Proof-of-Stake Consensus in the QuantumCoin Blockchain

Authors: The QuantumCoin Community

Revision 6 — April 2026

Disclaimer

QuantumCoin is a community driven project. All visions and projects are aspirational. There is no value attributed to anything. All projects are community driven and there is no guarantee of delivery. QuantumCoin is not intended to be, or to be the subject of, an investment opportunity, investment contract, or security of any type.

Scope. This paper describes the intended design of the QuantumCoin consensus protocol. All properties stated herein (safety, liveness, finality) are properties of the protocol design. Implementation defects, software bugs, or operational issues are outside the scope of this paper.

Abstract

We present the QuantumCoin consensus protocol, a deposit-weighted BFT protocol derived from the PBFT family that provides **immediate deterministic finality** for every block. The protocol introduces five mechanisms that, to the authors' knowledge, are novel among deployed proof-of-stake systems: (1) **committed-history randomness**, which derives consensus entropy from on-chain data committed 512,000 blocks in the past, eliminating both last-revealer bias and grindable-abort vulnerabilities; (2) **protocol-enforced round-robin transaction execution ordering**, which algorithmically separates transaction selection from execution order, making sandwich attacks probabilistically infeasible (Proposition 7); (3) a **three-pass bounded-committee selection algorithm** that compositionally combines deposit-ranked, coin-flip, and pseudorandom passes to guarantee both high deposit representation and small-validator inclusion; (4) **deposit normalization**, which dynamically clips any validator's effective deposit exceeding 10% of total eligible deposit and redistributes the clipped excess voting weight to validators with clean participation records (no actual coins are transferred); and (5) a **graduated offline penalty system** combining fixed deposit deductions, progressive voting weight reduction, exponential proposer deferral, and permanent exclusion — with penalties attributed only when individual proposer failure is distinguishable from network-wide issues.

We prove safety (Agreement, Theorem 1) and liveness (Termination, Theorem 3) under the standard partial synchrony model [5], with safety holding even during asynchronous periods (Theorem 2). We provide heuristic concentration bounds on per-block committee composition (Proposition 4; see methodological note therein), a normalization amplification bound with an observable sufficient condition for safety (Proposition 2, Corollary 2), and formal verification of bounded configurations via TLA+ model checking.

The protocol selects a per-block committee of up to 128 validators from an unbounded registered set; committee membership rotates at every block height. All phase transitions require a two-thirds supermajority quorum weighted by deposit. The protocol tolerates Byzantine faults comprising strictly less than one-third of the committee's post-normalization effective deposit. Graduated offline penalties, deposit normalization with redistribution, and a 32,000-block withdrawal delay complete the economic mechanism design.

1. Introduction

QuantumCoin is a quantum-resistant blockchain that uses proof-of-stake consensus for block production and finalization. The post-quantum cryptographic primitives (signature schemes, hash functions) are outside the scope of this paper; this paper addresses only the consensus protocol. The consensus protocol is cryptographically agile and does not depend on specific signature schemes; its quantum resistance derives from the hybrid post-quantum signature algorithms used for message authentication, which are documented separately. The consensus protocol described in this paper is a variant of PBFT [1] adapted for an open-registration proof-of-stake setting with deposit-weighted voting.

The protocol provides **immediate deterministic finality**: once a block is finalized, it cannot be reverted by design. There are no probabilistic confirmation periods and no fork-choice rules. Every finalized block represents an irreversible commitment by a supermajority of deposit.

Contributions. Beyond the standard PBFT-derived BFT consensus, this paper makes the following contributions:

- A **committed-history randomness** mechanism (Section 10.1) that derives consensus entropy entirely from on-chain data committed 512,000 blocks in the past, eliminating both last-revealer bias and grindable-abort vulnerabilities without requiring VRFs or VDFs.
- A **protocol-enforced round-robin transaction execution ordering** (Section 13) that algorithmically separates transaction selection from execution order, with a formal sandwich attack probability bound (Proposition 7).
- A **three-pass bounded-committee selection algorithm** (Section 10.2) that compositionally combines deposit-ranked, coin-flip, and pseudorandom passes to guarantee both high deposit representation and small-validator inclusion within a bounded committee of up to 128 validators.
- Formal proofs of **safety** (Theorem 1, Theorem 2) and **liveness** (Theorem 3) under the standard partial synchrony model, with heuristic concentration bounds on per-block committee composition (Proposition 4) and a normalization-aware safety condition (Corollary 2).
- Bounded formal verification of all safety and liveness properties via **TLA+ model checking** (Section 9).

A step-by-step protocol specification is given in Appendix A; the full glossary and design rationale are available in [6].

---

2. Related Work

QuantumCoin's consensus protocol builds on the PBFT family [1] and inherits the standard partial synchrony model of Dwork, Lynch, and Stockmeyer [5]. This section positions QuantumCoin relative to the major deployed and academically analyzed BFT consensus protocols.

