EC Protocol: Consensus Semantics and Byzantine Fault Model

Author: Lars Szuwalski

Year: 2025

Project: https://github.com/EcProtocol

Abstract

We present EC (Echo Consent), a distributed consensus protocol for maintaining shared state across a permissionless network without global coordination or immutable history. Unlike blockchain systems that prioritize permanent finality, EC treats consensus as a continuous, locality-scoped process where agreement reflects current majority alignment rather than irreversible commitment.

The protocol introduces several novel mechanisms: (1) **Proof-of-Aligned Storage (PoAS)**, a challenge-response scheme that verifies nodes store data consistent with the network majority, enabling peer selection without global knowledge; (2) **organic sharding** through address-space locality, where nodes maintain responsibility for nearby regions without explicit shard assignment; and (3) **token scattering**, an emergent security property where typical usage patterns naturally distribute tokens across neighborhoods, defeating targeted capture attacks.

EC operates under standard honest-majority assumptions but explicitly acknowledges and documents their implications. Safety is prioritized over liveness: committed state is locally immutable, while global consistency emerges probabilistically through continuous alignment pressure. Transactions support atomic multi-token operations, enabling applications like stateless gift cards, decentralized naming, and verifiable voting without shared infrastructure.

We provide formal analysis of vote threshold confidence (91% probability of majority support when threshold reached), neighborhood capture costs (linear scaling with network size), and PoAS discrimination (20× ratio between aligned and misaligned nodes at $\theta=8$ overlap threshold). The protocol occupies a distinct design space suited to applications where current shared agreement matters more than permanent historical record.

---

1. Scope and Intent

This document specifies the *consensus semantics*, *fault assumptions*, and *Byzantine tolerance model* of the EC protocol. The name "EC" stands for **Echo Consent** — reflecting the core mechanism where nodes broadcast their votes and wait for echoed responses from the network to establish consent.

This is not a wire protocol specification, nor does it define transaction formats or network APIs. Its purpose is to state, precisely and defensibly, **what properties the system claims and what it explicitly does not claim**.

The design goal is to enable a decentralized, permissionless network that maintains a shared, queryable mapping from opaque tokens to opaque blocks, while prioritizing **fault isolation, adaptability, and continuous operation** over immutable historical finality.

---

2. System Model

2.1 Nodes and State

Let:

- $N(t)$ be the set of active nodes at time t
- Each node stores a *finite subset* of key–value mappings: $S_i(t) \subseteq T \times B$, where T and B are the spaces of 256-bit opaque identifiers (tokens and blocks).

Nodes do not store the full global state. Storage is *organically sharded* based on node address, churn, and query-driven sampling.

2.2 Queries

Nodes receive queries of the form:

```
q = H(token)
```

where H is the protocol hash function (Blake3, see Appendix A). A node can respond to a query only if it knows the corresponding token preimage and associated block.

Queries are indistinguishable with respect to their origin (user-driven vs. protocol-driven). Election-related queries form a strict minority of total queries.

---

3. Global Assumptions and Invariants

This document analyzes the security properties of EC protocol under an explicit set of assumptions. These assumptions are not hidden; all guarantees and hypotheses are conditional on them.

A1. Uniform Address Distribution

Node addresses are assumed to be uniformly distributed over the 256-bit address space as a consequence of memory-hard proof-of-work address derivation. No party can choose addresses arbitrarily without incurring proportional computational cost.

A2. Majority Honesty Over Time

Security claims are evaluated over time intervals during which a strict majority (>50%) of active nodes are honest and maintain aligned storage for frequently queried state. No guarantees are made across intervals where this assumption does not hold.

A3. No Global Membership Knowledge

Nodes do not possess, and are not expected to obtain, a complete or consistent view of global membership. All decisions are made based on local observations, peer sampling, and query-driven interaction.

A4. Continuous Churn and Re-Election

Peer sets, neighborhood estimates, and responsibility intervals evolve continuously through elections, pruning, and random eviction. No election outcome or peer relationship is permanent.

A5. Safety Over Liveness

The protocol prioritizes safety (preventing acceptance of incorrect state) over liveness (guaranteeing progress). Election failure, delayed convergence, or temporary unavailability are acceptable outcomes under adversarial conditions.

A6. Influence Proportional to Aligned Participation

Influence in routing, storage, and peer selection is assumed to be approximately proportional to the number of active, responsive, and storage-aligned nodes controlled by a participant over time. No mechanism is intended to allow durable amplification of influence by a minority.

These assumptions serve as invariants referenced throughout subsequent sections.

---

4. Proof-of-Work Identity and Addressing

4.1 Address Space and Identity Generation

The EC protocol network operates over a 256-bit address space:

$$A = \{0,1\}^{256}$$

Each node generates a long-lived Ed25519 public/private key pair (pk_i, sk_i), used for authenticated and encrypted communication. Node addresses are not chosen arbitrarily. Instead, a node derives its network address by performing a memory-hard proof-of-work:

$$\text{Argon2}(pk_i, s) \rightarrow a_i \in A$$

where s is a freely chosen salt and the resulting address a_i must satisfy a fixed difficulty predicate (e.g., a minimum of 20 trailing zero bits).

Argon2 parameters are selected such that finding a valid salt is expected to require approximately one day of commodity computation.

Under standard cryptographic assumptions, valid node addresses are therefore uniformly distributed over A.

4.2 Sybil Resistance and Neighborhood Targeting Cost

Because node addresses are derived through proof-of-work, identity creation is rate-limited. An adversary seeking to control a specific neighborhood (i.e., a contiguous subspace of the address space) must generate addresses whose proof-of-work outputs fall within that region.

Let:

- d = proof-of-work difficulty in trailing zero bits
- N = total number of active nodes
- k = number of nodes defining a neighborhood
- f = k/N = fractional size of the neighborhood

Each proof-of-work attempt succeeds with probability 2^{-d}. Conditioned on success, the resulting address is uniformly distributed over the address space. The probability that a successful identity falls within the target neighborhood is therefore f.

The expected number of proof-of-work attempts required to generate a single valid address within the neighborhood is:

$$E[\text{attempts}] = 2^d \times (N/k)$$

For example, with d=20, N=1,000,000, and k=100, an adversary would expect to generate approximately N/k = 10,000 valid identities before obtaining one address in the desired neighborhood, corresponding to tens of years of serial computation at the calibrated difficulty.

This mechanism does not prevent a sufficiently powerful adversary from generating many identities in parallel. Rather, it ensures that targeted identity placement incurs proportional and escalating computational cost as network size increases.

4.3 Interaction with Peer Election and Storage Alignment

Possession of a valid address does not confer influence by itself. To affect network behavior, a node must:

- Remain online and responsive
- Store aligned token-to-block mappings

- Win peer elections through proof-of-aligned storage

An adversary that generates many identities but fails to maintain aligned storage will be excluded from peer sets over time. Conversely, an adversary that successfully maintains many aligned nodes effectively contributes storage, availability, and routing capacity to the network.

Under the majority-honesty assumption, large-scale adversarial participation therefore entails either:

- Bearing the ongoing operational cost of supporting the network, or
- Accepting limited influence due to failure in peer elections

---

5. Proof-of-Aligned Storage

5.1 Role in the Combined System

Proof-of-aligned storage (PoAS) is the mechanism by which EC protocol distinguishes nodes that are (i) storing relevant token-to-block mappings and (ii) sufficiently aligned with the majority view of frequently queried state. PoAS is used in two contexts:

1. **Peer election:** nodes use PoAS responses to identify storage-aligned candidates for inclusion in their peer sets.
2. **User-level confidence:** end-users (or user-operated nodes) may issue PoAS-style queries with configurable parameters to obtain higher confidence in the current consensus mapping for a token.

PoAS is intentionally **local and transient**: it provides evidence of alignment and storage availability in a region of the address space at a particular time, not a globally complete or permanent proof.

5.2 Threats Addressed

PoAS is designed to make the following behaviors economically and operationally unattractive:

- **Free-riding:** participating in routing and elections while storing little or no usable state.
- **Spoofing:** returning responses that appear consistent without actually holding the referenced mappings.
- **Stale participation:** remaining electable while failing to keep storage aligned with frequently queried mappings.

PoAS does not attempt to prevent denial-of-service, adversarial majority manipulation, or globally coordinated history rewriting; these are addressed (or explicitly excluded) by the global assumptions in Section 3.

5.3 Challenge Construction and Response Definition

For a query targeting token t, a responding node i that stores the mapping (t → b) computes a request-bound

digest:

$$h = \text{Blake3}_{100}(t, b, pk_r)$$

where pk_r is the requester's public key and b is the responder's currently stored block mapped from t. Both t and b are included to bind the proof to the responder's claimed state.

