Ɉ Montana: A Peer-to-Peer Trust-Anchored Temporal Currency

Version 4.0

Alejandro Montana
alejandromontana@tutamail.com
December 2025

Abstract

A purely peer-to-peer electronic cash system built on human trust rather than computational proof. Version 4.0 uses Bitcoin block anchoring as time oracle with native VDF fallback. Security derives from real human relationships: each node selects 12 trusted partners who share responsibility. Slashing one node slashes all vouchers. Transactions flow through RHEUMA — a blockless stream with no TPS limit. Hardware is the only constraint.

Time cannot be bought. Trust cannot be purchased.

Table of Contents

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 4 years 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.

1.1 Design Principles

Principle	Description
Bitcoin as Clock	15 years battle-tested timestamps
VDF as Backup	Sovereign timekeeping if Bitcoin fails
Trust as Consensus	12 personal relationships secure the network
RHEUMA as Flow	Transactions stream with no block bottleneck

1.2 The Quantum Threat

Current cryptographic systems face an existential threat: quantum computers. Shor's algorithm, when executed on a sufficiently powerful quantum computer, breaks:

- **ECDSA/Ed25519**: Digital signatures used in Bitcoin, Ethereum, and most cryptocurrencies
- **RSA**: Public key encryption and VDF constructions
- **X25519**: Key exchange protocols

Conservative estimates place cryptographically-relevant quantum computers 10-15 years away. The "harvest now, decrypt later" attack model means adversaries may already be collecting encrypted data for future decryption.

Ɉ Montana v4.0 implements quantum-resistant cryptography to ensure long-term security.

1.3 Version 4.0 Paradigm Shift

Version 4.0 represents a fundamental redesign of the Montana protocol:

Component	v3.1	v4.0
Time Source	VDF (primary)	Bitcoin blocks (primary)
VDF Role	Core consensus	Fallback only

Component	v3.1	v4.0
Transaction Model	Block-based	RHEUMA (blockless stream)
Trust Bonds	10 handshakes	12 Apostles
Slashing	Individual	Collective responsibility
Geographic Scoring	Yes (10% weight)	Removed (VPN-spoofable)
Correlation Detection	Yes	Removed (false positives)
Cluster Cap	33%	Removed
TIME Saturation	180 days	210,000 blocks (~4 years), resets at halving
Ring Signatures	Production	Pending quantum-safe implementation

---

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. Bitcoin Time Oracle

Every Montana record references the latest Bitcoin block hash. Bitcoin has 99.98% uptime since 2009. Only 2 outages in history:

Year	Duration
2010	8h 27m
2013	6h 20m

Zero outages since 2013 — over 12 years.

3.1 Why Bitcoin?

Bitcoin provides:

- **Immutable ordering**: Block N always precedes block N+1
- **Global consensus**: All nodes agree on block sequence
- **Unforgeable timestamps**: Cannot create fake blocks
- **15 years of security**: Battle-tested infrastructure
- **Decentralized**: No single point of failure

3.2 Anchor Structure

```
{
  "t": "2025-12-30T12:00:00Z",
  "btc_height": 876543,
  "btc_hash": "0000...abc",
  "prev": "sha3...",
  "sig": "sphincs..."
}
```

Key Property: A record with `btc_height` N cannot exist before block N existed. Window of certainty: ~10 minutes.

3.3 Time Derivation

Montana measures time in Bitcoin blocks, not wall-clock seconds:

```
def get_montana_time(btc_height):
    """
    Montana time is measured relative to Bitcoin halvings.
    Each halving resets the epoch counter.
    """
    HALVING_INTERVAL = 210_000

    epoch = btc_height // HALVING_INTERVAL
    blocks_since_halving = btc_height % HALVING_INTERVAL

    return {
        'epoch': epoch,
        'blocks': blocks_since_halving,
        'saturation': min(blocks_since_halving / HALVING_INTERVAL, 1.0)
    }
```

3.4 Block Monitoring

```
class BitcoinOracle:
    EXPECTED_BLOCK_TIME = 600  # 10 minutes
    MAX_BLOCK_VARIANCE = 1800  # 30 minutes
    FALLBACK_TRIGGER = 2       # 2 consecutive missed blocks

    def __init__(self):
        self.last_block_time = None
        self.last_block_height = None
        self.missed_blocks = 0

    def on_new_block(self, height, hash, timestamp):
        self.last_block_height = height
        self.last_block_time = timestamp
        self.missed_blocks = 0

        return AnchorRecord(
            btc_height=height,
            btc_hash=hash,
            timestamp=timestamp
        )

    def check_fallback_needed(self):
        time_since_last = now() - self.last_block_time
        expected_blocks = time_since_last // self.EXPECTED_BLOCK_TIME

        if expected_blocks >= self.FALLBACK_TRIGGER:
            return True, "Bitcoin appears down, activating VDF fallback"
        return False, None
```

---

4. VDF Fallback

Montana is not dependent on Bitcoin. If Bitcoin stops producing blocks, the network switches to native VDF automatically.

4.1 Trigger Condition

2 consecutive expected blocks not produced (~20+ minutes without progress).

