultaneous() / total_actions  # 50%
        dist_corr = cosine_similarity(actions_a, actions_b)  # 30%
        height_corr = jaccard_similarity(heights_a, heights_b)  # 20%
        return 0.5*timing_corr + 0.3*dist_corr + 0.2*height_corr
```

8.2 Global Cluster Cap

No cluster can exceed 33% of network influence.

```
MAX_CLUSTER_INFLUENCE = 0.33
MAX_CORRELATION_THRESHOLD = 0.7
MIN_NETWORK_ENTROPY = 0.5
```

---

9. Post-Quantum Cryptography

Complete quantum-resistant cryptographic stack following NIST standards.

9.1 Algorithm Selection

Function	Algorithm	Standard	Security
Signatures	SPHINCS+-SHAKE-128f	NIST FIPS 205	128-bit PQ
Hashing	SHA3-256	NIST FIPS 202	128-bit PQ
VDF	SHAKE256	NIST FIPS 202	128-bit PQ
Key Exchange	ML-KEM-768	NIST FIPS 203	128-bit PQ
VRF	ECVRF-ED25519-SHA512-TAI	RFC 9381	Classical

9.2 Implementation (PROMETHEUS)

```
from pantheon.prometheus import pq_crypto

# SHA3-256 hashing
hash_value = pq_crypto.sha3_256(data)

# SPHINCS+ signatures
public_key, private_key = pq_crypto.sphincs_keygen()
signature = pq_crypto.sphincs_sign(message, private_key)
valid = pq_crypto.sphincs_verify(message, signature, public_key)

# ECVRF
vrf_output = pq_crypto.ecvrf_prove(private_key, input_data)
valid = pq_crypto.ecvrf_verify(public_key, input_data, vrf_output)

# SHAKE256 VDF
vdf_output = pq_crypto.shake256_vdf(input_data, iterations)
```

---

10. Attack Resistance

10.1 Attack Vector Matrix

Attack	Difficulty	Mitigation
Flash Takeover	IMPOSSIBLE	210,000 BTC blocks (~4 years) saturation
Slow Takeover	VERY HARD	Behavioral correlation + 33% cluster cap
Sybil via Keypairs	VERY HARD	Hal Humanity System (N Ã— 4 years)
Fake Apostle Network	HARD	Humanity tier limits (3/6/12)
Hardware Spoofing	HARD	Multiple attestation sources
Quantum Attack	IMPOSSIBLE	SPHINCS+, SHA3, SHAKE256
Eclipse Attack	BLOCKED	Minimum 8 outbound connections

10.2 Sybil Attack Cost

Fake Identities	Cost at Tier 3
1	4 years
10	40 years
100	400 years

---

11. Network Protocol

11.1 PAUL: Network Module

```
# Default ports
DEFAULT_P2P_PORT = 9333
DEFAULT_RPC_PORT = 8332

# Connection limits
MIN_OUTBOUND_CONNECTIONS = 8
MAX_INBOUND_CONNECTIONS = 125
```

11.2 Noise Protocol Encryption

All peer connections use Noise Protocol Framework, XX pattern:

```
from noiseprotocol import NoiseConnection

def establish_connection(peer_pubkey, my_keypair):
    conn = NoiseConnection.from_name(b'Noise_XX_25519_ChaChaPoly_SHA256')
    conn.set_keypair_from_private_bytes(Keypair.STATIC, my_keypair)
    conn.start_handshake()

    # Three-way handshake
    message_1 = conn.write_message()  # â†' peer
    conn.read_message(response_1)      # â†© peer
    message_2 = conn.write_message()  # â†' peer

    return conn  # Encrypted channel established
```

11.3 Bitcoin Oracle

Real-time BTC block verification via multiple APIs:

```
BTC_APIS = [
    "https://blockstream.info/api",
    "https://mempool.space/api",
    "https://blockchain.info",
]

def get_btc_block_hash(height: int) -> Optional[str]:
    results = []
    for api in BTC_APIS:
        try:
```

```
            hash = fetch_block_hash(api, height)
            results.append(hash)
        except:
            continue

    # Require 2/3 consensus
    if len(results) >= 2:
        return most_common(results)
    return None
```

---

12. Privacy

Tiered privacy model implemented in NYX module.

12.1 Privacy Tiers

Tier	Hidden	Fee Multiplier	Status
T0	Nothing	1Ã—	Production
T1	Receiver	2Ã—	Production

12.2 Ed25519 Operations (NYX)

```
class Ed25519Point:
    """Ed25519 curve operations via libsodium."""