The 100-bit output is split into 10 chunks of 10 bits:

$$h = (c_1, c_2, \ldots, c_{10})$$

Let the node's locally stored mappings be indexed and ordered by H(t'), the 256-bit hash of the token. For each chunk c_j, the node deterministically selects the *first* matching mapping encountered when scanning in token-hash order for a suffix match with c_j. Five chunks are sampled by scanning upward from H(t) and five by scanning downward.

The scan is unbounded in principle but is assumed, under normal operating conditions, to encounter a match without wrapping around the address space. This relies on the assumption that the global system maintains a sufficiently dense set of token mappings. If wrap-around occurs, the response is considered low-quality and may be rejected by verifiers; minimum storage density requirements are a deployment-time parameter.

The response is the ordered set of 10 selected mappings:

$$R_i(t) = \{(t_1, b_1), (t_2, b_2), \ldots, (t_{10}, b_{10})\}$$

5.4 Alignment Test and Clustering

Given a multiset of responses $\{R_i(t)\}$, a verifier independently recomputes the digest $h = \text{Blake3}_{100}(t, b, pk_r)$ for each response and validates the following:

- Each returned token t_j hashes to $H(t_j)$ values that are strictly ordered.
- Exactly five mappings lie above and five below H(t).
- The suffix of each $H(t_j)$ matches the corresponding 10-bit chunk c_j derived from h.

Optionally, a verifier may compute the ring-distance width between the first and last $H(t_j)$ in a response and compare it to empirically observed norms. Responses with anomalous width may be deprioritized or rejected as low-quality, but width alone is not sufficient for acceptance.

After individual validation, responses are clustered under an overlap threshold:

$$R_i \sim R_k \text{ iff } |R_i \cap R_k| \geq \theta$$

where θ is a tunable parameter (default θ = 8 out of 10).

Nodes that omit even a small number of mappings or whose local ordering diverges from the majority-aligned state are expected to produce shifted selections, causing overlap with aligned responses to degrade rapidly.

This procedure yields a probabilistic alignment test rather than a deterministic global consensus check.

5.5 Security Properties Claimed (Conditional)

Under assumptions A1–A6 (Section 3) and standard cryptographic assumptions about Blake3 and H:

P1. Non-forgeability (local): A node cannot produce a valid response $R_i(t)$ consistent with aligned nodes unless it holds the queried mapping $(t \rightarrow b)$ and sufficient surrounding mappings to satisfy the deterministic selection rule.

P2. Anti-free-riding pressure: Nodes that do not store a sufficiently complete and up-to-date local set of mappings will (in expectation) fail overlap clustering more often and therefore be selected less frequently as peers.

P3. Request binding: Inclusion of pk_r in h binds the sampling pattern to the requester, preventing precomputation of a universal response and reducing the benefit of caching a single "best" response.

P4. Tunable confidence: By repeating independent PoAS queries (varying tokens, requesters, and/or epochs) and requiring consistency across runs, a verifier can increase confidence that a dominant cluster reflects the current majority-aligned state in the relevant region.

These properties are **local** (region-scoped) and **temporal** (time-scoped). They do not constitute a global safety or finality proof.

5.6 Known Limitations and Failure Modes

PoAS explicitly does not guarantee:

- completeness of global state replication
- protection against a coordinated adversarial majority
- immunity to skewed query patterns or adversarially concentrated workloads

Potential failure modes include:

- **Correlated sampling effects:** under some storage layouts or routing biases, response overlap may be more correlated than a simple independence heuristic would suggest.
- **Selective storage strategies:** adversaries may attempt to store dense subsets that maximize electability while reducing global coverage.
- **Liveness degradation:** adversaries may reduce election settlement by refusing to respond or forward, without enabling incorrect clusters to succeed.

The protocol relies on continuous elections, churn, and overlapping coverage to mitigate these effects over time.

6. Organic Sharding and Neighborhood Semantics

6.1 Motivation and Informal Goal

EC protocol does not assume that any node maintains a complete view of all nodes or all token mappings. Instead, the system aims for *organic sharding*: nodes preferentially maintain knowledge and storage for regions of the address space near their own address, enabling locality-aware routing and localized state maintenance.

The objective is not to define a globally unique shard assignment, but to ensure that for any token address x, there exists (with high empirical frequency) a bounded set of nodes near x that both:

- can be reached efficiently by routing, and
- maintain sufficiently aligned storage for that region to answer queries and participate in elections.

6.2 Peer Gradient Maintenance

Let $a_i \in A$ denote the address of node i. Each node maintains a time-varying peer set $P_i(t)$. The peer maintenance process is designed to approximate a biased sample of the network in which peer density increases as a function of proximity to a_i under the network distance metric $d(\cdot,\cdot)$ (ring distance).

Operationally, nodes may:

- initiate elections biased toward under-represented distance bands (searching outward from a_i), and
- prune peers more aggressively in over-represented bands (random eviction and QoS-based removal).

The resulting peer distribution forms a gradient around a_i that supports DHT-style routing toward target token addresses.

6.3 Neighborhood Interval as a Local Estimate

Nodes do not know their "true" nearest neighbors in the global node set N(t). This is by design: global knowledge would collapse neighborhoods into dense, highly overlapping sets. Instead, each node defines a *local neighborhood interval* as an estimate based on its observed peers.

Let $p^-_{i,4}(t)$ be the 4th closest observed peer preceding a_i in address order, and $p^+_{i,4}(t)$ the 4th closest observed peer succeeding a_i. Define the neighborhood interval of node i at time t as:

$$I_i(t) = (p^-_{i,4}(t), p^+_{i,4}(t))$$

The parameter value 4 is used consistently throughout the protocol for neighborhood definition and vote eligibility.

Important: $I_i(t)$ is not intended to be a globally consistent partition. Intervals overlap and evolve over time due to churn, elections, and pruning.

6.4 Responsibility as Soft Coverage

A node treats tokens whose addresses fall within $I_i(t)$ as "in scope" for its local storage and alignment efforts. Because intervals overlap and are not globally coordinated, responsibility is inherently soft:

- Multiple nodes may simultaneously cover the same address region.
- Coverage can shift over time ("ownership changes") as peer sets evolve.
- Correctness relies on redundancy and continuous re-alignment rather than fixed assignment.

This design choice is intentional and aligns with the majority-based semantics of EC protocol.

6.5 Vote-Eligible Peers

For any token t, a node considers votes from the **±4 nearest peers** to t's address within its peer set. That is, votes are counted from:

- The 4 peers in the peer set with addresses closest to and less than H(t)
- The 4 peers in the peer set with addresses closest to and greater than H(t)

This bounds the vote-eligible set to at most 8 peers per token, ensuring that the +2 vote threshold represents a meaningful majority of eligible voters.

6.6 Coverage Definition

To reason about availability and alignment at the system level, it is useful to define a notion of coverage.

A token address x is said to be *r-covered* at time t if there exist at least r distinct active nodes i such that:

- $x \in I_i(t)$, and
- node i can successfully answer proof-of-aligned-storage challenges for x at time t.

Coverage is not guaranteed to be uniform across the address space at all times. Instead, the protocol relies on overlapping responsibility intervals, continuous peer elections, and churn to restore coverage in regions where it temporarily degrades.

6.7 Failure Modes

Organic sharding introduces localized failure modes:

- Under churn, $I_i(t)$ may fluctuate, temporarily reducing coverage for some regions.
- A minority adversary may bias observed peer sets locally, affecting interval estimates and election liveness.

The intended security posture is that such effects remain local and are mitigated over time by continuous elections, peer rotation, and majority-alignment pressure.

---

7. Transaction, Block, and Authorization Semantics

7.1 Tokens and Opaqueness

A *token* is a 256-bit opaque identifier. The protocol assigns no intrinsic semantics to tokens; any interpretation is external to the system. Nodes validate and propagate transactions solely based on cryptographic rules and local state, not on token meaning.

7.2 Transactions

A transaction is a cryptographically signed object containing up to six independent update entries. Each entry is a triplet of the form:

```
(t, prev, pkHash)
```

where:

- t is a token identifier
- prev is the hash of the previous block for token t (or 0 to indicate creation)
- pkHash is the hash of a public key authorized to perform the *next* update (or 0 to indicate destruction)

All entries in a transaction are validated independently but are committed atomically: a transaction is only acceptable if all contained entries are valid and reach the acceptance threshold.

7.3 Blocks

A *block* is defined as the cryptographic hash of a transaction:

```
b = H(tx)
```

Blocks are immutable identifiers and are not stored separately from transactions. For each token t, the sequence of accepted blocks referencing t forms a linear, append-only chain.

7.4 Authorization and Ownership

For each token t, the currently mapped block b_t corresponds to a transaction tx_t that established the current state. That transaction's entry for t contains a pkHash field specifying who may author the next update.