PBFT and its derivatives. The original PBFT protocol [1] provides safety and liveness under partial synchrony with $O(n^2)$ message complexity and a fixed, known validator set. PBFT's normal-case protocol uses three phases (Pre-Prepare, Prepare, Commit) with a separate View

Change sub-protocol for leader replacement. QuantumCoin extends this to a four-phase message structure (Proposal, Acknowledgment, Precommit, Commit) and retains the two-thirds quorum threshold but departs from PBFT in three further respects: (i) voting is deposit-weighted rather than per-validator, (ii) the validator committee rotates at every block via a three-pass selection algorithm (Section 10.2), and (iii) a two-round design with forced NIL escalation replaces the view-change sub-protocol. Tendermint/CometBFT adopts a similar PBFT-derived structure with deposit-weighted voting and a round-based design [19]. Tendermint's validator set can be updated at every block height (via the application's EndBlock callback), but is fixed within a height across rounds; it uses a weighted round-robin proposer selection algorithm [18] and gossip-based block dissemination through proposal and block-part messages rather than the hash-only proposal model used here.

Ethereum (Gasper). Ethereum's Gasper protocol [13] combines the Casper FFG finality gadget with the LMD-GHOST fork-choice rule. Finality is achieved after two epochs (~12.8 minutes) rather than per-block. The full active validator set (several hundred thousand validators as of 2026) is shuffled and divided into per-slot beacon committees across the 32 slots of each epoch; each validator attests exactly once per epoch in its assigned slot, with up to 64 committees per slot (target minimum committee size: 128) [17]. Randomness is RANDAO-based (accumulator), with a one-bit last-revealer bias [9]. QuantumCoin differs in providing immediate deterministic finality (no fork-choice rule), using a single bounded 128-validator committee per block (vs. Ethereum's per-slot aggregate of multiple committees), and employing committed-history randomness (vs. RANDAO). The trade-off is that Ethereum guarantees every active validator exactly one attestation slot per epoch (~6.4 minutes), ensuring predictable and uniform participation, while QuantumCoin's per-block committee selection is probabilistic — a validator may go many consecutive blocks without being selected.

Algorand. Algorand uses VRF-based cryptographic sortition to privately select committee members for each protocol step [14]. Committee membership is unknown until revealed, providing strong unpredictability. QuantumCoin's committee selection is deterministic and publicly computable (Section 10.2), sacrificing unpredictability for simplicity and auditability. Both protocols bound per-block communication complexity through committee selection from an unbounded participant set. Algorand's BA⋆ protocol reaches consensus in a constant expected number of steps (independent of network size), with especially fast completion when the selected block proposer is honest [14]; QuantumCoin guarantees finality within at most two rounds (worst case).

HotStuff and its variants. HotStuff [15] achieves O(n) message complexity (linear) via a leader-based three-phase protocol with threshold signatures, compared to QuantumCoin's $O(n^2)$ quadratic complexity. The linearity comes from aggregating votes through the leader rather than broadcasting. QuantumCoin's bounded committee ($n \leq 128$) makes the quadratic overhead acceptable in practice: $128^2 = 16{,}384$ messages per block, compared to millions for an unbounded full-set protocol. HotStuff uses a pacemaker for view synchronization; QuantumCoin uses timeout-based round escalation.

Randomness mechanisms. The committed-history randomness approach (Section 10.1) contrasts with three alternatives: RANDAO accumulators [2] (Ethereum), which suffer from last-revealer bias; VRF-based private sortition (e.g., Algorand [14]), which provides unpredictability but admits participant-side selective-abort behavior; and verifiable delay functions (VDFs), which provide unbiasable randomness but require specialized hardware and

introduce latency. QuantumCoin's approach trades predictability for non-manipulability: the seed is publicly computable once its inputs are known, but no current-block participant can influence it.

MEV mitigation. Most deployed protocols grant the block producer full control over transaction ordering. Flashbots' MEV-Boost implements proposer-builder separation (PBS) for Ethereum, delegating ordering to a specialized builder [16]. QuantumCoin's round-robin execution ordering is a protocol-level alternative: the proposer controls transaction selection but not execution order, which is fixed algorithmically. This approach does not require a trusted builder or relay infrastructure but provides weaker ordering guarantees than a cryptographic commit-reveal scheme.

---

3. Design Principles

The QuantumCoin consensus system is guided by the following principles, ordered by priority:

Safety. The protocol must never produce conflicting finalized blocks, even under adversarial conditions, network partitions, or asynchrony. Safety is independent of timing assumptions: it holds during periods of asynchrony, not only under partial synchrony. (The < 1/3 Byzantine fault tolerance bound remains a prerequisite; "independent of timing" means safety does not additionally require bounded message delay.)

Finality. Every finalized block is irreversible by design. The protocol provides no mechanism for rollback, chain reorganization, or re-voting on finalized blocks. Applications that observe a finalized block may treat it as a permanent commitment.

Decentralization. The protocol favors decentralized participation. Deposit normalization caps the influence of any single validator. Validator selection uses a randomness source that is resistant to proposer manipulation. Offline validators are progressively penalized to prevent a small number of inactive entities from degrading the network.

Active participation. Block rewards and transaction fee revenue accrue exclusively to the proposer of an accepted (OK-finalized) block; no passive yield is distributed to non-proposing validators. Proposers that fail to produce accepted proposals incur graduated penalties up to permanent exclusion (Section 11). This work-contingent reward structure renders passive deposit registration economically irrational (Section 11.7).

Liveness. Under partial synchrony (bounded message delay after an unknown stabilization time), the protocol guarantees that every honest validator eventually finalizes a block. The round escalation mechanism provides a bounded fallback that ensures progress even when Round 1 fails.