Rationale: - Normal variance can reach 30 minutes for a single block - Two consecutive missed blocks indicates serious network issue - False positives are acceptable (VDF is valid fallback)

4.2 VDF Construction: SHAKE256

Post-quantum VDF construction using SHAKE256:

```
class SHAKE256VDF:
    STATE_SIZE = 32  # bytes
    CHECKPOINT_INTERVAL = 1000

    def compute(self, input_bytes, iterations):
        """
        Sequential hash iteration.
        Properties:
        - Cannot parallelize
        - Quantum-safe
        - Verifiable via STARK proofs
        """
        state = SHA3_256(input_bytes)
        checkpoints = [state]

        for i in range(iterations):
            if i % self.CHECKPOINT_INTERVAL == 0:
                checkpoints.append(state)
            state = SHAKE256(state)

        return state, checkpoints

    def verify_stark(self, input_bytes, output, proof, iterations):
        return STARK_verify(input_bytes, output, proof, iterations)
```

4.3 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
VDF_MIN_ITERATIONS = 1_000        # Testing
VDF_MAX_ITERATIONS = 10**11       # Safety limit
```

4.4 STARK Proofs for VDF Verification

STARK (Scalable Transparent ARguments of Knowledge) enables O(log T) verification:

```
AIR (Algebraic Intermediate Representation):
  Constraint: next_state = SHAKE256(current_state)
  Boundary: state[0] = input, state[T] = output

Proof Generation:
  proof = STARK_prove(input, output, checkpoints, iterations)
  # Proof size: ~50-200 KB

Verification:
  valid = STARK_verify(input, output, proof, iterations)
  # Complexity: O(log T)
```

Properties: - Proof size: ~50-200 KB - Verification: O(log T) operations - Transparent: No trusted setup - Quantum-safe: Hash-based

4.5 Fallback Philosophy

Bitcoin has not had downtime since 2013. VDF is insurance against an event that may never occur. But it exists — **Montana survives Bitcoin's death.**

```
class TimeOracle:
    def __init__(self):
        self.bitcoin_oracle = BitcoinOracle()
        self.vdf_fallback = SHAKE256VDF()
        self.mode = "BITCOIN"

    def get_timestamp(self):
        if self.mode == "BITCOIN":
            needs_fallback, reason = self.bitcoin_oracle.check_fallback_needed()
            if needs_fallback:
                self.mode = "VDF"
                log.warning(f"Switching to VDF: {reason}")

        if self.mode == "VDF":
            # Check if Bitcoin is back
            if self.bitcoin_oracle.is_producing_blocks():
                self.mode = "BITCOIN"
                log.info("Bitcoin restored, switching back")

        if self.mode == "BITCOIN":
            return self.bitcoin_oracle.get_current_anchor()
        else:
            return self.vdf_fallback.compute_timestamp()
```

---

5. RHEUMA: Transaction Stream

RHEUMA (Greek: ῥεῦμα — flow, stream). No blocks. No batching. No waiting. No TPS limit.

5.1 Philosophy

Time flows continuously. Not in blocks. Not in intervals.

Transaction = drop in stream.
Once fallen — already in the river.
River flows to Bitcoin ocean.
From ocean, no return.

5.2 Transaction Structure

```
{
  "t": "2025-12-30T12:00:00.001Z",
  "btc": 876543,
  "prev": "sha3_of_previous",
  "seq": 847291,
  "from": "MONT_xyz...",
  "to": "MONT_abc...",
  "amount": 100000000,
  "nonce": 47,
  "sig": "sphincs..."
}
```

Field	Description
`t`	Timestamp (millisecond precision)
`btc`	Bitcoin block height anchor
`prev`	SHA3 hash of previous transaction in stream
`seq`	Global sequence number
`from`	Sender address
`to`	Recipient address
`amount`	Amount in base units
`nonce`	Sender's transaction counter
`sig`	SPHINCS+ signature

5.3 Why Unlimited TPS?

Traditional blockchain:

```
transactions wait for block
TPS = block_size / block_time
```

RHEUMA:

```
transactions wait for nothing
TPS = hardware_limit
```

No architectural ceiling.

5.4 Node Requirements

- Internet connection
- UTC time synchronization

That's all.

Want more throughput? Better hardware. Network doesn't wait for anyone.

5.5 Throughput Estimates

Scenario	TPS	Conditions
Minimum	10,000	100 nodes, basic hardware
Typical	100,000	1,000 nodes, good hardware
Optimal	1,000,000+	Optimized implementations
Theoretical	Unlimited	Hardware is only limit

5.6 Finality

Type	Timing	Description
Soft finality	Instant	All nodes see, all verify signature
Hard finality	~10 min	Next Bitcoin block — irreversible anchor

5.7 Double-Spend Protection

```
def validate_transaction(tx, state):
    # 1. Nonce must be exactly previous+1
    expected_nonce = state.get_nonce(tx.sender) + 1
    if tx.nonce != expected_nonce:
        return False, "Invalid nonce"

    # 2. Balance check
    if state.get_balance(tx.sender) < tx.amount + tx.fee:
        return False, "Insufficient balance"

    # 3. SPHINCS+ signature verification
    if not SPHINCS_verify(tx.sender_pubkey, tx.hash(), tx.sig):
        return False, "Invalid signature"

    # 4. First valid transaction wins
    if state.has_transaction_with_nonce(tx.sender, tx.nonce):
        return False, "Nonce already used"

    return True, None
```

Two transactions with same nonce = only first valid.

5.8 Stream Ordering

```
class RHEUMAStream:
    def __init__(self):
        self.sequence = 0
        self.prev_hash = GENESIS_HASH
        self.pending = PriorityQueue()  # By timestamp

    def submit(self, tx):
        # Validate
        valid, error = validate_transaction(tx, self.state)
        if not valid:
            return False, error

        # Assign sequence
        tx.seq = self.sequence
        tx.prev = self.prev_hash

        # Update stream
        self.sequence += 1
        self.prev_hash = SHA3_256(tx.serialize())

        # Broadcast
        self.broadcast(tx)

        return True, tx.seq

    def anchor_to_bitcoin(self, btc_block):
        """Called when new Bitcoin block arrives"""
        checkpoint = StreamCheckpoint(
            btc_height=btc_block.height,
            btc_hash=btc_block.hash,
            stream_seq=self.sequence,
            stream_hash=self.prev_hash,
            timestamp=now()
        )

        # Sign checkpoint
        checkpoint.sig = SPHINCS_sign(self.node_key, checkpoint.hash())

        return checkpoint
```

---

6. Node Identity

Every node receives a permanent sequential number upon first connection.

6.1 Number Assignment