To update token t, a transaction entry must satisfy:

1. prev = b_t (extends the current chain head)
2. The transaction is signed with a keypair whose public key hash equals the pkHash stored in tx_t's entry for t

This construction binds the right to update a token to knowledge of a secret key designated by the *previous* transaction. The public key itself need not be revealed until an update is performed.

Special cases:

- *Token creation*: prev = 0. Any valid signature establishes the initial mapping. The protocol does not enforce token uniqueness; collision avoidance is delegated to higher-layer protocols (e.g., deriving token identifiers from content hashes that include creator-specific nonces).
- *Token destruction*: pkHash = 0, rendering the token permanently unupdatable.

Nodes MUST reject any transaction entry that does not extend the currently accepted block for the token or that fails signature verification against the authorized public key hash.

7.5 Immutability Rule

Nodes enforce a strict immutability rule: once a block is accepted for a token, it is never replaced or removed locally. Transactions that do not extend the locally stored chain are invalid and MUST NOT be committed.

This rule is enforced independently by each node and does not rely on majority agreement. The immutability rule takes precedence over all other considerations, including conflict resolution (Section 8.5).

---

8. Voting, Commitment, and Propagation Semantics

8.1 Scope and Design Goal

This section defines the *properties* governing how valid transactions (Section 7) are propagated, evaluated, and locally committed by nodes under partial information and concurrency. It does not prescribe a global ordering or leader-based protocol. Instead, each node independently applies the rules below to reach irreversible local conclusions.

The primary design goals are:

- Safety: committed state must be valid and immutable
- Locality: decisions are based on neighborhood-scoped information
- Fault isolation: contention and failure remain token-local

8.2 Local Commit Semantics

A node *commits* a transaction when it persistently updates its local token-to-block mappings ($t \rightarrow b$) for all entries in the transaction. Commits are irreversible: once a block is accepted as the head for a token, it is never replaced or removed.

Commit is a purely local action. Nodes do not emit explicit commit notifications. Each node must independently reach its own commit decision based on observed votes and local validation.

8.3 Vote Semantics

Votes are transient signals exchanged between peers. A vote pertains to a specific transaction and contains per-token flags for each entry in that transaction.

A vote has the structure:

```
Vote {
  transaction_hash: H(tx),
  token_flags: [(t_1, flag_1), (t_2, flag_2), ...],
  version: integer,
  voter_id: node_address,
  signature: ...
}
```

For each token t_i in the transaction, a node sets:

- **+1 flag** if:
  - The transaction entry for t_i is locally valid under Section 7
  - prev equals the node's currently stored head for t_i
  - No conflicting transaction is known (or this transaction has the highest block-id among known conflicts, see Section 8.5)
- **-1 flag** if:
  - The transaction entry conflicts with another valid entry for the same token, OR
  - The node believes the entry does not extend the currently accepted chain, OR
  - The transaction loses the block-id ordering against a known conflict

If a node lacks sufficient local state to evaluate an entry (e.g., the token is outside its responsibility region), it may set a -1 flag, but this vote will be filtered out by receivers who don't consider the voter eligible for that token (Section 8.4).

8.4 Vote Eligibility and Neighborhood Filtering

Votes are only *counted* for a token if they originate from nodes the receiver considers eligible for that token.

Eligibility is determined locally as follows:

- The voting node must be present in the receiver's current peer set
- The voting node's address must be among the receiver's ±4 nearest peers to the token's address

For a transaction touching multiple tokens ($t_1, t_2, \ldots$), votes are filtered independently per token. A single vote message may contribute to different tokens' balances through different eligibility sets.

Votes from nodes outside the locally defined ±4 neighborhood for a token are ignored for that token's commitment calculation, though they may assist with propagation.

8.5 Conflict Detection and Resolution

If two or more valid transactions reference the same prev value for a token (i.e., attempt to extend the same chain head with different blocks), they are *conflicting*.

Conflict ordering:

Conflicting transactions are ordered by block-id (highest first):

$$tx_a > tx_\beta \text{ iff block-id}(tx_a) > \text{block-id}(tx_\beta)$$

where block-id = H(tx) including all signatures, and comparison is lexicographic. This produces a stable total order over any number of conflicts.

Voting on conflicts:

When a node detects conflicting transactions, it:

1. Orders them by block-id
2. Issues +1 for the highest (if otherwise valid)
3. Issues -1 for all others
4. Includes conflict evidence in the vote message

This propagates both the conflict and the proposed resolution simultaneously.

Nodes that cannot validate any conflicting transaction (e.g., missing history for the token) vote -1 on all but SHOULD still include conflict evidence and ordering in their vote messages. This ensures conflict information propagates through the network even via nodes that cannot participate in resolution.

Commitment rule:

A node MUST NOT commit a transaction tx if a conflicting transaction tx' is known with block-id(tx') > block-id(tx), unless tx' has timed out without reaching acceptance. This prevents premature commitment to a transaction that would lose the conflict ordering.

Once the highest-block-id transaction reaches +2, it commits normally. Lower-ranked conflicts are rejected and eventually expire from mempools.

Propagation dynamics and race conditions:

In a healthy network, conflicting transactions propagate via the same viral/gossip mechanism. When both exist simultaneously:

- Both spread through overlapping peer sets
- Nodes that see both immediately apply ordering
- Votes for the higher-ranked transaction (+1) and lower-ranked (-1) propagate together
- The lower-ranked transaction cannot accumulate +2 once conflict evidence reaches its neighborhood

This creates a natural race that the higher-block-id transaction wins, provided both are propagating. For the "wrong" transaction to commit, it must reach +2 before conflict evidence arrives — requiring either network partition, adversarial suppression of the higher-ranked transaction, or the higher-ranked transaction being created after the lower one already committed.

These scenarios represent adversarial conditions or network failures, which are handled by the partition tolerance mechanism (Section 11).

Immutability constraint:

If a node has already committed a transaction before learning of a higher-ranked conflict, the committed state is retained. The immutability rule (Section 7.5) always takes precedence. This may result in temporary partition, resolved through the slow healing mechanism described in Section 11.

Security note:

The block-id ordering provides no advantage to attackers:

- *Attacker controls signing keys:* Could generate many candidate transactions and select one with a favorable block-id. But this provides no benefit — simply submitting the preferred transaction without creating a conflict achieves the same outcome.
- *Attacker doesn't control signing keys:* Cannot create valid conflicting transactions, as signatures would fail verification.
- *Attacker tricks user into double-signing:* Requires user error or compromised wallet software. The user's own signatures on both transactions constitute evidence of the double-spend attempt.

The ordering mechanism primarily benefits honest users who accidentally create conflicts (e.g., concurrent submissions from multiple devices). In such cases, the deterministic resolution ensures a clear outcome without permanent token damage.

8.6 Acceptance Threshold

For each transaction, a node maintains per-token vote balances by summing +1 and -1 flags from eligible neighbors (as determined by Section 8.4).

A transaction becomes *acceptable* when:

1. The vote balance exceeds +2 for **every** token in the transaction
2. Sufficient witness responses have been received (Section 8.7)

The +2 threshold, applied to a ±4 peer neighborhood (8 eligible voters maximum), requires a clear majority of responding peers to support the transaction. This threshold is a tunable parameter subject to empirical validation.

Commitment is atomic: either all tokens in a transaction reach threshold and commit together, or none do.

8.7 Witness-Based Propagation

In addition to neighborhood-scoped voting, each transaction is routed to a *witness point* determined by the block hash. The witness point is simply H(tx) treated as an address:

```
witness_address(tx) = H(tx)
```

Nodes select peers from their peer-set near this witness address and propagate the transaction to them, using the same gradient-based routing as any other address lookup. This provides a pseudo-random propagation path independent of the tokens involved.

Witness nodes process and propagate the transaction as normal. They may vote according to the rules above if they consider themselves eligible; otherwise, they simply relay.

Receivers require observing a minimum number of witness-originated signals (e.g., ≥2) to consider a transaction sufficiently propagated. Witness signals reduce the risk of localized suppression by ensuring transactions reach diverse regions of the network.

8.8 Vote Delivery and Versioning

Votes are delivered via **push**: nodes continuously broadcast their current vote for each pending transaction to peers in their peer set. There is no vote forwarding—a vote message is always interpreted as originating from its direct sender.

Votes are **versioned**. Each vote carries a version number, and receivers retain only the highest-versioned vote from each sender for each transaction. This handles:

- Vote updates (e.g., node learns of conflict and changes vote)
- Network reordering (late-arriving old votes are discarded)

If a node receives a vote for a transaction it hasn't seen, it requests the transaction from the voter and stores the vote pending transaction arrival.