    POINT_SIZE = 32
    SCALAR_SIZE = 32
    CURVE_ORDER = 2**252 + 27742317777372353535851937790883648493

    @staticmethod
    def scalarmult_base(scalar: bytes) -> bytes:
        """s * G"""
        return nacl.bindings.crypto_scalarmult_ed25519_base_noclamp(scalar

    @staticmethod
    def scalarmult(scalar: bytes, point: bytes) -> bytes:
        """s * P"""
        return nacl.bindings.crypto_scalarmult_ed25519_noclamp(scalar, poi

    @staticmethod
    def point_add(p: bytes, q: bytes) -> bytes:
        """P + Q"""
        return nacl.bindings.crypto_core_ed25519_add(p, q)
```

12.3 LSAG Ring Signatures

Linkable Spontaneous Anonymous Group signatures:

```
@dataclass
class LSAGSignature:
    key_image: bytes      # I = x * Hp(P) - links signatures
```

```
    c0: bytes             # Initial challenge scalar
    responses: List[bytes] # Response scalars for each ring member

class LSAG:
    @staticmethod
    def sign(message: bytes, ring: List[bytes],
             secret_index: int, secret_key: bytes) -> LSAGSignature:
        """
        Generate LSAG ring signature.
        Ring must have at least 2 members.
        """

    @staticmethod
    def verify(message: bytes, ring: List[bytes],
               signature: LSAGSignature) -> bool:
        """Verify ring closes: c_n == c_0"""

    @staticmethod
    def link(sig1: LSAGSignature, sig2: LSAGSignature) -> bool:
        """Check if two signatures are from same secret key."""
        return sig1.key_image == sig2.key_image
```

12.4 Stealth Addresses

CryptoNote-style one-time addresses:

```
@dataclass
class StealthKeys:
    view_secret: bytes   # a - view secret key
    spend_secret: bytes  # b - spend secret key
    view_public: bytes   # A = a*G
    spend_public: bytes  # B = b*G

class StealthAddress:
    @staticmethod
    def create_output(recipient_view_public: bytes,
                      recipient_spend_public: bytes) -> StealthOutput:
        """
        Create stealth output:
        1. Generate random r, R = r*G
        2. s = Hs(r*A)
        3. P = s*G + B (one-time address)
        """

    @staticmethod
    def scan_output(output: StealthOutput, view_secret: bytes,
                    spend_public: bytes) -> bool:
        """
        Check if output is ours:
        P' = Hs(a*R)*G + B
        Return P' == P
        """
```

```
    @staticmethod
    def derive_spend_key(output: StealthOutput, view_secret: bytes,
                         spend_secret: bytes) -> bytes:
        """Derive one-time spend key: x = Hs(a*R) + b"""
```

12.5 Pedersen Commitments

```
class Pedersen:
    """C = v*H + r*G"""

    @staticmethod
    def commit(value: int, blinding: bytes = None) -> PedersenCommitment:
        """Create Pedersen commitment."""

    @staticmethod
    def verify_sum(inputs: List, outputs: List, fee: int) -> bool:
        """Verify Î£ C_in = Î£ C_out + fee*H"""
```

---

13. Emission Schedule

13.1 Supply Parameters

```
# From config.py
INITIAL_REWARD = 3000       # 50 minutes in seconds
HALVING_INTERVAL = 210_000  # Bitcoin blocks per epoch
TOTAL_SUPPLY = 1_260_000_000  # 21 million minutes in seconds
```

13.2 Block Rewards

Era	Block Reward	Cumulative Supply
1	50 minutes	630,000,000 Éˆ
2	25 minutes	945,000,000 Éˆ
3	12.5 minutes	1,102,500,000 Éˆ
4	6.25 minutes	1,181,250,000 Éˆ

```
def get_block_reward(height: int) -> int:
    halvings = height // HALVING_INTERVAL
    if halvings >= 33:
        return 0
    return INITIAL_REWARD >> halvings
```

---

14. Implementation

14.1 Repository Structure

```
montana/
â”œâ”€â”€ pantheon/                          # 11 GODS
â”‚   â”œâ”€â”€ adam/                            # God of Time
```