```
#1 (first)
#2 (second)
...
#N (most recent)
```

6.2 Properties

Property	Description
Anonymous	Number reveals nothing about identity
Seniority	Lower number = older node
Immutable	Cannot be transferred
Scarce	Cannot buy a lower number — only arrive earlier

6.3 Registration Process

```
def register_node(public_key, first_block_produced):
    """
    Node registration occurs ONLY after first valid block/transaction.
    Cannot register through network messages alone (Sybil protection).
    """
    # Verify first contribution
    if not verify_contribution(public_key, first_block_produced):
        return None

    # Assign next available number
    node_number = get_next_node_number()

    # Record permanently
    node_registry[public_key] = NodeRecord(
        number=node_number,
        registered_at=first_block_produced.btc_height,
        public_key=public_key
    )

    return node_number
```

---

7. The Five Fingers of Adonis

Node influence computed from five dimensions. All weights mathematically justified.

Finger	Dimension	Weight	Rationale
Thumb	TIME	50%	Core innovation — must dominate
Index	INTEGRITY	20%	Honesty = 2/5 of time value
Middle	STORAGE	15%	Infrastructure contribution
Ring	EPOCHS	10%	Long-term commitment
Pinky	HANDSHAKE	5%	Social trust layer

7.1 THUMB: TIME (50%)

The dominant factor. Core innovation of Montana.

```
HALVING_INTERVAL = 210_000  # blocks (~4 years)
```

```
def compute_time_score(node, current_btc_height):
    """
    TIME resets to zero at every Bitcoin halving.
    No permanent dynasties — everyone competes fresh each epoch.
    """
    current_epoch = current_btc_height // HALVING_INTERVAL
    blocks_since_halving = current_btc_height % HALVING_INTERVAL

    # Node's blocks contributed this epoch
    node_blocks_this_epoch = count_blocks_since(
        node,
        epoch_start=current_epoch * HALVING_INTERVAL
    )

    # Saturation at full halving period
    f_time = min(node_blocks_this_epoch / HALVING_INTERVAL, 1.0)

    return f_time
```

Key Change from v3.1: TIME now saturates at 210,000 blocks (~4 years) and resets at each halving, not 180 days. This prevents permanent dynasties while providing longer-term commitment incentive.

7.2 INDEX: INTEGRITY (20%)

No protocol violations. Starts at 1.0, decreases with misbehavior.

```
PENALTIES = {
    'invalid_block': -0.15,
    'invalid_signature': -0.20,
    'equivocation': -1.00,  # + 180,000 block quarantine
}

RECOVERY_RATE = 0.01  # per 1,000 clean blocks

def compute_integrity_score(node):
    score = 1.0

    # Apply penalties
    for violation in node.violations:
        score += PENALTIES.get(violation.type, 0)

    # Apply recovery
    clean_blocks = node.blocks_since_last_violation
    recovery = (clean_blocks // 1000) * RECOVERY_RATE
    score = min(1.0, score + recovery)

    return max(0.0, score)
```

7.3 MIDDLE: STORAGE (15%)

Percentage of chain history stored.

```
def compute_storage_score(node):
    """
    Minimum for participation: 80%
    Full score: 100%
    """
    stored = node.blocks_stored
    total = get_total_blocks()

    ratio = stored / total
    if ratio < 0.80:
        return 0.0  # Cannot participate

    # Linear scaling from 80% to 100%
    f_storage = (ratio - 0.80) / 0.20
    return min(f_storage, 1.0)
```

7.4 RING: EPOCHS (10%)

Replaces GEOGRAPHY from v3 (was VPN-spoofable).

```
def compute_epochs_score(node):
    """
    Maximum 5 epochs = 20 years to full score.
    Cannot be faked — tied to chain history.
    """
    halvings_completed = node.halvings_participated
    f_epochs = min(halvings_completed / 5, 1.0)
    return f_epochs
```

Halvings Completed	Score	Approximate Time
0	0.0	< 4 years
1	0.2	4 years
2	0.4	8 years
3	0.6	12 years
4	0.8	16 years
5+	1.0	20+ years

7.5 PINKY: HANDSHAKE (5%)

Trust bonds with the Twelve Apostles.

```
def compute_handshake_score(node):
    """
    Maximum 12 handshakes.
    See Section 8 for Apostle requirements.
    """
    valid_handshakes = count_valid_handshakes(node)
    f_handshake = valid_handshakes / 12
    return f_handshake
```

7.6 Aggregate Score Computation

```
WEIGHTS = {
    'TIME': 0.50,
    'INTEGRITY': 0.20,
    'STORAGE': 0.15,
    'EPOCHS': 0.10,
    'HANDSHAKE': 0.05,
}

def compute_adonis_score(node, current_btc_height):
    scores = {
        'TIME': compute_time_score(node, current_btc_height),
        'INTEGRITY': compute_integrity_score(node),
        'STORAGE': compute_storage_score(node),
        'EPOCHS': compute_epochs_score(node),
        'HANDSHAKE': compute_handshake_score(node),
    }

    total = sum(WEIGHTS[dim] * scores[dim] for dim in WEIGHTS)

    # Apply slashing penalty if applicable
    if node.is_slashed:
        total = 0.0

    return total, scores
```

---

8. The Twelve Apostles

Each node chooses exactly 12 trust partners. No more. No less for full participation. These are your Apostles — people you vouch for with your reputation.

8.1 Why Twelve?

Reason	Explanation
Dunbar's inner circle	Humans maintain ~12-15 close relationships
Manageable responsibility	You can truly know 12 people
Game-theoretic limit	Prevents trust dilution

8.2 Requirements

```
def can_form_handshake(requester, target):
    """
    Both nodes must meet all requirements.
    """
    # 1. Mutual confirmation required
    if not (requester.wants_handshake(target) and
            target.wants_handshake(requester)):
        return False, "Not mutual"

    # 2. Partner INTEGRITY >= 50%
    if target.integrity_score < 0.50:
        return False, "Target integrity too low"
    if requester.integrity_score < 0.50:
        return False, "Requester integrity too low"

    # 3. Partner not currently slashed
    if target.is_slashed or requester.is_slashed:
        return False, "One party is slashed"

    # 4. Valid SPHINCS+ signatures from both parties
    # (verified during handshake protocol)

    return True, "Requirements met"
```

8.3 Seniority Bonus

Older nodes vouching for newer nodes carries more weight:

```
def compute_handshake_value(my_number, partner_number):
    """
    Handshakes with more senior nodes are worth more.