8.9 Proxy Propagation for Distant Tokens

When a node processes a transaction touching a token t distant from its own address, it may lack direct peers in t's neighborhood. In this case, commitment information propagates through the peer gradient:

1. Node A (far from t) counts votes from its ±4 nearest peers to t
2. Those peers are closer to t and have their own ±4 peers even closer
3. Paths converge toward t's actual neighborhood

Security for distant tokens reduces to the integrity of the target neighborhood. Votes arrive via independent paths (from both sides of the address space) that converge only near the target token. An adversary would need to control the convergence point—the target neighborhood itself—to mislead distant nodes.

Known limitation: Proxy trust for distant tokens is weaker than direct neighborhood verification. Path diversity degrades in sparse network regions. This is documented as an accepted trade-off.

8.10 Handling Missing History

If a node considers itself responsible for token t but lacks the referenced prev block, it treats the transaction entry as blocked and initiates retrieval of the missing predecessor.

This process may recurse if multiple predecessors are missing. To prevent indefinite stalling, blocked transactions are subject to shorter timeouts than the transactions that reference them. If reconstruction fails within the allotted time, the node abandons the attempt and issues no +1 vote for the dependent entry.

8.11 Mempool, Timeouts, and Fairness

Transactions that are valid but not yet acceptable are stored transiently in a mempool. Each transaction is subject to a wall-clock timeout.

Upon timeout, the transaction and any associated votes are discarded. Transactions may be resubmitted later without penalty.

Fairness mechanism: To prevent flooding attacks, nodes enforce rate limits on mempool insertion. Transactions are accepted only from peers in the peer-set (or designated trusted clients), and each peer is allowed roughly equal transaction throughput. This prevents any single peer from dominating mempool capacity.

Liveness is token-local: contention or failure for one token does not impede progress for unrelated tokens.

8.12 Security Properties (Conditional)

Under assumptions A1–A6:

- **Safety:** A committed transaction must be locally valid and extend the existing chain; committed history is never rewritten.
- **Isolation:** Conflicts and contention are confined to the affected token's neighborhood.
- **Convergence (hypothesis):** Under stable neighborhoods and honest-majority participation, non-conflicting transactions are expected to be committed consistently across nodes over time.
- **Liveness:** Best-effort only; adversarial behavior, conflicts, or missing history may delay or prevent commitment.

These properties are local and temporal; no global total order or finality is claimed.

---

9. Peer Election

9.1 Peer Election Overview

Peer elections are used to construct and refresh local peer sets. Elections do not attempt to enumerate or rank nodes globally; instead, they sample *aligned storage* through randomized, query-driven interaction.

Nodes initiating an election select a random target token and collect proof-of-aligned-storage responses from the network. Responses are clustered by overlap, and only clusters exceeding a fixed threshold (e.g., 8/10 shared mappings) are considered eligible.

This mechanism induces selection pressure toward nodes whose stored mappings are consistent with the majority view of frequently queried state.

9.2 Ticket-Based Election Channels and Anti-Equivocation

Peer elections are conducted over multiple independently routed channels. For each channel, the initiating node derives a unique, opaque ticket bound to a selected entry-point. Tickets are not broadcast; they are propagated only along the routing path originating at the chosen entry-point.

Nodes may only respond to tickets they directly receive and may contribute at most one response per ticket. If multiple responses are received for the same ticket, the channel is discarded and all responses on that channel are ignored.

Because the initiating node controls entry-point selection and draws entry-points from its existing peer set, tickets are distributed across diverse network paths under the majority-honesty assumption.

Elections yield a set of responses forming one or more winning clusters under the overlap threshold. From the responses in a winning cluster, the initiating node selects exactly one peer: the responding node whose address is closest to the requested token under the network distance metric. Each election therefore admits at most one new peer.

9.3 Continuous Refresh

Nodes continuously initiate new elections and update their peer sets over time. Existing peers may be removed based on randomized eviction, quality-of-service metrics, or observed responsiveness. This ongoing churn ensures that peer sets remain dynamic and that no single election outcome is permanent.

Nodes that systematically refuse to participate in elections reduce their probability of being selected as peers and therefore exclude themselves from network influence.

---

10. Node Commit Logs, Replay Synchronization, and Accountability

10.1 Node-Local Commit Log

Each node i maintains a node-local, append-only commit log over all blocks it commits, in the order of local commit. The log is intended as a replay substrate and an accountability surface, not as a global ordering mechanism.

Formally, let $b_1, b_2, \ldots$ denote the sequence of block identifiers committed by node i. The node maintains a hash chain:

$$L_i[0] = 0$$
$$L_i[n] = H(L_i[n-1] \| b_n)$$

where $\|$ denotes concatenation. The current head of the node commit log is $L_i[n]$.

10.2 Signed Head Announcements

Nodes include their current commit-log head $L_i[n]$ in protocol responses and sign the response content. Signed head announcements provide non-repudiable evidence of the node's claimed commit-log prefix at the time of the response.

10.3 Replay and State Catch-Up

The commit log addresses bootstrap circularity: new or lagging nodes require aligned state to win elections, but winning elections is facilitated by being aligned.

Upon peer discovery, a node selects peers on both sides of its address and requests:

- current token-to-block mappings for relevant address regions, and
- commit-log heads (and, as needed, log suffixes) from multiple peers

The node then replays committed blocks from these peers' commit logs to reconstruct and/or update local state.

Replay is inherently heuristic and subject to the same majority-alignment assumptions as the rest of the system. A robust implementation is expected to:

- tail multiple peers and compare results
- prioritize prefixes that are consistent across peers
- treat inconsistent logs as degraded-quality inputs rather than definitive truth

Conflict resolution during replay: If a replaying node encounters conflicting transactions from different peers' logs, it applies the block-id ordering rule (Section 8.5) to determine which to accept. This enables gradual healing of partitioned state as new nodes join neighborhoods.

10.4 Equivocation and Evidence

Signed commit-log heads enable detection of equivocation. Two signed head announcements $(L_i[a], \sigma_a)$ and $(L_i[b], \sigma_\beta)$ from the same node i are contradictory if neither head is a prefix of the other in the hash-chain sense.

Such contradictions constitute evidence that the node has presented inconsistent commit histories to different observers. This evidence can:

- bias peer-election outcomes against the equivocating node
- be circulated among operators or users
- inform higher-level policy or governance decisions

10.5 Reaction Policy and Non-Goals

This document does not prescribe an automatic global slashing or exclusion mechanism. Node commit logs provide:

- reduced feasibility of non-repudiable lying
- support for operator and community-level responses
- bias in peer selection away from nodes exhibiting inconsistent signed behavior

Node commit logs do not define a global total order or global finality; they are local accountability and replay mechanisms.

---

11. Partition Tolerance and Fork Healing

11.1 Partition Scenarios

Despite the conflict resolution mechanism, network partitions can lead to inconsistent committed state:

Scenario: Conflicting transactions tx_a and tx_β for token t are submitted simultaneously. Due to network delays:

- Nodes in region X see only tx_a, reach +2, and commit

- Nodes in region Y see only tx_β, reach +2, and commit
- Conflict evidence arrives after both sides have committed

Under the immutability rule, neither side can uncommit. The token is now *partitioned*.

11.2 Consequences of Partition

A partitioned token exhibits degraded behavior:

- Different nodes report different chain heads for the token
- Future transactions see split state, making +2 threshold harder to reach
- The token's utility is reduced but not eliminated

11.3 Slow Healing via Churn

Partitions are not forcibly resolved. Instead, healing occurs through demographic shift:

1. New nodes join the neighborhood via peer election
2. During replay, they encounter conflicting logs from existing peers
3. They apply block-id ordering to determine which side to accept
4. Over time, the higher-block-id side accumulates more nodes
5. Eventually, the losing side becomes a minority

This healing is:

- **Probabilistic:** depends on new node arrival rate
- **Slow:** timescale is proportional to inverse churn rate
- **Unbounded:** no guarantee of completion

11.4 Design Rationale

The alternative—allowing uncommit/rewrite—would enable adversaries to:

1. Engineer a partition
2. Wait for one side to "win"
3. Force history rewrite on the losing side

The current design prevents this by making committed state sacred. The cost is slow/incomplete partition healing. The benefit is that no adversary can force history rewrite by engineering partitions.

This trade-off aligns with assumption A5 (safety over liveness).

11.5 Operational Implications

Applications built on EC protocol should:

- Monitor for partition evidence (conflicting signed heads)
- Treat partitioned tokens as degraded
- Consider partition likelihood when designing token usage patterns

Partitions are expected to be rare under honest-majority operation with reasonable network connectivity. They represent a failure mode, not normal operation.

---

12. Threat Model and Attack Analysis

12.1 Adversarial Model

We consider adversaries with the following capabilities:

- The ability to create and operate multiple nodes, subject to proof-of-work identity costs
- Full control over the behavior of adversarial nodes, including message timing, omission, and equivocation
- Partial control over network routing, including selective forwarding and eclipse attempts

We assume adversaries are computationally bounded with respect to cryptographic primitives and do not break standard hash functions or digital signatures. All guarantees are conditional on assumptions A1–A6.