Scalability. The protocol selects a bounded-size validator committee per block (up to 128 validators) from the full registered set, which has no protocol-imposed upper bound. Any number of validators may register by locking a deposit; the 128 limit applies only to the per-block consensus committee, not to network membership. Committee membership rotates at every block height, giving all registered validators ongoing opportunity for selection, subject to the offline penalty and exclusion rules (Section 11.3, Section 11.5). The design bounds communication complexity while maintaining deposit-weighted security guarantees and is intended to be adaptable to improving hardware and network conditions.

---

4. Terminology

Term	Definition
Validator	A node representing an account that has registered by locking a deposit and is eligible to participate in consensus. There is no protocol-imposed limit on the total number of registered validators; the per-block committee size (up to 128) is a communication-complexity bound, not a network membership cap. The filtered validator set is deterministically selected per block (see Section 10.2).
Proposer	The single validator deterministically selected per round to create and broadcast a block proposal.
Round	An attempt to reach consensus on a block. Each block permits at most 2 rounds. Round 1 is a normal round. Round 2 is a forced empty-block round.
Proposal	A message broadcast by the proposer referencing the transactions to include in the block (by their hashes), the round number, and the proposed block time. Receiving validators resolve each referenced hash against their local mempool. In Round 2, the proposal references zero transactions.
Acknowledgment (ACK)	A vote sent by each validator in response to a proposal or a proposal timeout. Contains a vote type (OK or NIL) and a proposal hash. For NIL votes (timeout or Round 2), a deterministic nil-proposal hash derived from the parent hash and round number is used in place of an actual proposal hash.
Precommit	A vote sent by each validator after a two-thirds supermajority of acknowledgment votes has been collected. Computed as a hash of the proposal hash and vote type.
Commit	A vote sent by each validator after a two-thirds supermajority of precommit votes has been collected. Computed as a hash of the precommit hash.
OK	A vote type indicating acceptance of the proposed block (with transactions). Used only in Round 1.
NIL	A vote type indicating a vote for an empty block. Used when a proposal times out, or unconditionally in Round 2.
Deposit	The amount of coins locked by a validator upon registration. All quorum thresholds are weighted by deposit, not by validator count. This paper uses "deposit" as the canonical term; "stake" and "staked deposit" are synonyms.
Quorum	A set of validators whose combined deposit meets or exceeds the two-thirds threshold of the filtered set's total deposit.
Finalization	A block is finalized when a two-thirds weighted supermajority of commit votes has been collected. The block may contain transactions (OK) or be empty (NIL).
Honest Validator	A validator that strictly follows the protocol: it sends messages as specified, does not equivocate, and does not withhold votes.

Byzantine Validator	A validator that deviates arbitrarily from the protocol.
NIL Block	A finalized block that contains no transactions, produced when the consensus outcome is a NIL vote. This occurs when the proposal times out in Round 1 or unconditionally in Round 2. The chain advances (block number increments, block timestamp advances, and consensus metadata such as validator penalties and nil block counters are updated) but no user transactions are processed.
NilBlockCount	An on-chain counter tracking the number of times a validator has been attributed as the proposer of a NIL block finalized at Round 1. Drives progressive voting weight reduction, proposer deferral, and validator exclusion. Reset to zero when the validator successfully proposes an OK block.
Consensus Context	The deterministic pseudorandom seed used for validator selection and proposer ordering for a given block. Computed as Keccak256(BlockParentHash(N − 512,000) ‖ ValidatorCount(N)), where Keccak256 is defined in [11]. This is the base seed; individual selection hashes additionally incorporate the current block number to prevent periodic repetition when the validator set is static (see Section 10.1, "Block-number entropy").
Round Escalation	The transition from Round 1 to Round 2 within a single block, triggered when Round 1 cannot reach the two-thirds threshold. Analogous to a "view change" in PBFT or a "round change" in Tendermint. See Section 5.3.
Round Attribution	The principle that offline penalties are applied only when a block is finalized at Round 1 (indicating individual proposer failure), not at Round 2 (indicating a network-level issue). See Section 11.1.
Filtered Validator Set	The per-block subset of registered validators (drawn from the unbounded registered set) that remain eligible after applying minimum deposit requirements, validator exclusion, offline weight reduction, and committee selection. When deposit normalization applies (Section 12), quorum thresholds are computed against the post-normalization deposits; otherwise they are computed against the filtered set's total deposit. See Section 10.2.
Deposit Normalization	A single-pass per-block mechanism that clips any validator's effective deposit exceeding 10% of the filtered set's total deposit and redistributes the clipped excess to validators with a clean participation record. This adjustment affects only the effective deposit used for quorum calculations in the current block; no actual coins are transferred. Recipients may exceed 10% after redistribution. See Section 12.
Proposer Deferral	A penalty that excludes an offline validator from proposer selection for an exponentially increasing number of blocks based on its NilBlockCount. See Section 11.4.

Block Period	The fallback time increment between consecutive blocks (default: 6 seconds in genesis configuration). When a block does not carry a valid wall-clock time proposal (e.g., NIL-finalized blocks), its timestamp is parent timestamp + block period. For OK-finalized blocks with a valid proposed time, the timestamp is resynchronized to wall-clock time (see Section 5.5). Observed wall-clock block production times on mainnet are typically 9–30 seconds, depending on network propagation latency and consensus phase durations.