    Examples:
    - Node #1000 shakes #50:  value = 1 + log10(1000/50)  = 2.30
    - Node #1000 shakes #999: value = 1 + log10(1000/999) = 1.00
    """
    import math

    base = 1.0
    if partner_number < my_number:
        bonus = math.log10(my_number / partner_number)
        return base + bonus
    else:
        return base
```

8.4 Trust Manifesto

Before forming a handshake, ask yourself:

> ***Do I know this person?***
> *Not an avatar — 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 — do not shake.***

8.5 Handshake Protocol

```
class HandshakeProtocol:
    def initiate(self, my_key, target_pubkey):
        """Step 1: Initiate handshake request"""
        request = HandshakeRequest(
            from_pubkey=my_key.public,
            to_pubkey=target_pubkey,
            timestamp=get_bitcoin_time(),
            nonce=random_bytes(32)
        )
        request.signature = SPHINCS_sign(my_key.secret, request.hash())
        return request

    def respond(self, my_key, request):
        """Step 2: Respond to handshake request"""
        # Verify request signature
        if not SPHINCS_verify(request.from_pubkey,
                              request.hash(),
                              request.signature):
            return None, "Invalid request signature"

        # Check requirements
        can_shake, reason = can_form_handshake(
            get_node(request.from_pubkey),
            get_node(my_key.public)
        )
        if not can_shake:
            return None, reason

        response = HandshakeResponse(
            request_hash=request.hash(),
            from_pubkey=my_key.public,
            accepted=True,
            timestamp=get_bitcoin_time()
        )
        response.signature = SPHINCS_sign(my_key.secret, response.hash())
        return response, "Accepted"

    def finalize(self, request, response):
        """Step 3: Record handshake on-chain"""
        handshake = Handshake(
            party_a=request.from_pubkey,
            party_b=response.from_pubkey,
            request_sig=request.signature,
            response_sig=response.signature,
            btc_height=get_current_btc_height()
        )

        # Broadcast to network
        broadcast_handshake(handshake)

        return handshake
```

---

9. Slashing: Collective Responsibility

When a node attacks the network, everyone who trusted them pays.

9.1 Attack Types

Attack	Description	Detection
Equivocation	Signing conflicting blocks/transactions	Two valid signatures for same slot
Invalid Block	Producing malformed block	Validation failure
Invalid Signature	Submitting forged signature	Signature verification failure
Double Spend	Attempting to spend same funds twice	Conflicting transactions

9.2 Attack Consequences

```
SLASHING_PENALTIES = {
    'attacker': {
        'time_reset': True,           # TIME = 0
        'integrity_reset': True,       # INTEGRITY = 0
        'quarantine_blocks': 180_000,  # ~3 years
    },
    'voucher': {
        'integrity_penalty': -0.25,    # Who shook attacker's hand
    },
    'associate': {
        'integrity_penalty': -0.10,    # Attacker shook their hand
    },
}

def apply_slashing(attacker_node, attack_type):
    """
    Slash attacker and all connected parties.
    """
    # 1. Slash the attacker
    attacker_node.time_score = 0
    attacker_node.integrity_score = 0
    attacker_node.quarantine_until = (
        get_current_btc_height() + 180_000
    )
    attacker_node.is_slashed = True

    # 2. Slash vouchers (nodes who trusted the attacker)
    for handshake in attacker_node.incoming_handshakes:
        voucher = handshake.other_party
        voucher.integrity_score -= 0.25
        voucher.integrity_score = max(0, voucher.integrity_score)
        log.warning(f"Voucher {voucher.id} penalized for trusting attacker")

    # 3. Slash associates (nodes the attacker trusted)
    for handshake in attacker_node.outgoing_handshakes:
        associate = handshake.other_party
        associate.integrity_score -= 0.10
        associate.integrity_score = max(0, associate.integrity_score)
        log.warning(f"Associate {associate.id} penalized")

    # 4. Dissolve all handshakes
    dissolve_all_handshakes(attacker_node)

    return SlashingReport(
        attacker=attacker_node.id,
        vouchers_penalized=len(attacker_node.incoming_handshakes),
        associates_penalized=len(attacker_node.outgoing_handshakes),
        attack_type=attack_type
    )
```

9.3 Game Theory

Action	Outcome	Expected Value
Trust honest stranger	+0.4% influence	Positive (rare)
Stranger attacks	-5% influence (voucher penalty)	Very negative
Expected value of trusting strangers	**NEGATIVE**	

Optimal strategy: Only shake hands with people you genuinely trust.

9.4 Social Consequences

```
Your 12 Apostles know who you are (they trusted you).
They suffer financial loss because of your action.
Real-world relationships damaged permanently.

Network attack = social suicide.
```

9.5 Voucher Defense

Vouchers can recover, but slowly:

```
def voucher_recovery(node):
    """
    Recovery path after being penalized as voucher.
    """
    # Standard integrity recovery applies
    clean_blocks = node.blocks_since_penalty
    recovery = (clean_blocks // 1000) * 0.01
    node.integrity_score = min(1.0, node.integrity_score + recovery)

    # Can form new handshakes after integrity >= 50%
    if node.integrity_score >= 0.50:
        node.can_handshake = True
```

---

10. Post-Quantum Cryptography

Complete quantum-resistant cryptographic stack following NIST post-quantum standards.

10.1 Quantum Threat Timeline

Algorithm	Classical Security	Quantum Status	Threat Timeline
Ed25519	128-bit	BROKEN by Shor	10-15 years
RSA-2048	112-bit	BROKEN by Shor	10-15 years
SHA-256	256-bit	128-bit (Grover)	Secure
SPHINCS+-128f	128-bit	**SECURE**	N/A
SHA3-256	256-bit	128-bit (Grover)	Secure

10.2 SPHINCS+ Signatures (NIST FIPS 205)

Hash-based signature scheme with conservative security assumptions.