The adversary's objectives may include:

- Controlling update outcomes for specific tokens
- Suppressing or delaying transactions
- Biasing peer selection and elections
- Misleading new or lagging nodes during bootstrap
- Equivocation (presenting inconsistent histories to different observers)

12.2 Sybil and Neighborhood Capture Attacks

Sybil attacks are mitigated through proof-of-work address derivation, which enforces a cost on identity creation and makes targeted address placement expensive (Section 4). To control a specific neighborhood, an adversary must generate identities whose addresses fall within that region and then maintain alignment sufficient to win local elections.

Because neighborhoods are locally defined and overlap, partial capture affects only limited regions of the address space. Under the honest-majority assumption, attempts to control large fractions of the network scale linearly in cost and operational complexity.

The protocol does not prevent a sufficiently powerful adversary from controlling a neighborhood; instead, it aims to ensure that such control is expensive, localized, and visible through degraded alignment and equivocation evidence.

12.3 Eclipse and Routing Attacks

An adversary may attempt to isolate a node by controlling its peer set or intercepting routing paths. Such eclipse attacks can bias the node's perception of neighborhoods, votes, and commit logs.

The design mitigates eclipse impact through:

- Peer-set diversity maintained by gradient sampling across the address space
- Randomized election entry points and witness-based propagation
- Reliance on multiple peers for replay and state reconstruction

Complete eclipse of a node may delay progress or mislead that node locally, but does not directly compromise the global safety properties of other nodes.

12.4 Vote Manipulation

Votes are scoped to locally defined neighborhoods and filtered by peer membership. This limits the effectiveness of vote stuffing by external nodes.

Because vote thresholds are applied to the ±4 neighborhood (8 peers maximum), adversaries cannot inflate their voting power by operating many distant nodes. An adversary must control actual neighborhood peers to influence votes.

12.5 Conflict-Based Attacks

An adversary controlling the key for a token can create conflicts by signing multiple transactions with the same prev. This is self-inflicted damage:

- The adversary's own token becomes partitioned or degraded
- The conflict resolution mechanism is deterministic, so the adversary gains nothing from creating conflicts
- Honest users' tokens are not affected (only key holders can create conflicts)

External adversaries cannot create conflicts for tokens they don't control due to authorization requirements.

12.6 Replay Poisoning and Bootstrap Attacks

During bootstrap or recovery, a node may tail commit logs from peers to reconstruct state. An adversary may attempt to provide inconsistent or misleading logs.

Mitigations include:

- Tailing multiple peers and preferring consistent prefixes
- Prioritizing peers near the node's address for replay
- Applying block-id ordering to resolve conflicting entries
- Treating inconsistent logs as degraded inputs

Bootstrap correctness is probabilistic and improves with the number and diversity of peers consulted.

12.7 Summary of Security Posture

The protocol prioritizes:

- Safety over liveness
- Locality over global agreement
- Economic and operational cost over absolute prevention

Attacks are expected to be localized, detectable, and increasingly expensive as network participation grows. No single mechanism provides complete protection; security emerges from the composition of identity costs, local alignment pressure, continuous churn, and accountability through signed state.

---

13. Comparison to Other Systems

13.1 Classical BFT Protocols

Classical Byzantine Fault Tolerant (BFT) protocols such as PBFT assume a fixed, known validator set and provide strong safety and liveness guarantees under bounded Byzantine faults (typically $f < n/3$). These systems rely on explicit quorum formation and global agreement rounds.

Advantages of classical BFT:

- Strong safety guarantees (no conflicting decisions)
- Deterministic finality once agreement is reached
- Well-understood theoretical foundations

Limitations relative to EC protocol:

- Poor scalability due to all-to-all communication
- Global failure modes (a single stalled round halts progress)
- Limited applicability in permissionless or high-churn environments

By contrast, EC protocol intentionally abandons deterministic finality in favor of localized interaction and continuous operation under churn. It does not attempt to solve the same problem as classical BFT and should not be evaluated under the same fault thresholds.

13.2 Nakamoto-Style Proof-of-Work Blockchains

Proof-of-work (PoW) blockchains provide probabilistic consensus over an append-only ledger, assuming that a majority of computational power is controlled by honest participants.

Advantages:

- Permissionless participation
- Open membership under a majority-hashpower assumption
- Global consensus without explicit identity management

Limitations relative to EC protocol:

- Global mempools and block production create systemic bottlenecks
- High latency to reach practical finality
- Inefficient use of computation and energy
- Limited fault isolation (hotspots and attacks affect the entire network)

Like PoW systems, EC protocol relies on a majority-honesty assumption. However, EC protocol replaces global serialization with locality and organic sharding, trading irreversible history for improved adaptability and isolation of failure.

13.3 Proof-of-Stake Systems

Proof-of-stake (PoS) systems weight influence according to stake and often introduce governance layers.

Advantages:

- Lower energy consumption than PoW
- Faster finality via committee-based consensus
- Explicit mechanisms for protocol upgrades

Limitations relative to EC protocol:

- Centralization pressure toward large stakeholders
- Governance complexity and social-layer coordination
- Global validator sets and epochs remain coordination points

While PoS systems formalize majority control through stake and governance, EC protocol encodes majority control directly into its consensus semantics without privileged roles, epochs, or validator classes. All nodes remain symmetric, and influence emerges from participation rather than stake.

13.4 DHT-Based Systems

Distributed hash tables (DHTs) such as Kademlia provide scalable routing and data location but typically rely on weak consistency models and are vulnerable to Sybil and eclipse attacks.

EC protocol combines DHT-style routing with cryptographic alignment pressure, proof-of-work identity costs, and continuous peer re-election. Unlike traditional DHTs, responsibility is not static, and alignment is enforced through proof-of-aligned storage and voting.

13.5 Summary of Trade-offs

EC protocol occupies a distinct point in the design space:

- It prioritizes *continuous majority alignment* over immutable history
- It replaces global agreement rounds with local sampling and alignment pressure
- It offers probabilistic, temporal correctness rather than deterministic safety

These choices make EC protocol unsuitable for applications requiring strong finality or minority protection, but potentially well-suited for large-scale, permissionless systems where adaptability, fault isolation, and openness are primary concerns.

---

14. Emergent Properties and Scaling Hypotheses

14.1 Scaling Assumptions

Let N denote the number of active nodes in the network. Node identities are permissionless but rate-limited by memory-hard proof-of-work, constraining the rate at which any single adversary can introduce new identities.

Neighborhoods are formed through randomized, query-driven peer selection. While neighborhood size may grow over time due to churn and increased participation, it does so sublinearly with respect to N.

14.2 Emergent Security With Network Growth

The security thesis of EC protocol is not based on absolute guarantees, but on scaling pressure:

- As N increases, the cost of acquiring sufficient identities to dominate a large number of independently formed neighborhoods grows superlinearly relative to the influence gained within any single neighborhood.
- Organic sharding limits the blast radius of compromised nodes to local regions of the address space.

- Continuous churn and re-sampling erode long-term adversarial positioning.

While individual neighborhoods may be compromised, global compromise requires sustained control across a large number of independently formed neighborhoods, which is assumed to be infeasible under the identity cost and churn model.

14.3 Throughput Scaling

Transaction processing is localized to neighborhoods associated with the affected state. For transactions touching a bounded number of tokens, the expected per-node processing load grows slowly relative to N, while the number of concurrently processable transactions increases with the number of disjoint neighborhoods.

System throughput increases approximately linearly with network size until localized contention limits are reached. This scaling is achieved without introducing global coordination points.

This claim is intentionally limited to *expected behavior under bounded skew* and does not apply to adversarially concentrated workloads.

14.4 Implications

If validated, these emergent properties imply that EC protocol becomes *more secure and more useful* as participation increases:

- Security improves through increased difficulty of coordinated neighborhood capture
- Throughput increases through parallelism across organic shards
- The network becomes suitable for progressively higher-value use cases as stability and alignment strengthen

These claims are hypotheses subject to empirical validation and adversarial testing.

---

15. Failure Modes and Non-Goals

EC protocol explicitly does not guarantee:

- **Immutability of historical state:** Consensus reflects current majority alignment, not permanent record
- **Protection of minority views:** A coordinated majority can override any position
- **Byzantine safety under adversarial majority:** The network may converge on incorrect state if majority is adversarial
- **Liveness under adversarial conditions:** Progress may stall for individual tokens
- **Global total ordering:** No single ordering of all transactions exists

Under adversarial majority control, the network may converge on an internally consistent but externally incorrect state. This outcome is considered unavoidable in permissionless systems without trusted anchors.