```
â”‚   â”‚   â”œâ”€â”€ adam.py                  # 7 temporal levels, Bitcoin
â”‚   â”œâ”€â”€ paul/                        # Network
â”‚   â”‚   â”œâ”€â”€ network.py               # P2P, Noise Protocol
â”‚   â”œâ”€â”€ hades/                       # Storage
â”‚   â”‚   â”œâ”€â”€ database.py              # SQLite backend
â”‚   â”‚   â”œâ”€â”€ dag.py                   # DAG structure
â”‚   â”‚   â”œâ”€â”€ dag_storage.py           # DAG persistence
â”‚   â”œâ”€â”€ athena/                      # Consensus
â”‚   â”‚   â”œâ”€â”€ consensus.py             # Leader selection, finality
â”‚   â”‚   â”œâ”€â”€ engine.py                # Unified engine
â”‚   â”œâ”€â”€ prometheus/                  # Cryptography
â”‚   â”‚   â”œâ”€â”€ pq_crypto.py             # VDF, VRF, SPHINCS+
â”‚   â”œâ”€â”€ plutus/                      # Wallet
â”‚   â”‚   â”œâ”€â”€ wallet.py                # Argon2id, AES-256-GCM
â”‚   â”œâ”€â”€ nyx/                         # Privacy
â”‚   â”‚   â”œâ”€â”€ privacy.py               # LSAG, Stealth, Pedersen
â”‚   â”œâ”€â”€ themis/                      # Validation
â”‚   â”‚   â”œâ”€â”€ structures.py            # Block, Transaction
â”‚   â”œâ”€â”€ iris/                        # RPC Server
â”‚   â”‚   â”œâ”€â”€ rpc.py                   # JSON-RPC 2.0
â”‚   â”œâ”€â”€ apostles/                    # 12 Apostles Trust
â”‚   â”‚   â”œâ”€â”€ handshake.py             # Handshake protocol
â”‚   â”‚   â”œâ”€â”€ slashing.py              # Collective slashing
â”‚   â”œâ”€â”€ hal/                         # Humanity
â”‚       â”œâ”€â”€ humanity.py              # Core verification
â”‚       â”œâ”€â”€ reputation.py            # Five Fingers
â”‚       â”œâ”€â”€ behavioral.py            # Sybil detection
â”‚       â”œâ”€â”€ slashing.py              # Penalty manager
â”‚       â”œâ”€â”€ hardware.py              # TPM/Enclave/FIDO2
â”‚       â”œâ”€â”€ social.py                # Social graph
â”‚       â”œâ”€â”€ timelock.py              # Time-locked proofs
â”œâ”€â”€ tests/
    â”œâ”€â”€ test_integration.py
    â”œâ”€â”€ test_dag.py
    â”œâ”€â”€ test_fuzz.py
    â”œâ”€â”€ test_security_proofs.py
```

14.2 Module Summary (11 Gods)

Module	Name	Responsibility
ADAM	God of Time	7 temporal levels, Bitcoin anchor, VDF fallback
PAUL	Network	P2P, Noise Protocol, bootstrap
HADES	Storage	SQLite, DAG persistence
ATHENA	Consensus	Leader selection, finality
PROMETHEUS	Crypto	VDF, VRF, SPHINCS+, SHA3
PLUTUS	Wallet	Keys, transactions, AES-256-GCM
NYX	Privacy	T0/T1, LSAG, stealth addresses
THEMIS	Validation	Block/transaction validation
IRIS	RPC	JSON-RPC 2.0 server
APOSTLES	Trust	12 Apostles, seniority bonus
HAL	Humanity	Reputation, Sybil detection, slashing

14.3 Running a Node

```
pip install pynacl
python node.py --run
```

14.4 RPC Interface (IRIS)

```
# Get node status
curl -X POST http://localhost:8332 \
  -H "Content-Type: application/json" \
  -d '{"jsonrpc":"2.0","method":"getinfo","params":[],"id":1}'

# Get blockchain info
curl -X POST http://localhost:8332 \
  -d '{"jsonrpc":"2.0","method":"getblockchaininfo","params":[],"id":1}'