```
Algorithm: SPHINCS+-SHAKE-128f

Properties:
  - Public key: 32 bytes
  - Secret key: 64 bytes
  - Signature: 17,088 bytes (~17 KB)
  - Security: 128-bit post-quantum

Signing:
  signature = SPHINCS_sign(secret_key, message)

Verification:
  valid = SPHINCS_verify(public_key, message, signature)
```

Security Basis: Security relies only on hash function properties (SHA3/SHAKE). No number-theoretic assumptions that quantum computers break.

Trade-off: Larger signatures (~17 KB vs 64 bytes for Ed25519) increase bandwidth but provide quantum resistance.

10.3 SHA3-256 Hashing (NIST FIPS 202)

Keccak-based hash function:

```
Properties:
  - Output: 256 bits
  - Rate: 1088 bits
  - Capacity: 512 bits
  - Rounds: 24

Usage:
  block_hash = SHA3-256(block_header)
  merkle_root = SHA3-256(left || right)
  address = SHA3-256(public_key)
```

Quantum Security: Grover's algorithm reduces security from 256-bit to 128-bit — still computationally infeasible (2^128 operations).

10.4 ML-KEM Key Encapsulation (NIST FIPS 203)

Lattice-based key exchange replacing X25519:

```
Algorithm: ML-KEM-768 (Kyber)

Key Generation:
  (secret_key, public_key) = ML_KEM.keygen()

Encapsulation:
  (ciphertext, shared_secret) = ML_KEM.encapsulate(public_key)

Decapsulation:
  shared_secret = ML_KEM.decapsulate(secret_key, ciphertext)
```

Used for: - P2P connection encryption - Wallet key exchange - Secure channel establishment

10.5 Post-Quantum VRF

Hash-based VRF construction for leader selection (when applicable):

```
def vrf_prove(secret_key, alpha):
    """
    Verifiable Random Function using SPHINCS+.
    """
    # Compute deterministic output
    beta = SHA3_256(secret_key || alpha || b"vrf_output")

    # Generate proof (SPHINCS+ signature)
    proof = SPHINCS_sign(secret_key, alpha || beta)

    return beta, proof

def vrf_verify(public_key, alpha, beta, proof):
    return SPHINCS_verify(public_key, alpha || beta, proof)
```

10.6 Crypto-Agility Architecture

```
┌──────────────────────────────┐
│     CryptoProvider           │
│   (Abstract Interface)       │
└──────────────────────────────┘
        │       │       │
   ┌────┘       │       └────┐
   ▼            ▼            ▼
┌───────────┐ ┌───────────┐ ┌───────────┐
│  Legacy   │ │Post-Quantum│ │  Hybrid   │
│           │ │           │ │           │
│  Ed25519  │ │ SPHINCS+  │ │ Ed25519 + │
│  SHA-256  │ │ SHA3-256  │ │ SPHINCS+  │
│ Wesolowski│ │ SHAKE256  │ │           │
└───────────┘ └───────────┘ └───────────┘
```

Runtime switching enables: - Gradual migration from legacy to post-quantum - Backward compatibility during transition - Testing without network disruption

10.7 Performance Impact

Operation	Ed25519	SPHINCS+-128f	Factor
Key Gen	<1 ms	~50 ms	50×
Sign	<1 ms	~100 ms	100×
Verify	<1 ms	~10 ms	10×
Signature	64 B	17,088 B	267×

RHEUMA Mitigation: Since RHEUMA has no block size limits, larger signatures do not create bottlenecks. Bandwidth is the only constraint.

---

11. Attack Resistance Analysis

11.1 Attack Vector Matrix

Attack	Difficulty	Defense	Quantum-Safe
Flash Takeover	IMPOSSIBLE	210,000 blocks (~4 years) to saturate TIME	Yes
Sybil (many nodes)	HARD	Each node needs 12 real human trust relationships	Yes
Nation-State	HARD	10,000 nodes require 120,000 real humans willing to vouch	Yes
Bitcoin Attack	IMPOSSIBLE	VDF fallback exists — network survives	Yes
Quantum Attack	IMPOSSIBLE	SPHINCS+ post-quantum signatures	Yes
Double Spend	IMPOSSIBLE	Nonce sequence + instant propagation	Yes

11.2 Flash Takeover Attack

Attack: Acquire resources, immediately gain 51% influence.

Defense: TIME dimension requires 210,000 blocks (~4 years) to saturate.

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

```
def time_to_attack():
    """
    Minimum time to gain significant influence.
    """
    # Even with infinite resources:
```

```
    # - Cannot accelerate Bitcoin blocks
    # - Cannot forge TIME accumulation
    # - Must wait 210,000 blocks for full TIME saturation

    minimum_blocks = 210_000
    average_block_time = 10  # minutes

    minimum_time = minimum_blocks * average_block_time
    return minimum_time  # ~4 years
```

11.3 Sybil Attack

Attack: Create many identities to gain influence.

Defense: Multi-layered:

1. **TIME**: Each identity needs ~4 years to saturate
2. **HANDSHAKE**: Each identity needs 12 REAL human relationships
3. **STORAGE**: Each identity must store 80%+ of chain
4. **EPOCHS**: Cannot fake participation across halvings

```python
def sybil_cost(num_identities):
    """
    Cost to run N Sybil identities.
    """
    # Hardware/bandwidth per identity
    hardware_cost = num_identities * HARDWARE_COST_PER_NODE