---

16. Known Limitations

This section consolidates acknowledged weaknesses in the protocol design:

16.1 Proxy Trust for Distant Tokens

Verification of distant tokens relies on multi-path convergence through the peer gradient. While this provides meaningful security through path diversity, it is weaker than direct neighborhood verification. Path diversity degrades in sparse network regions.

16.2 Partition Healing Timescale

When partitions occur, healing depends on new nodes joining and applying block-id ordering during replay. The timescale is:

- Proportional to inverse churn rate
- Unbounded in worst case
- Not formally analyzed

Low-churn networks may sustain partitions for extended periods.

16.3 Conflict Resolution Determinism

The block-id ordering for conflict resolution is deterministic and pre-computable by key holders. This is not a vulnerability: an attacker who controls signing keys gains nothing from creating conflicts they could avoid — simply submitting the preferred transaction achieves the same outcome. The mechanism resolves conflicts deterministically for the benefit of honest users who accidentally create conflicts; it does not protect tokens from malicious key holders.

16.4 Bootstrap Trust

New nodes must trust their initial peer sample. An adversary controlling a significant fraction of the network during a node's bootstrap can mislead that node. The node has no cryptographic guarantee of correct initial state —only probabilistic confidence improving with peer diversity.

---

17. Design Rationale and Economic Model

This section explains the economic assumptions and design philosophy underlying EC protocol, providing context for why certain trade-offs were made.

17.1 Tokens Are Not Precious

Unlike systems where tokens represent scarce digital assets (Bitcoin, Ethereum), EC protocol treats tokens as **cheap, disposable identifiers**. Key differences:

Scarce token model	EC token model
Tokens are mined/limited	Tokens are freely created
Losing a token is catastrophic	Losing a token is inconvenient
Security = protect the token itself	Security = trust the issuer/application
Finality is essential	Settlement is sufficient
Token = value	Token = pointer to value

This design choice has profound implications:

- **No artificial scarcity:** Token creation has no protocol-level cost beyond transaction processing
- **Application-defined semantics:** Tokens mean whatever the application layer decides
- **Reduced attack incentive:** No "digital gold" to steal at the protocol layer
- **Issuer-based trust:** Value guarantees come from issuers, not the protocol

17.2 The Issuer-Redemption Pattern

The primary pattern for value transfer in EC protocol:

1. Issuer creates token T: (T, prev=0, pkHash=H(issuer_pk))
2. Issuer transfers to user: (T, prev=b_1, pkHash=H(user_pk))
3. User transfers to merchant: (T, prev=b_2, pkHash=H(merchant_pk))
4. Merchant redeems with issuer: (T, prev=b_3, pkHash=H(issuer_pk))
5. Issuer honors redemption out-of-band

Properties:

- Only issuer can create tokens (they hold initial key)
- Transfer chain is cryptographically signed
- Redemption requires issuer cooperation
- Double-spend → issuer sees conflict → refuses both redemptions
- Network partition → issuer sees inconsistent responses → investigates

This pattern doesn't require global finality because **the issuer is the trust anchor**. EC protocol provides the transfer mechanism; the issuer provides the value guarantee.

Examples:

- Bank issues payment tokens, redeems for account credit
- Event organizer issues tickets, redeems for venue entry
- Game publisher issues items, redeems for in-game assets
- Certificate authority issues credentials, verifies for authentication

17.3 Tiered Confidence Model

Applications should choose confirmation levels appropriate to their risk tolerance:

Level	Meaning	Typical time	Use case
Seen	Transaction in mempool, propagating	~100ms	Optimistic UI updates
Local +2	Neighborhood accepts	~1-5s	Low-value transfers, gaming
Witnessed	Cross-network propagation confirmed	~5-10s	Medium-value commerce
Time-stable	Consistent state observed over interval	~minutes	High-value transactions
Issuer-confirmed	Issuer explicitly acknowledges	varies	Redemption, settlement

User-side verification:

Users (or their applications) can verify state using the same election mechanism as nodes. A user queries multiple nodes via randomized channels and clusters responses by overlap, exactly as described in Section 9. The user can tune:

- Number of channels (more channels → higher confidence)
- Required cluster size (larger cluster → stronger majority evidence)
- Overlap threshold (stricter matching → tighter alignment requirement)

This provides tunable confidence in the current state reading.

Handling conflicting views:

If the election produces ambiguous results:

- *Slow settlement:* Extend the election with more channels; if a clear majority emerges, the state is settling normally

- *Weak majority:* Assess cluster sizes; a dominant cluster with small minority indicates likely convergence
- *Split-brain:* Two or more comparably-sized clusters indicate genuine partition or active conflict; application should wait or escalate

The appropriate level depends on:

- Transaction value
- Trust in counterparty
- Reversibility of real-world action
- Issuer's dispute resolution policy

17.4 Participation Incentives

EC protocol uses structural incentives rather than explicit rewards:

Write access requires contribution:

- Only connected nodes can submit transactions
- Node identity requires PoW investment
- Peer election requires PoAS (proving aligned storage)
- PoAS proves neighborhood storage, not just own data

This creates bundled incentives:

> Want to write → Must run node → Must store neighborhood → Contributes to others

Read access is open but prioritized:

- Any node can query
- Non-peer queries receive referral responses (indirect)
- Peer queries receive direct responses (faster)
- Running a node provides better read performance

Fair-balance prevents free-riding:

- Transaction throughput is rate-limited per peer
- No single node can dominate transaction submission
- Sustained participation required for sustained access

17.5 Why Takeover is Self-Defeating

Consider an adversary attempting silent network takeover:

Phase 1: Identity accumulation

- Adversary generates many PoW identities
- Cost: O(N) computation to match network size

Phase 2: Election dominance

- Adversary nodes win peer elections
- Honest nodes increasingly excluded from peer sets

Phase 3: Value destruction

- Honest nodes lose write access
- Honest nodes notice degraded service
- Honest users abandon network
- Network value collapses

Adversary's dilemma:

```
U(adversary) = V(control) × NetworkValue(t) - Cost(attack)
```

As adversary gains control, NetworkValue decreases. Complete control yields complete value destruction. The optimal extraction point may exist, but the attack is visible throughout — honest participants can detect and respond.

Contrast with "burn it down" adversary: A purely destructive adversary (nation-state, ideological attacker) is not deterred by economic arguments. Mitigations are social/political:

- Geographic/jurisdictional distribution
- No concentrated high-value targets
- Visible attribution of attack

17.6 Application Example: Stateless Gift Cards

This example illustrates how EC's design philosophy enables applications impossible or impractical on other platforms.

Scenario: A mall consortium wants to offer gift cards accepted at any member shop, without shared infrastructure.

Issuance:

1. Mall authority creates root document: {issuer: "MallAuth", value: 100, sig: ...}
2. Customer computes t_0 = SHA(root_document)
3. Customer registers t_0 in EC with their public key

Spending ($10 at Shop A):

1. Customer generates a random SALT for this split
2. Customer creates two child documents (each contains root + split history):
   - doc_1: {root: ..., splits: [{value: 10, salt: SALT}]}
   - doc_2: {root: ..., splits: [{value: 90, salt: SALT}]}
3. Customer computes token IDs:
   - t_1 = SHA(t_0 || 10 || SALT)
   - t_2 = SHA(t_0 || 90 || SALT)
4. Customer submits atomic EC transaction:
   - Destroy t_0
   - Create t_1, owner = Shop A's public key
   - Create t_2, owner = customer
5. Customer presents doc_1 to Shop A

Key insight: Using the same SALT for both children, the validator can reconstruct the sibling token ID without seeing the sibling document. Given doc_1 showing value=10, SALT, and parent value=100:

- Compute t_1 = SHA(t_0 || 10 || SALT) ✓ (matches presented token)
- Compute t_2 = SHA(t_0 || 90 || SALT) (sibling must exist)
- Verify t_0 was destroyed in the same transaction that created t_1 and t_2

No need to see doc_2 or know who holds t_2.

Shop verification (stateless):

1. Verify root document signature (issuer authenticity)
2. Replay split chain: recompute each token ID from parent + value + SALT
3. Verify final token t_1 exists in EC, owned by shop's key

4. Verify all ancestor tokens are destroyed
5. Submit EC transaction: destroy t_1
6. Provide goods

Second redemption attempt fails:

1. Customer presents same t_1 to Shop B
2. Shop B queries EC → t_1 is destroyed (pkHash = 0)
3. Reject

Properties achieved:

Property	How
No shared database	EC is the only shared state
No shop-side state	Destroy on redeem; EC tracks used/unused
Multi-shop acceptance	Shared issuer key, independent verification
Self-certifying chain	Same SALT enables sibling verification
No breach risk	No database of balances to steal
Atomic splitting	6-entry transaction ensures conservation
Offline documents	Customer holds value proofs; EC holds ownership

Minimal shop requirements:

- Keypair (published)
- EC query access
- EC write access (destroy on redeem)

A shop could implement acceptance as a single-page webapp with no backend. The mall consortium needs only to agree on the issuer signing key — no shared infrastructure, no reconciliation, no integration.