---

5. Consensus Protocol

The detailed 14-step protocol flow, including branching logic for proposer vs. non-proposer paths, timeout handling, and round escalation conditions, is given in Appendix A. This section summarizes the protocol structure.

Communication model. Validators exchange all consensus messages (proposals, acknowledgments, precommits, and commits) via authenticated peer-to-peer broadcast over a gossip network. Each message is cryptographically signed by the sender and relayed to all connected peers; receiving validators re-broadcast messages they have not previously seen. The protocol does not require direct point-to-point channels between every pair of validators.

5.1 Protocol States

Each round progresses through the following states in strict sequential order:

State	Exit Condition (transition to next state)
WAITING_FOR_PROPOSAL	Proposal message received from the round's proposer, or proposal timeout exceeded
WAITING_FOR_ACK	Acknowledgment votes for a single (vote type, proposal hash) pair reach the two-thirds deposit threshold
WAITING_FOR_PRECOMMIT	Precommit votes for a single precommit hash reach the two-thirds deposit threshold
WAITING_FOR_COMMIT	Commit votes for a single commit hash reach the two-thirds deposit threshold
FINALIZED	Terminal state; the block is finalized (OK or NIL, depending on the certified vote type)

5.2 Message Phases

The protocol proceeds through four phases per round for the proposer (three phases for non-proposer validators, who begin at Phase 2):

Phase 1 — Proposal. The proposer for the current round constructs a proposal referencing pending transactions by their hashes (or zero transactions if Round 2) and broadcasts it to other validators. Receiving validators verify the proposal by checking that every referenced transaction hash exists in their local mempool; transactions propagate to validator mempools via the normal gossip layer independently of the proposal. The proposer has full discretion over transaction selection (inclusion or exclusion); however, the execution order of included transactions is determined by a deterministic round-robin algorithm that the proposer cannot override (see Section 13 for MEV implications). In Round 2, Phase 1 is retained for protocol

uniformity; validators do not wait for the proposal before voting NIL, so the Round 2 proposer's reachability does not affect progress.

Phase 2 — Acknowledgment. Each validator evaluates the proposal:

- If a valid proposal (correctly signed, matching the expected round and parent hash) was received before the proposal timeout and the current round is Round 1, the validator broadcasts an acknowledgment with vote type OK for that proposal's hash. An honest validator acknowledges at most one proposal per round: it accepts the first valid proposal it receives and ignores any subsequent proposals for the same round.
- If the current round is Round 2, the validator broadcasts an acknowledgment with vote type NIL regardless of whether a proposal was received.
- If the proposal timeout was exceeded (Round 1 only), the validator broadcasts an acknowledgment with vote type NIL.

Each acknowledgment carries a vote type and a proposal hash. For OK votes, this is the hash of the received proposal; for NIL votes, it is a deterministic nil-proposal hash derived from the parent hash and round number. Once acknowledgment votes representing at least two-thirds of total filtered deposit have been collected **for a single (vote type, proposal hash) pair** (the threshold must be met entirely by votes sharing the same vote type and the same proposal hash; votes for different proposal hashes do not combine even if they share the same vote type), each validator broadcasts a precommit vote and advances to Phase 3.

Phase 3 — Precommit. Each validator computes its precommit as a hash of the proposal hash and vote type from the acknowledgment certificate it formed. Precommit votes are therefore bound to a specific certified value. Each validator collects precommit votes; once precommit votes for a single precommit hash representing at least two-thirds of total filtered deposit have been collected, the validator broadcasts a commit vote and advances to Phase 4.

Phase 4 — Commit. Each validator computes its commit as a hash of the precommit hash, again binding it to the same certified value. Each validator collects commit votes; once commit votes for a single commit hash representing at least two-thirds of total filtered deposit have been collected, the block is finalized. If the vote type is OK, the block includes the proposed transactions. If the vote type is NIL, the block is empty.

At any point during the above phases, if a valid finalized block for the current block number is received from the network (e.g., from validators that completed consensus earlier), the local validator aborts its current consensus instance and proceeds to the next block.

5.3 Round Escalation

If Round 1 cannot reach the two-thirds threshold in the acknowledgment or precommit phase, validators escalate to Round 2. The escalation triggers differ by phase:

- **Acknowledgment phase (Step 9.2).** A validator escalates if the acknowledgment timeout expires, **or** if it observes (through receipt of Round 2 messages) that validators already participating in Round 2 hold enough deposit that Round 1 can never reach the two-thirds threshold. Either condition alone is sufficient.
- **Precommit phase (Step 11.2).** A validator escalates only if the precommit timeout has expired **and** it observes that Round 2 is already deposit-dominant. Both conditions are required.

Rationale for the asymmetry. In the acknowledgment phase, a validator has not yet formed a quorum certificate and can safely abandon Round 1 as soon as Round 1 quorum becomes mathematically impossible. In the precommit phase, the validator has already collected an acknowledgment quorum certificate (binding it to a specific vote type and proposal hash) and may be close to forming a precommit quorum. Requiring the timeout prevents premature abandonment of a Round 1 precommit quorum that might still succeed — a validator that has collected, say, 60% of precommit deposit should wait for the timeout before discarding that progress, in case the remaining honest precommit votes arrive within the timeout window. The AND condition is a conservative choice that avoids unnecessary Round 2 escalations at the cost of slightly higher latency when Round 1 genuinely cannot succeed at the precommit phase.