    # Human relationships needed
    humans_needed = num_identities * 12  # Each needs 12 Apostles

    # TIME investment (cannot be parallelized across identities)
    time_needed = 210_000  # blocks, same for all

    return {
        'hardware': hardware_cost,
        'humans': humans_needed,
        'time_blocks': time_needed,
        'feasible': humans_needed < 100  # Practical limit
    }
```

Result: Creating 1,000 Sybil nodes requires convincing 12,000 real humans to vouch for fake identities. Socially infeasible.

11.4 Nation-State Attack

Attack: Government deploys resources to control network.

Defense:

1. **Human requirement**: 10,000 nodes need 120,000 trusted relationships
2. **TIME**: Cannot accelerate past Bitcoin block production
3. **Collective slashing**: If discovered, all colluding nodes and their vouchers are slashed
4. **VDF fallback**: Even if Bitcoin is attacked, network continues

11.5 Bitcoin Dependency Attack

Attack: Attack Bitcoin to destabilize Montana.

Defense: VDF fallback activates automatically.

```
def handle_bitcoin_outage():
    """
    Network continues operating without Bitcoin.
    """
    # 1. Detect outage (2 consecutive missed blocks)
    # 2. Switch to VDF time oracle
    # 3. Continue processing RHEUMA transactions
    # 4. Resume Bitcoin anchoring when available

    # Network is NEVER down due to Bitcoin issues
    pass
```

11.6 What Was Removed from v3.1

Feature	Reason for Removal
Correlation detection	False positives, evadable with jitter
33% cluster cap	Depended on detection
Geography scoring	VPN-spoofable
Ring signatures	Pending quantum-safe implementation

Philosophy: Simpler = fewer bugs = harder to attack.

---

12. Known Limitations

We do not hide our weaknesses.

12.1 Bitcoin Dependency

Limitation: Normal operation requires Bitcoin blocks. If extended downtime occurs, VDF fallback activates.

Mitigation: Bitcoin has not had downtime since 2013. VDF provides sovereign backup.

12.2 Trust Bootstrapping

Limitation: New nodes need 12 handshakes. In early network, finding 12 trustworthy partners may be difficult.

Mitigation: Cold start problem is real. Early participants must know each other. Network grows organically through existing trust relationships.

12.3 Signature Size

Limitation: SPHINCS+ signatures are 17KB each (vs Ed25519 64 bytes). Larger bandwidth requirement.

Future Work: Signature aggregation.

12.4 No Privacy Layer

Limitation: Ring signatures removed pending quantum-safe implementation. Transactions are pseudonymous, not anonymous.

Future Work: Post-quantum ring signatures when available.

12.5 Halving Reset

Limitation: Every ~4 years all TIME scores reset to zero. Intentional disruption to prevent permanent dynasties.

Mitigation: EPOCHS dimension rewards long-term participants (up to 20 years). Halving reset affects TIME only.

12.6 Small Network Vulnerability

Limitation: With fewer than 50 nodes, game-theoretic assumptions weaken.

Mitigation: Network should not be considered production-ready until achieving 50+ diverse nodes with genuine trust relationships.

---

13. Network Protocol

13.1 Message Types

Type	Name	Description
0	VERSION	Initial handshake
1	VERACK	Handshake acknowledgment
2	GETADDR	Request peer addresses
3	ADDR	Peer address list
4	INV	Inventory announcement
5	GETDATA	Request full objects
6	NOTFOUND	Object not available
10	TX	Transaction (RHEUMA stream)
11	CHECKPOINT	Bitcoin anchor checkpoint
12	HANDSHAKE_REQ	Apostle handshake request
13	HANDSHAKE_RESP	Apostle handshake response
14	SLASH	Slashing evidence
20	PING	Keepalive
21	PONG	Keepalive response

Type	Name	Description
100	NOISE_INIT	Noise handshake initiate
101	NOISE_RESP	Noise handshake respond
102	NOISE_FINAL	Noise handshake complete

13.2 Encryption: Noise Protocol

All peer connections use Noise Protocol Framework, XX pattern:

```
Initiator (I)                         Responder (R)
     |                                      |
     |-- e, es -------------------------->   | # I sends ephemeral, DH
     |                                      |
     |<-- e, ee, s, es ------------------   | # R sends ephemeral, static
     |                                      |
     |-- s, se -------------------------->   | # I sends static
     |                                      |
     |<========= Encrypted =============>   | # All further messages encrypted
```

Cipher Suite: - Key exchange: ML-KEM (post-quantum) - Encryption: ChaCha20-Poly1305 (AEAD) - Hash: SHA3-256

13.3 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_TX = 50
SCORE_DOS_ATTEMPT = 100
```

13.4 RHEUMA Transaction Propagation

```
def propagate_transaction(tx):
    """
    RHEUMA transactions propagate immediately.
    No batching, no waiting for blocks.
    """
    # Validate locally
    if not validate_transaction(tx):
        return False

    # Add to local stream
    add_to_stream(tx)

    # Broadcast to all peers
    for peer in connected_peers:
        peer.send(Message(type=TX, payload=tx.serialize()))
    return True
```

13.5 Bitcoin Checkpoint Propagation

```
def propagate_checkpoint(checkpoint):
    """
    Checkpoints anchor RHEUMA stream to Bitcoin.
    Propagated when new Bitcoin block arrives.
    """
    # Verify checkpoint
    if not verify_checkpoint(checkpoint):
        return False

    # Store locally
    store_checkpoint(checkpoint)

    # Broadcast
    for peer in connected_peers:
        peer.send(Message(type=CHECKPOINT, payload=checkpoint.serialize()))

    return True
```

---

14. Emission Schedule

14.1 Supply Parameters

Parameter	**Value**
Name	Ɉ Montana
Symbol	Ɉ
Ticker	$MONT
Base unit	1 Ɉ = 1 second
Total supply	1,260,000,000 Ɉ (21 million minutes)
Smallest unit	10^-8 Ɉ (10 nanoseconds)