Extensions:

- *Partial redemption:* Shop creates change token back to customer
- *Transfer:* Customer updates token owner to friend's key, hands over documents
- *Expiration:* Include expiry in root document; validators reject if expired

- *Refunds:* Shop issues fresh token from their allocation

This pattern — documents carry value semantics, EC carries ownership — generalizes beyond gift cards to tickets, credentials, vouchers, and any bearer instrument.

17.7 Token Scattering as Security Property

The create-destroy-create pattern common in EC applications provides a natural defense against neighborhood capture attacks.

Observation: When a token is destroyed and new tokens created, the new token addresses are:

```
t_new = SHA(new_document || SALT)
```

These addresses are effectively random in the 256-bit space, independent of the original token's address.

Security implication:

Consider an adversary who invests resources to capture neighborhood N (controlling majority of peers near some address range):

1. Adversary controls tokens currently in N
2. User spends/splits a token $t \in N$
3. Transaction destroys t, creates t', t'' with random addresses
4. t', t'' land in neighborhoods N', N'' (overwhelming probability: $N' \neq N$, $N'' \neq N$)
5. Adversary's captured neighborhood now contains one fewer valuable token

The adversary faces a dilemma:

- **Static attack:** Capture one neighborhood → tokens flow away → diminishing returns
- **Follow tokens:** Must capture new neighborhoods → cost scales with transaction volume
- **Capture everything:** Cost scales with network size (Section D.2)

Comparison to account-based systems:

System	Token/Account location	Capture persistence
Bitcoin	Fixed addresses	Captured addresses stay captured
Ethereum	Fixed accounts	Captured accounts stay captured
EC protocol	Random per operation	Tokens scatter away from captured regions

In EC, active tokens continuously redistribute across the address space. A captured neighborhood is like a hole in a net — the fish swim through.

Quantification:

If tokens are split/spent with average lifetime T, and adversary captures neighborhood covering fraction f of address space:

> E[tokens remaining in captured region after one operation] = f

For f = 0.001 (0.1% of address space, an expensive capture):

- After 1 operation: 0.1% chance token stays
- After 3 operations: 0.0001% chance token stays

Active usage naturally evacuates value from any captured region.

17.8 Comparison to Honest-Majority Assumptions Elsewhere

EC protocol's honest-majority assumption is often criticized, but it's universal:

System	Assumption	Consequence of violation
Bitcoin	>50% hashpower honest	History rewrite (has happened to forks)
Ethereum PoS	>67% stake honest	Finality failure
PBFT variants	>67% validators honest	Safety/liveness failure
EC protocol	>50% nodes honest	Consensus drift

EC is more *honest* about this assumption, not more *vulnerable* to it. The difference is that EC's failure mode (gradual drift) may be more recoverable than others (immediate rewrite or halt).

17.9 Further Applications

The gift card example (Section 17.6) illustrates the issuer-redemption pattern with token splitting. Other applications follow similar or distinct patterns.

Naming and DNS

A name service maps human-readable identifiers to network addresses. In EC:

- Token t = H("example.com")

- Token content: IPv4/6 address + port + SALT fits in 32 bytes (token size)
- No off-chain document needed — endpoint data is in the transaction itself

Resolution: Query EC for H("example.com"), read endpoint from transaction content. Owner updates by submitting new transaction with updated endpoint.

This follows a simpler pattern than gift cards: no document chain, just direct token → endpoint mapping.

Decentralized Identity

Social identity where the user controls portability:

- Token t = H("@alice")
- Token content: CDN pointer, content hash, or public key
- Switching platforms = updating the pointer, not migrating data

Applications verify identity by checking token ownership. The user's key can sign content directly, providing cryptographic proof of authorship without platform intermediation.

PKI Equivalence

Naming and identity together provide a decentralized Public Key Infrastructure:

- Name → public key binding via token ownership
- Resolution: query H("example.com") → retrieve owner's public key
- No certificate authority required; the token *is* the certificate

Trust establishment via co-signing:

- Transaction entries can require multiple signatures
- A name registration co-signed by multiple known entities establishes trust
- Example: H("example.com") owned by key K, transaction co-signed by K and well-known auditor keys
- Verifiers check co-signatures to assess trustworthiness

This enables flexible trust models — from self-sovereign (single key, no co-signers) to highly audited (multiple institutional co-signers) — without protocol changes.

Mitigating Neighborhood Capture for Static Tokens

Unlike gift cards (which scatter via splits), naming and identity tokens are static and long-lived, making them potential targets for neighborhood capture attacks.

Mitigation via redundant tokens:

- Instead of single token H("example.com"), register multiple:
  - t_0 = H("0/example.com")
  - t_1 = H("1/example.com")
  - ...
  - t_k = H("k/example.com")

These land in k+1 different, uncorrelated neighborhoods. An attacker must capture multiple neighborhoods simultaneously. Readers verify that all tokens are controlled by the same key — any inconsistency indicates attack or misconfiguration.

The security level scales with k. For high-value names, k = 10 provides meaningful protection; for most names, k = 2-3 suffices.

Resolution Proxies

For naming and identity, resolution speed matters. Organizations may operate proxy services:

- Offer HTTPS endpoint (easier than direct EC queries for some clients)
- Cache hot tokens with recent election evidence
- Serve cached responses with proof of freshness

Token popularity likely follows a power-law distribution. Proxy caching strategy:

- On cache miss: stall client, fetch from EC, probabilistically cache
- On cache hit: serve with election evidence
- On expiry: probabilistically refresh (hot tokens refresh often, cold tokens evict)

Popular tokens naturally accumulate more refresh opportunities; cold tokens are evicted. This provides efficient resolution while preserving decentralization — any proxy can be verified against EC directly.

Voting and Verifiable Elections

This pattern differs fundamentally from the examples above, demonstrating batch commitment with public verification.

Setup phase:

1. Election authority generates seed S
2. Publishes commitment H(S)
3. Derives token sequence: $t_i = H(S \| i)$ for i = 1..N
4. Distributes tokens to eligible voters via blinding scheme (authority cannot link voter ↔ token)

Voting phase:

1. Each voter receives one token t_i (doesn't know i)
2. Voter transfers t_i to published candidate key: $(t_i, \text{prev=current}, \text{pkHash=H(candidate\_pk)})$
3. Transfer is anonymous (token is unlinkable to voter)

Verification phase:

1. Authority publishes seed S
2. Anyone regenerates full token set from S
3. Anyone queries EC for each token's final owner
4. Anyone tallies: count tokens owned by each candidate key

Properties:

Property	**Mechanism**
Eligibility	Only authority-generated tokens exist
One-person-one-vote	Each token transferable once
Ballot secrecy	Blinding prevents voter↔token linkage
Public verifiability	Seed publication enables full audit
No central counter	Anyone can independently tally
Coercion resistance	Voter can transfer again before close (optional)

This demonstrates EC's suitability for batch operations with cryptographic commitment, where millions of cheap, single-use tokens enable transparent verification without sacrificing voter privacy.

---

18. Summary

EC protocol encodes a deliberate design choice:

- Consensus is defined by majority alignment, not immutable history
- Faults and attacks are localized through organic sharding
- Correctness is probabilistic and temporal, not absolute
- Safety is prioritized over liveness

- Committed state is immutable locally, even at the cost of partition persistence
- Tokens are cheap identifiers, not precious assets — value semantics live in application layer
- Create-destroy-create patterns scatter tokens across neighborhoods, defeating targeted attacks

This model trades strong finality for resilience, openness, and continuous operation, and is intended for use cases where *current shared agreement* matters more than permanent historical record.

Appendix A: Hash Function Usage

The protocol standardizes on Blake3 for all internal hashing due to its speed and security properties:

Notation	Function	Output	Usage
$H(\cdot)$	Blake3	256 bits	Token addressing, block hashes, pkHash, general-purpose
$\text{Blake3}_{100}(\cdot)$	Blake3 truncated	100 bits	PoAS challenge construction
$\text{Argon2}(\cdot)$	Argon2id	256 bits	PoW address derivation (memory-hard)

Note on token creation: Users may derive token identifiers using any hash function of their choice (SHA-256, Blake3, application-specific schemes). The protocol treats tokens as opaque 256-bit identifiers. The $H(\cdot)$ notation above refers only to protocol-internal operations such as computing token addresses for routing.

All comparisons of hash outputs are lexicographic over the byte representation.