Lemma 3 (Precommit-phase deadlock unreachability). Under Assumption A3 (Byzantine deposit < 1/3), the state in which all honest validators are in Round 1 precommit and no honest validator is in Round 2 is unreachable. Therefore the Step 11.2 AND condition cannot produce a liveness deadlock.

Proof. A validator enters the precommit phase only after collecting an acknowledgment quorum — a set of ACK votes for a single (vote type, proposal hash) pair whose combined deposit meets the two-thirds threshold (Step 9.1). Each honest validator's precommit hash is deterministically derived from this quorum certificate. We consider three exhaustive cases for how honest validators are distributed across protocol phases:

Case 1 — All honest validators form the same ACK quorum (all OK, or all NIL). All honest validators enter precommit with identical precommit hashes. Their precommit votes are mutually compatible and accumulate. Since honest deposit exceeds two-thirds (Assumption A3), precommit quorum is reached after GST. The validator advances directly to the commit phase via the normal quorum-accumulation path (Step 11.1). The Step 11.2 escalation condition is never evaluated. No deadlock.

Case 2 — Some honest validators form an ACK quorum; others do not. The honest validators that did not collect an ACK quorum remain in the acknowledgment phase. Their acknowledgment timeout eventually fires, triggering escalation to Round 2 via the OR condition (Step 9.2: timeout alone is sufficient). These validators enter Round 2 and broadcast Round 2 messages. Any honest validator still in Round 1 precommit that cannot reach precommit quorum will, after its precommit timeout, observe the Round 2 deposit evidence from the escalated validators, satisfying the AND condition and escalating. No deadlock.

Case 3 — No honest validator forms an ACK quorum. All honest validators remain in the acknowledgment phase, timeout, and escalate to Round 2 via Step 9.2. No validator enters precommit. No deadlock.

The scenario "all honest validators are in precommit and no one is in Round 2" can only arise in Case 1, where all honest validators hold compatible precommit hashes and precommit quorum succeeds. In Cases 2 and 3, at least some honest validators escalate from the acknowledgment phase, providing the Round 2 evidence that Step 11.2 requires.

Two conflicting ACK quorums (one OK, one NIL) cannot coexist under Assumption A3. Any two quorums of two-thirds deposit must overlap by more than one-third of total deposit (Lemma 1). Since Byzantine deposit is strictly less than one-third, the overlap contains at least one honest validator. That honest validator would need to have cast both an OK and a NIL acknowledgment for the same (height, round) pair, violating the single-value binding rule

(Lemma 2). Therefore Byzantine equivocation at the ACK phase can cause at most one quorum to form, never both simultaneously. ∎

No dual-round participation invariant. An honest validator participates in at most one round per block height. Once a validator escalates to Round 2 (via step 14 in Appendix A), it reinitializes its local state for Round 2 and does not process or contribute further Round 1 votes. This is enforced by the local state machine: the round escalation step sets round = 2 and returns to the proposer selection step (step 5), discarding all Round 1 state. There is no protocol path that returns a validator to Round 1 after escalation. This invariant is a prerequisite for the safety argument (Section 8.1, Theorem 1): the proof that honest validator V "cannot have produced both commits" relies on the fact that V's votes are confined to a single round.

Gossip relay of Round 1 messages after escalation. The gossip layer operates below the consensus state machine: a validator re-broadcasts all validly signed consensus messages it receives, regardless of its own current round. After escalating to Round 2, a validator may receive and relay delayed Round 1 messages from slow network paths. This does not violate the no-dual-round-participation invariant — the invariant constrains voting, not message relay. The relayed messages are votes originally cast by their signers while those signers were in Round 1; the relay creates no new votes. A validator W that has not yet escalated may use these relayed Round 1 votes to form a Round 1 quorum, which is correct behavior: W is still in Round 1 and the votes are valid Round 1 votes from their original signers. Safety is preserved because the quorum intersection argument (Section 8.1, Lemma 1) depends on the signing validator's round confinement, not on the relay path.

Round 2 is a **forced empty-block round**. A different proposer is selected for Round 2 (using the same consensus context but with the round number as an additional input). The Round 2 proposer must include zero transactions, and all validators must vote NIL. Under the partial synchrony assumption (A1, Section 6.1; see Theorem 3, Section 8.3), this guarantees that the two-thirds threshold is reached: honest validators control more than two-thirds of deposit, all vote NIL unconditionally, and their messages are eventually delivered, producing a finalized empty block.

Each block permits at most 2 rounds. There is no Round 3. If Round 2 cannot reach the two-thirds threshold (possible only during a severe network partition where honest messages are not delivered), the chain halts at this block height and waits until message delivery resumes. Safety is preserved during the halt; liveness resumes once partial synchrony is restored.

5.4 Design Rationale: Why Round 2 Forces Empty Blocks

When Round 1 cannot reach the two-thirds threshold — because some validators accepted the proposal while others voted NIL (having not received the proposal in time, or not having received the proposed transactions in their mempool, or having rejected the proposal), with neither vote type holding sufficient deposit — the protocol must resolve the deadlock.