14.2 Block Rewards

Epoch	**Blocks**	**Reward**	**Cumulative**
0	1 – 210,000	50 min	630M Ɉ
1	210,001 – 420,000	25 min	945M Ɉ
2	420,001 – 630,000	12.5 min	1,102.5M Ɉ
3	630,001 – 840,000	6.25 min	1,181.25M Ɉ
...	...	halving	...
Final	—	0	1,260M Ɉ

Halving interval: 210,000 blocks (~4 years)
Full emission: ~132 years

14.3 Reward Distribution in RHEUMA

```
def distribute_reward(checkpoint):
    """
    Rewards distributed based on Adonis scores.
    """
    current_epoch = checkpoint.btc_height // HALVING_INTERVAL

    # Calculate reward for this checkpoint
    base_reward = 3000  # 50 minutes in seconds
    reward = base_reward >> current_epoch  # Halving

    if reward == 0:
        return  # Emission complete

    # Get all active nodes
    active_nodes = get_active_nodes()

    # Calculate total weight
    total_weight = sum(
        compute_adonis_score(node, checkpoint.btc_height)[0]
        for node in active_nodes
    )

    # Distribute proportionally
    for node in active_nodes:
        score, _ = compute_adonis_score(node, checkpoint.btc_height)
        node_reward = reward * (score / total_weight)
        credit_balance(node, node_reward)
```

14.4 Transaction Fees

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

```
# Minimum fee: 1 Ɉ (1 second)
MIN_FEE = 1

def calculate_fee(tx):
    """
    Fee based on transaction size.
    """
    size_bytes = len(tx.serialize())
    fee = max(MIN_FEE, size_bytes // 1000)
    return fee

# Priority: Higher fee = earlier processing
```

---

15. Implementation

15.1 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 if fallback needed) - 16 GB RAM - SSD storage - Reliable internet connection

15.2 Repository Structure

```
montana/
├── pantheon/
│   ├── prometheus/           # Cryptography
│   │   ├── crypto.py         # Legacy primitives
│   │   ├── crypto_provider.py # Abstraction layer
│   │   ├── pq_crypto.py      # Post-quantum
│   │   └── winterfell_ffi.py # STARK FFI
│   ├── athena/               # Consensus
│   │   ├── bitcoin_oracle.py # Bitcoin time oracle
│   │   ├── vdf_fallback.py   # VDF fallback
│   │   └── adonis.py         # Five Fingers scoring
│   ├── hermes/               # Networking
│   │   ├── rheuma.py         # Transaction stream
│   │   └── p2p.py            # Peer-to-peer layer
│   ├── hades/                # Storage
│   ├── plutus/               # Wallet
│   └── apostles/             # Trust system
│       ├── handshake.py      # Apostle handshakes
│       └── slashing.py       # Collective responsibility
├── winterfell_stark/         # Rust STARK prover
│   ├── Cargo.toml
│   ├── src/lib.rs
│   └── build.sh
└── tests/
    ├── test_integration.py
    ├── test_rheuma.py
    ├── test_apostles.py
    └── test_pq_crypto.py
```

15.3 Configuration

```
from config import NodeConfig, CryptoConfig

config = NodeConfig()
config.crypto = CryptoConfig(
    backend="post_quantum",     # legacy | post_quantum | hybrid
    sphincs_variant="fast",     # fast | secure
    vdf_backend="shake256",     # wesolowski | shake256
    stark_proofs_enabled=True
)

config.consensus = ConsensusConfig(
    time_oracle="bitcoin",      # bitcoin | vdf
    vdf_fallback_enabled=True,
    halving_interval=210_000,
)

config.apostles = ApostlesConfig(
    max_handshakes=12,
    min_integrity=0.50,
    slashing_enabled=True,
)

provider = config.crypto.initialize_provider()
```

15.4 Environment Variables

```
MONT_CRYPTO_BACKEND=post_quantum
MONT_SPHINCS_VARIANT=fast
MONT_VDF_BACKEND=shake256
MONT_TIME_ORACLE=bitcoin
MONT_NETWORK=mainnet
MONT_DATA_DIR=~/.montana
MONT_MAX_APOSTLES=12
```

15.5 Running a Node

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

---

16. Conclusion

16.1 What We Guarantee

Guarantee	Mechanism
No instant takeover	210,000 blocks minimum (~4 years)
No permanent dynasties	TIME resets at halving
Social accountability	Attack = betray 12 real friends
Bitcoin-grade timestamps	15 years proven security
Sovereign fallback	VDF if Bitcoin fails
Quantum resistance	SPHINCS+ signatures
Unlimited throughput	RHEUMA has no TPS ceiling

16.2 What We Cannot Guarantee

Limitation	Reason
Trust relationships are genuine	Humans must verify
Privacy	Ring signatures pending quantum-safe implementation
Smooth halving transitions	Intentional disruption
Early network stability	< 50 nodes

16.3 Comparison

Property	PoW	PoS	Montana v4
Attack resource	Energy	Capital	Time + Trust
Flash attack	Buy hardware	Borrow capital	Impossible

Property	PoW	PoS	Montana v4
Social cost	None	None	12 betrayed friends
TPS limit	Block size	Block size	None
Quantum-safe	No	No	Yes

16.4 Future Work

1. **Signature aggregation**: Reduce bandwidth impact
2. **ML-DSA (Dilithium)**: Alternative PQ signature scheme
3. **Post-quantum ring signatures**: Quantum-safe anonymity
4. **Geographic proofs**: Latency triangulation, TEE attestation
5. **Hardware wallet**: Ledger/Trezor integration
6. **Mobile clients**: Light client protocol
7. **Cross-chain bridges**: Interoperability with other networks

16.5 Final Statement

Ɉ Montana does not require trust in institutions, corporations, or algorithms. It requires trust in people you choose — your Twelve Apostles.