Appendix B: Parameter Summary

Parameter	Description	Default	Notes
d	PoW difficulty (trailing zero bits)	20	~1 day per identity
±4	Neighborhood size (peers each direction)	4	Used for intervals and vote eligibility
θ	PoAS overlap threshold	8/10	Minimum shared mappings for clustering
Vote threshold	Per-token acceptance balance	+2	Applied to ±4 peer neighborhood
Witness minimum	Required witness signals	2	For propagation confidence
Transaction timeout	Mempool expiry	TBD	Deployment parameter

Appendix C: Document Revision History

Version	Date	Changes
Draft 0.1	—	Initial draft
Draft 0.2	—	Consolidated duplicate sections; corrected authorization semantics; added conflict resolution; clarified vote structure and filtering; added partition healing section; documented known limitations
Draft 0.3	—	Renamed from "ecRust" to "EC protocol" (Echo Consent); simplified witness mechanism; revised conflict resolution security analysis
Draft 0.4	—	Converted math notation to Unicode; fixed neighborhood parameter consistency
Draft 0.5	—	Replaced pairwise hash-battle with stable block-id ordering for conflict resolution; added Section 17 (Design Rationale and Economic Model) covering token philosophy, issuer pattern, tiered confidence, and participation incentives; clarified propagation dynamics for conflict races
Draft 0.6	—	Added Appendix D (Formal Analysis) with proofs for vote threshold, capture cost, and PoAS alignment
Draft 0.7	—	Added Sections 17.6-17.7: Stateless Gift Cards application example and Token Scattering security analysis
Draft 0.8	—	Added Section 17.9 (Further Applications: naming, identity, voting)
Draft 0.9	—	Clarified user-side elections in 17.3; fixed SALT mechanism in 17.6 (same SALT enables sibling verification); updated 17.9 with inline endpoint data for DNS, redundant tokens for static token security, and resolution proxy pattern
Draft 0.10	—	Standardized all protocol hashing on Blake3 (Appendix A); added PKI equivalence note with co-signing for trust establishment (Section 17.9)
Draft 0.11	—	Added abstract
Draft 0.12	—	Added author, year, and project link

Appendix D: Formal Analysis

This appendix provides formal analysis of key protocol mechanisms, establishing quantitative bounds on security properties.

D.1 Vote Threshold Analysis

Model: For a token t, votes are collected from the ±4 nearest peers (up to 8 voters). Each voter independently supports the transaction with probability p, where p reflects the fraction of the true neighborhood that considers the transaction valid.

Let $X \sim \text{Binomial}(8, p)$ be the number of +1 votes. The vote balance is:

$$\text{balance} = X - (8 - X) = 2X - 8$$

The acceptance threshold (+2) requires balance ≥ 2, equivalently $X \geq 5$.

Proposition D.1 (Threshold Probability):

$$P(\text{balance} \geq +2 \mid p) = \sum_{k=5}^{8} C(8,k) \times p^k \times (1-p)^{8-k}$$

Computed values:

True support p	P(balance ≥ +2)	Interpretation
0.50	0.363	Split neighborhood
0.60	0.594	Slight majority
0.70	0.806	Clear majority
0.80	0.944	Strong consensus
0.90	0.995	Near-unanimous

Proposition D.2 (Bayesian Inference):

Given observed balance ≥ +2 and uniform prior on p:

$$P(p > 0.5 \mid \text{balance} \geq +2) \approx 0.91$$

This provides 91% confidence that the true neighborhood majority supports the transaction when the threshold is reached.

Corollary D.3 (Multi-Token Protection):

For transactions touching k independent tokens, each with split neighborhoods ($p = 0.5$):

$$P(\text{all tokens reach } +2) = 0.363^k$$

Tokens k	P(all reach +2)
1	0.363
2	0.132
3	0.048
4	0.017
6	0.002

Multi-token transactions have exponentially decreasing false-positive rates, providing natural protection against split-neighborhood errors.

D.2 Neighborhood Capture Cost

Model: An adversary seeks to control a majority of a target neighborhood of size k within a network of N nodes. Node addresses are derived via PoW with difficulty d (expected time per identity: 2^d attempts, calibrated to approximately 1 day).

Each successful identity has address uniformly distributed over the address space. The probability of landing in the target neighborhood is k/N.

Proposition D.4 (Capture Cost):

To place m adversarial identities in a target neighborhood:

$$E[\text{total identities generated}] = m \times N/k$$
$$E[\text{PoW attempts}] = m \times 2^d \times N/k$$

For majority control: $m = \lfloor k/2 \rfloor + 1 = 26$ for $k = 50$.

Computed Values (d = 20, k = 50):

Network N	Identities needed	Serial time	Parallel (1000 workers)
1,000,000	520,000	1,424 years	1.4 years
10,000,000	5,200,000	14,237 years	14.2 years
100,000,000	52,000,000	142,368 years	142 years

Corollary D.5 (Linear Scaling):

Capture cost scales linearly with network size N. A 10× increase in network participation requires 10× the adversarial investment for equivalent neighborhood control.

D.3 PoAS Alignment Analysis

Model: Two nodes with storage alignment α (fraction of mappings they agree on) perform the PoAS challenge walk. Starting from the query token, each walks 5 steps up and 5 steps down, selecting tokens by suffix matching.

Pessimistic Model (No Re-convergence):

If the nodes diverge at any step (encounter a mapping one has and the other lacks), they remain diverged for subsequent steps.

$$P(\text{aligned at step } j) = \alpha^{(j-1)}$$

Expected matches in one direction:

$$E[\text{matches}] = 1 + \alpha + \alpha^2 + \alpha^3 + \alpha^4 = (1 - \alpha^5)/(1 - \alpha)$$

Proposition D.6 (Match Distribution):

Under the pessimistic model with independent directions:

$$P(\geq 8/10 \text{ matches} \mid \alpha) = \Sigma \text{ over valid } (k_1,k_2) \text{ of } P(k_1 \text{ up}) \times P(k_2 \text{ down})$$

Computed values:

Alignment α	P(≥8/10)	P(=10/10)
0.99	0.960	0.923
0.95	0.809	0.663
0.90	0.638	0.430
0.80	0.367	0.168
0.70	0.188	0.058
0.50	0.031	0.004

Empirical Validation:

Simulation of the PoAS mechanism with varying storage density confirms the analytical model:

Density	Simulated P(≥8/10)	Analytical P(≥8/10)
99%	0.94	0.96
95%	0.74	0.81
90%	0.47	0.64
80%	0.34	0.37
70%	0.15	0.19

The analytical model provides a reasonable approximation, with the simulation showing slightly lower match rates due to boundary effects and walk structure.

Proposition D.7 (Alignment Inference):

Given observed ≥8/10 overlap between two PoAS responses, with uniform prior on α:

$$P(\alpha > 0.7 \mid \text{overlap} \geq 8) \approx 0.88$$
$$P(\alpha > 0.5 \mid \text{overlap} \geq 8) \approx 0.99$$

This provides high confidence that clustered responses come from well-aligned nodes.

Corollary D.8 (Threshold Selection):

The overlap threshold $\theta = 8$ provides effective discrimination:

Threshold θ	FPR (α=0.5)	TPR (α=0.9)	Ratio
6	0.188	0.919	4.9
7	0.078	0.768	9.8
8	0.031	0.638	20.4
9	0.012	0.526	44.9

At $\theta = 8$, a misaligned node ($\alpha = 0.5$) passes only 3.1% of the time, while a well-aligned node ($\alpha = 0.9$) passes 63.8% of the time — a 20× discrimination ratio.

D.4 Convergence Analysis

Model: A valid, non-conflicting transaction tx propagates through a connected network graph with diameter D and per-hop delay bounded by δ. Honest nodes form a majority in each ±4 neighborhood.

Proposition D.9 (Propagation Bound):

Under the gossip propagation model:

1. tx reaches all nodes within time $D \times \delta$
2. All honest nodes vote +1 for valid tx
3. Vote propagation completes within additional $D \times \delta$ time

Total time to convergence: $O(D \times \delta)$

For typical gossip networks, $D \sim O(\log N)$, giving convergence in $O(\log N \times \delta)$.

Proposition D.10 (Amplification Cascade):

Once a node commits tx:

1. It responds to subsequent vote requests with confirmation
2. This accelerates balance accumulation for uncommitted nodes
3. Creates cascade: first commits → neighbors commit → their neighbors → ...

The cascade completes in O(D) additional rounds after first commitment.

Proposition D.11 (Invalid Transaction Rejection):

For invalid tx' (fails local validation), honest nodes vote -1. Under honest-majority neighborhoods:

E[balance for tx'] < 0

P(tx' reaches +2) < 0.363 (from Proposition D.1 with p < 0.5)

Combined with multi-token protection (Corollary D.3), invalid transactions are rejected with high probability.

Proposition D.12 (Partition Behavior):

If network partitions into components $C_1, C_2, ...$:

1. Each component converges independently
2. Honest majority in component → converges to valid state
3. Reconnection → conflict detection → healing via block-id ordering

Global convergence requires global connectivity or eventual reconnection with healing.