# Get balance
curl -X POST http://localhost:8332 \
  -d '{"jsonrpc":"2.0","method":"getbalance","params":[],"id":1}'
```

---

15. Conclusion

15.1 Security Guarantees

1. **No instant takeover:** TIME resets at each halving
2. **Cluster cap:** No coordinated group exceeds 33% influence
3. **Quantum resistance:** SPHINCS+, SHA3, SHAKE256
4. **Sybil resistance:** N fake identities = N Ã— 4 years
5. **Time-locked identity:** Bitcoin halving anchors cannot be faked
6. **Collective accountability:** 12 Apostles + slashing
7. **Bitcoin-anchored time:** 7 levels with VDF fallback
8. **Clean architecture:** 11 production-ready modules

15.2 Final Statement

É^ Montana removes capital as the basis of influence. The system uses: - **Time** â€” cannot be purchased, accelerated, or concentrated - **Humanity** â€” cannot be multiplied across Bitcoin halvings

With quantum-resistant cryptography and the Hal Humanity System, these guarantees extend indefinitely into the future.

â€œRunning bitcoinâ€ â€” Hal Finney, January 2009

â€œTime is priceless. Humanity is sacred. Now both have cryptographic proof.â€

É^

References

[1] S. Nakamoto, â€œBitcoin: A Peer-to-Peer Electronic Cash System,â€ 2008.

[2] D. Boneh et al., â€œVerifiable Delay Functions,â€ CRYPTO 2018.

[3] NIST FIPS 202, 203, 205 â€” Post-Quantum Standards, 2024.

[4] H. Finney, â€œRPOW - Reusable Proofs of Work,â€ 2004.

[5] R. Dunbar, â€œHow Many Friends Does One Person Need?â€ 2010.

[6] Y. Sompolinsky, A. Zohar, â€œPHANTOM: A Scalable BlockDAG Protocol,â€ 2018.

Appendix A: Constants Reference

```
# ============================================================
# ADAM: TIME LEVELS
# ============================================================
NTP_SERVERS = [
    "time.nist.gov", "time.windows.com", "time.apple.com",
    "ntp.ubuntu.com", "pool.ntp.org", "time.google.com",
    "time.cloudflare.com", "ntp.ix.ru", "ntp.nict.jp",
    "time.kriss.re.kr", "time.nplindia.in", "ntp.tpg.com.au"
]
VDF_INTERVAL = 600              # 10 minutes
BTC_RECOVERY_BLOCKS = 20        # Blocks before VDF deactivates

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

# ============================================================
# EPOCHS
# ============================================================
HALVING_INTERVAL = 210_000      # Bitcoin blocks per epoch
MAX_EPOCHS_FOR_SATURATION = 4   # 16 years

# ============================================================
# 12 APOSTLES
# ============================================================
MAX_APOSTLES = 12
MIN_INTEGRITY_FOR_HANDSHAKE = 0.50
HANDSHAKE_COOLDOWN = 86400      # 24 hours

# ============================================================
# HAL HUMANITY SYSTEM
```

```
# ============================================================
MAX_APOSTLES_HARDWARE = 3       # Tier 1
MAX_APOSTLES_SOCIAL = 6         # Tier 2
MAX_APOSTLES_TIMELOCKED = 12    # Tier 3

HUMANITY_WEIGHT_HARDWARE = 0.3
HUMANITY_WEIGHT_SOCIAL = 0.6
HUMANITY_WEIGHT_TIMELOCKED = 1.0

HANDSHAKE_MIN_HUMANITY = 0.3

HARDWARE_PROOF_VALIDITY = 86400 * 365      # 1 year
SOCIAL_PROOF_VALIDITY = 86400 * 365 * 2    # 2 years
TIMELOCK_PROOF_VALIDITY = 86400 * 365 * 4  # 4 years

# ============================================================
# SLASHING
# ============================================================
ATTACKER_QUARANTINE_BLOCKS = 180_000  # ~3 years
VOUCHER_INTEGRITY_PENALTY = 0.25      # -25%
ASSOCIATE_INTEGRITY_PENALTY = 0.10    # -10%

# ============================================================
# ANTI-CLUSTER
# ============================================================
MAX_CORRELATION_THRESHOLD = 0.7
MAX_CLUSTER_INFLUENCE = 0.33
MIN_NETWORK_ENTROPY = 0.5

# ============================================================
# NETWORK
# ============================================================
MIN_OUTBOUND_CONNECTIONS = 8
MAX_INBOUND_CONNECTIONS = 125
DEFAULT_P2P_PORT = 9333
DEFAULT_RPC_PORT = 8332

# ============================================================
# PRIVACY (NYX)
# ============================================================
CURVE_ORDER = 2**252 + 27742317777372353535851937790883648493
RING_SIZE = 11  # From config.PROTOCOL.RING_SIZE
```