An alternative design would allow additional rounds where validators vote freely (OK or NIL) on a new proposal. Note that QuantumCoin does select a different proposer for Round 2, but the critical difference is that Round 2 forces all validators to vote NIL unconditionally — the proposer's reachability is irrelevant. A design that instead allowed free voting in additional rounds would have two fundamental problems:

1. **No convergence guarantee.** If the network is degraded, every retry round may stall for the same reason: honest validators have differing views of the proposal, leading to a split vote. Additional rounds do not resolve the underlying disagreement; they repeat it. The chain could stall indefinitely, violating liveness.
2. **Increased Byzantine attack surface.** Each additional round with free voting gives Byzantine validators another opportunity to equivocate or selectively withhold proposals.

By forcing NIL votes in Round 2, the protocol eliminates the source of disagreement. Every honest validator votes NIL unconditionally, regardless of whether the Round 2 proposal was received. Since honest validators control more than two-thirds of deposit and their messages are eventually delivered (under the partial synchrony assumption A1, Section 6.1; see Theorem 3, Section 8.3), the NIL threshold is guaranteed to be reached. Round 2 succeeds by construction under these assumptions.

Validator re-selection advantage. A further benefit of finalizing quickly — even with an empty block — is that validator selection is performed independently for each block from the full registered validator set, which has no upper bound. The validator set for block N + 1 is derived from on-chain state at block N and may differ from the set at block N. By advancing to a new block, the protocol obtains a potentially different validator set that may include better-connected nodes or exclude validators that were unreachable. Additional rounds within the same block reuse the same validator set that already failed to reach consensus, offering no opportunity for the network topology to improve the outcome. Finalizing an empty block and re-entering validator selection converts a per-block liveness failure into an independent retry with fresh sampling.

Trade-off. One block's worth of transactions is delayed — pending transactions remain in the mempool and are included in a subsequent block. Under sustained network degradation affecting all validator sets equally, the chain produces consecutive empty blocks — formally live but not processing transactions until conditions improve.

5.5 Block Timestamp Validity and Clock Synchronization

Block timestamp validity. Block timestamps advance through two complementary mechanisms:

- **Incremental fallback.** By default, each block's timestamp is the parent block's timestamp plus the block period (6 seconds, as configured in genesis). This applies to NIL-finalized blocks and to any block where the proposer does not provide a valid wall-clock time. The 6-second increment is a minimum-progress guarantee that keeps timestamps monotonically increasing regardless of actual wall-clock elapsed time.
- **Wall-clock resynchronization.** At eligible block heights, the proposer includes a proposed block time derived from its local UTC clock, rounded down to the nearest minute. Validators accept this time if and only if (a) it has minute-level granularity (seconds and sub-second components must be zero), (b) it falls within ±3 minutes of the receiving validator's local clock, and (c) it exceeds the parent block's timestamp. When all three conditions are met and the block finalizes with vote type OK, the block's timestamp is set to the proposed time, resynchronizing the on-chain clock to real wall-clock time.

Resynchronization frequency. Every OK-finalized block carries a wall-clock time proposal from the block proposer, so the on-chain clock resynchronizes at every OK block where the proposed time exceeds the parent timestamp. NIL-finalized blocks always use the 6-second

fallback, because the proposer's time proposal is discarded when the block finalizes as NIL.

Clock synchronization dependency. The ±3-minute timestamp validity window assumes that all validators maintain loosely synchronized clocks, typically via NTP or an equivalent time synchronization service. A validator whose clock drifts beyond this window will reject valid proposals or propose blocks that other validators reject, effectively behaving as a crash fault. In particular, if a proposer's clock is skewed, other validators will reject the proposed block time, vote NIL, and the block will finalize as NIL at Round 1 — costing the proposer the block reward and incrementing its NilBlockCount penalty counter (Section 11.1). The protocol does not enforce or verify clock synchronization; operators are responsible for maintaining adequate time synchronization on their nodes. This is a standard operational assumption in distributed systems and is shared by most blockchain protocols that use wall-clock timestamps.

---

6. System Model and Quorum Threshold

6.1 System Model

The protocol operates under the following assumptions, which are referenced by the safety and liveness proofs in Section 8.

(A1) Partial synchrony. There exists an unknown Global Stabilization Time (GST) and an unknown finite bound Δ such that, after GST, all messages between honest validators are delivered within time Δ [5]. Before GST, messages may be delayed arbitrarily. Safety does not depend on this assumption; liveness does.

(A2) Authenticated channels. All consensus messages are digitally signed using hybrid post-quantum cryptographic signatures [10]. Honest validators' signatures cannot be forged. This is a prerequisite for the quorum intersection argument: without authentication, a Byzantine validator could impersonate an honest validator.

(A3) Byzantine bound. The total deposit controlled by Byzantine validators is strictly less than 1/3 of the post-normalization committee effective deposit. Byzantine validators may exhibit arbitrary behavior: equivocation, selective delivery, arbitrary voting, phase skipping, or crash (see Section 7 for the full fault model).

(A4) Deposit-weighted voting. All quorum thresholds are computed as fractions of total effective deposit, not validator count. A phase transition requires votes whose combined deposit weight meets or exceeds the quorum threshold.

(A5) Single proposer per round. Exactly one validator is deterministically designated as proposer for each (block height, round) pair (Section 10.3). A Byzantine proposer may equivocate (send different proposals to different validators) but cannot alter the fact that honest validators accept at most one proposal per round.