The network is robust because humans are accountable to each other.

Attack the network, lose your friends.

This is not a bug. This is the design.

> *Time is priceless. Trust is sacred.*
> *Both now have a protocol.*

Ɉ

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] NIST, "FIPS 202: SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," 2015.

[5] NIST, "FIPS 205: Stateless Hash-Based Digital Signature Standard (SLH-DSA)," 2024.

[6] NIST, "FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard (ML-KEM)," 2024.

[7] D. J. Bernstein et al., "SPHINCS+: Submission to the NIST Post-Quantum Cryptography Standardization," 2022.

[8] E. Ben-Sasson et al., "Scalable, transparent, and post-quantum secure computational integrity," 2018.

[9] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," 2018.

[10] R. I. M. Dunbar, "Neocortex size as a constraint on group size in primates," Journal of Human Evolution, 1992.

[11] E. Hughes, "A Cypherpunk's Manifesto," 1993.

[12] H. Finney, "RPOW - Reusable Proofs of Work," 2004.

[13] T. Perrin, "The Noise Protocol Framework," 2018.

[14] 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_EPOCHS = 0.10     # RING (replaces GEOGRAPHY)
WEIGHT_HANDSHAKE = 0.05  # PINKY

# ==============================================================
# TIME PARAMETERS
# ==============================================================
HALVING_INTERVAL = 210_000        # blocks (~4 years)
TIME_RESETS_AT_HALVING = True     # No permanent dynasties

# ==============================================================
# APOSTLE PARAMETERS
# ==============================================================
MAX_APOSTLES = 12                 # Exactly 12 trust partners
MIN_INTEGRITY_FOR_HANDSHAKE = 0.50
SENIORITY_BONUS_ENABLED = True

# ==============================================================
# SLASHING PARAMETERS
# ==============================================================
ATTACKER_QUARANTINE_BLOCKS = 180_000  # ~3 years
VOUCHER_INTEGRITY_PENALTY = 0.25
ASSOCIATE_INTEGRITY_PENALTY = 0.10
INTEGRITY_RECOVERY_RATE = 0.01        # per 1,000 clean blocks

# ==============================================================
# STORAGE THRESHOLDS
# ==============================================================
K_STORAGE_MINIMUM = 0.80   # 80% minimum for participation
K_STORAGE_FULL = 1.00      # 100% for full score

# ==============================================================
# EPOCHS PARAMETERS
# ==============================================================
MAX_EPOCHS_FOR_FULL_SCORE = 5  # 5 halvings = 20 years

# ==============================================================
# BITCOIN ORACLE
# ==============================================================
EXPECTED_BLOCK_TIME = 600      # 10 minutes
```

```
VDF_FALLBACK_TRIGGER = 2       # 2 consecutive missed blocks

# ==============================================================
# 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

# ==============================================================
# CRYPTOGRAPHY
# ==============================================================
HASH_ALGORITHM = "SHA3-256"       # Post-quantum
SIGNATURE_ALGORITHM = "SPHINCS+"  # Post-quantum
VDF_ALGORITHM = "SHAKE256"        # Post-quantum
KEM_ALGORITHM = "ML-KEM-768"      # Post-quantum

# ==============================================================
# VDF (FALLBACK)
# ==============================================================
VDF_CHECKPOINT_INTERVAL = 1000     # STARK checkpoints
VDF_MIN_ITERATIONS = 1_000
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
MIN_FEE = 1                        # 1 second
```

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
2.6	Dec 2025	All security properties proven via executable tests
3.0	Dec 2025	Post-quantum cryptography: SPHINCS+, SHA3-256, SHAKE256 VDF, STARK proofs, ML-KEM, crypto-agility layer
3.1	Dec 2025	Network security hardening: static IP validation, VPN/proxy detection, Sybil protection, Eclipse defense, rate limiting, wallet encryption
4.0	**Dec 2025**	**Paradigm shift**: Bitcoin time oracle (primary), VDF fallback, RHEUMA blockless transaction stream, 12 Apostles trust system, collective slashing responsibility, removed correlation detection/cluster cap/geography scoring (simplification), TIME resets at halving (no dynasties), EPOCHS replaces GEOGRAPHY

Appendix C: Migration from v3.1 to v4.0

C.1 Breaking Changes

Component	v3.1	v4.0	Migration
TIME saturation	180 days	210,000 blocks	Recalculate all scores
GEOGRAPHY	10% weight	Removed	Use EPOCHS instead
Handshakes	10 max	12 max	Allow 2 additional
Blocks	Required	RHEUMA stream	New transaction model
Correlation detection	Active	Removed	Simplify codebase
Cluster cap	33%	Removed	Rely on trust system

C.2 Node Upgrade Process

```
def upgrade_to_v4():
    """
    Upgrade node from v3.1 to v4.0.
    """
    # 1. Update configuration
    config.version = "4.0"
    config.time_oracle = "bitcoin"
    config.transaction_model = "rheuma"

    # 2. Recalculate Adonis scores
    for node in all_nodes:
        # Replace GEOGRAPHY with EPOCHS
        node.geography_score = None
        node.epochs_score = calculate_epochs(node)

        # Recalculate TIME based on Bitcoin blocks
        node.time_score = calculate_time_v4(node)

    # 3. Allow additional handshakes
    config.max_handshakes = 12

    # 4. Enable RHEUMA
    initialize_rheuma_stream()

    # 5. Connect to Bitcoin oracle
    bitcoin_oracle.connect()

    log.info("Upgrade to v4.0 complete")
```

C.3 Backward Compatibility

v4.0 nodes can communicate with v3.1 nodes during transition period, but: - v3.1 nodes cannot process RHEUMA transactions - v3.1 nodes will not understand Bitcoin anchors - Recommended: All nodes upgrade simultaneously

---

Ɉ Montana Technical Specification v4.0
December 2025