6.2 Quorum Threshold

The protocol uses 67% of total filtered deposit as the quorum threshold for all phase transitions. This is the protocol's operational rule: a phase transition requires deposit weight satisfying

```
required_deposit = total_filtered_deposit * 67 / 100
```

computed using integer arithmetic (truncating division).

Derivation. The threshold is derived from the BFT quorum intersection requirement. Let f denote the maximum fraction of total deposit controlled by Byzantine validators, and let q denote the quorum threshold. For safety, any two quorums must share at least one honest validator:

```
2q − 1 > f
```

Setting f < 1/3 (the standard BFT bound):

```
2q − 1 > 1/3
q > 2/3
```

The quorum threshold must therefore exceed 2/3 (66.67%). The protocol chooses 67% — the smallest integer percentage that satisfies this bound — as its fixed threshold.

Proposition 1 (Integer arithmetic safety). For all positive integers N >= 298, floor(N × 67 / 100) > 2N/3.

Proof. For any positive integer N, the remainder of N × 67 divided by 100 is at most 99, so floor(N × 67 / 100) >= (67N - 99) / 100. The inequality (67N - 99) / 100 > 2N/3 is equivalent to 3(67N - 99) > 200N, i.e., 201N - 297 > 200N, i.e., N > 297. Therefore for all integers N >= 298, floor(N × 67 / 100) > 2N/3. ∎

Corollary 1. For N >= 298, the quorum overlap 2 × floor(N × 67 / 100) - N > N/3 >= floor(N/3) >= B, where B is the maximum Byzantine deposit. Any two quorums therefore overlap by more than the maximum Byzantine deposit, guaranteeing that the overlap contains at least one honest validator.

Application to protocol parameters. All deposits are denominated in wei (1 coin = 10^18 wei). The protocol enforces a minimum block deposit of 500 billion coins (5 × 10^29 wei) across at least 3 validators, each holding at least 5 million coins (5 × 10^24 wei). The minimum possible N is therefore 5 × 10^29, which exceeds 298 by a factor of approximately 10^27. Proposition 1 applies unconditionally to all deposit values the protocol permits.

Exhaustive verification for small N. For completeness: computational enumeration confirms that for N < 298, there exist values where floor(N × 67 / 100) <= 2N/3 (specifically, at every multiple of 3 from 3 to 297 and at various other small values). The protocol's minimum deposit thresholds preclude all such values.

---

7. Fault Model

The protocol is Byzantine Fault Tolerant and tolerates up to strictly less than one-third of the committee's effective deposit controlled by Byzantine validators (Assumption A3 in Section 6.1).

Network-level to committee-level translation. Because the committee is a subset of the full registered set and voting is deposit-weighted, an attacker controlling fraction f of total

network deposit controls in expectation at most fraction f of the committee's pre-normalization deposit. Committee selection cannot amplify the attacker's expected deposit fraction — it can only reduce it (if some attacker identities are not selected) or leave it unchanged. However, for any specific block, sampling variance in Passes 2 and 3 may cause the realized committee Byzantine fraction to exceed f (see Proposition 4 for tail bounds on this deviation). The $< 1/3$ bound stated at the network level therefore implies the same bound at the committee level before normalization in expectation; the per-block tail probability decreases exponentially with the gap between f and 1/3 (Proposition 4). After normalization (Section 12), the effective Byzantine fraction may differ: clipping large honest validators or redistributing excess to Byzantine validators with clean records can shift the ratio. The protocol's safety assumption (A3) is ultimately that Byzantine validators control less than 1/3 of the **post-normalization committee effective deposit**.

Byzantine validators may exhibit any combination of the following behaviors:

- **Equivocation.** Send OK votes to some peers and NIL votes to others for the same acknowledgment phase, causing different honest validators to observe different vote types from the same source.
- **Selective delivery.** A Byzantine proposer may deliver the proposal to only an arbitrary subset of validators, or to none.
- **Arbitrary voting.** Vote OK or NIL in any phase regardless of protocol rules or what was received.
- **Phase skipping.** Advance to later phases (precommit, commit) without completing earlier ones.
- **Crash or abstention.** Stop participating entirely — go offline, crash, or withhold all votes. This is a special case of Byzantine behavior: a Byzantine validator that chooses inaction is indistinguishable from a crashed validator.

Message authentication. All consensus messages (proposals, acknowledgments, precommits, and commits) are digitally signed using the sending validator's registered key pair (Assumption A2). The protocol uses hybrid post-quantum cryptographic signatures (see [10] for the PQC specification); signature verification on receipt prevents impersonation and enables identification of the sender. This is a prerequisite for the fault model: without authenticated messages, a Byzantine validator could forge messages on behalf of honest validators, breaking the quorum intersection argument.

Equivocation evidence. The protocol does not currently include an on-chain equivocation evidence mechanism (where two conflicting signed votes from the same validator for the same round and phase could be submitted as proof to trigger punitive slashing). Economic disincentives for Byzantine behavior rest on the 1/3 fault tolerance bound, the per-NIL-block deposit deduction (Section 11.2), and the progressive offline penalty system (Sections 11.3-11.5). An on-chain equivocation evidence and slashing mechanism is a potential future enhancement.

Penalty asymmetry. The absence of equivocation slashing creates an asymmetry in the penalty structure: an honestly offline proposer incurs deposit deductions and progressive penalties (Section 11), while an actively Byzantine validator that equivocates (sending conflicting votes to different peers) faces no additional economic penalty beyond the inherent 1/3 fault tolerance bound. Furthermore, a Byzantine coalition controlling just under 1/3 of deposit can equivocate or selectively withhold votes to force Round 2, shielding co-conspirator proposers from offline penalties (per Section 11.1) at zero direct cost for the

equivocation itself. This asymmetry is a known consequence of the current penalty design; the protocol's safety guarantee does not depend on punishing equivocation — it depends on the quorum intersection property — but the lack of equivocation penalties may affect the economic deterrence model for rational adversaries. On-chain equivocation evidence would close this gap. A secondary motivation for omitting equivocation slashing is to avoid penalizing operators for accidental conflicting votes caused by faulty software, hardware failures, or misconfigured redundancy (e.g., the same validator key running on two nodes simultaneously after a crash-restart), which are operationally common and do not indicate malicious intent.

Under Assumptions A1–A5 (Section 6.1), the protocol guarantees:

- **Safety (Agreement) — Theorem 1 (Section 8.1).** No two honest validators finalize at the same block height with different blocks.
- **Safety under Asynchrony — Theorem 2 (Section 8.2).** Safety holds regardless of network timing, including during periods of asynchrony before GST.
- **Liveness (Termination) — Theorem 3 (Section 8.3).** Under partial synchrony (after GST), all honest validators eventually finalize a block within bounded time.

---

8. Safety and Liveness Proofs

This section presents formal safety and liveness proofs under the system model defined in Section 6.1. All proofs are pen-and-paper arguments from Assumptions A1–A5; the TLA+ model checking (Section 9) provides complementary bounded verification.

Notation. Throughout Sections 8–11, set-measure expressions such as "|S|", "S exceeds two-thirds", or "the overlap of Q_1 and Q_2" refer to deposit-weighted measure unless explicitly stated otherwise. That is, |S| denotes the sum of effective deposits of all validators in set S, and "S exceeds two-thirds" means $|S| \geq \text{floor}(N \times 67 / 100)$ where N is the total committee deposit. Validator counts (cardinalities) are always written explicitly (e.g., "the number of validators in S").

Definition 1 (Finalization). A block b at height h is finalized by an honest validator V when V collects a commit certificate: a set of commit votes for a single certified value (vote type, proposal hash) with combined deposit weight $\geq \text{floor}(N \times 67 / 100)$, where N is the total post-normalization committee deposit.

Definition 2 (Certified value). A certified value is a pair (voteType, proposalHash) where voteType $\in$ {OK, NIL}. Each commit vote is deterministically derived from a precommit vote, which is derived from an acknowledgment vote, which binds the validator to a specific certified value for a given (height, round) pair.

Definition 3 (Quorum). A quorum Q is a set of validators whose combined post-normalization deposit weight is at least $\text{floor}(N \times 67 / 100)$.

Lemma 1 (Quorum intersection). Under Assumption A3 (Byzantine bound), any two quorums Q_1 and Q_2 share at least one honest validator.

Proof. By Proposition 1 (Section 6.2), for all $N \geq 298$, $\text{floor}(N \times 67 / 100) > 2N/3$. Therefore $|Q_1| + |Q_2| > 2 \times (2N/3) = 4N/3$. Since the total deposit is N, the overlap is $|Q_1 \cap Q_2| > 4N/3 - N = N/3$. By A3, Byzantine deposit is strictly less than N/3, so the overlap contains at least one

honest validator. ∎

Lemma 2 (Single-value binding). An honest validator produces at most one certified value per (height, round) pair.

Proof. By protocol rules (Section 5.2), an honest validator accepts only the first valid proposal it receives for a given round and derives its acknowledgment, precommit, and commit deterministically from that single proposal. An honest validator does not participate in Round 1 phases after entering Round 2. By Assumption A2 (authenticated channels), an honest validator's signed votes cannot be forged. Therefore, for any (height, round) pair, an honest validator's commit votes correspond to exactly one certified value. ∎

8.1 Safety Proof (Theorem 1: Agreement)

Theorem 1 (Agreement). Under Assumptions A2–A5, no two honest validators finalize at the same block height with different certified values.

Proof. Suppose for contradiction that honest validator A finalizes with certified value c_1 and honest validator B finalizes with certified value $c_2 \neq c_1$ at the same height h. By Definition 1, A collected a commit quorum Q_1 for c_1 and B collected a commit quorum Q_2 for c_2. By Lemma 1, $Q_1 \cap Q_2$ contains at least one honest validator V. We consider two cases.

Case 1 — Same round. Both Q_1 and Q_2 contain votes from the same round r. By Definition 2, V's commit for c_1 was derived from an acknowledgment for c_1 in round r, and V's commit for c_2 was derived from an acknowledgment for c_2 in round r. But by Lemma 2, V produces at most one certified value per (height, round) pair. Contradiction.

Case 2 — Cross-round. Q_1 contains Round 1 votes and Q_2 contains Round 2 votes (or vice versa). Then V must have cast a commit vote in Round 1 and a commit vote in Round 2 for the same block height. A commit vote in Round 1 requires V to have completed the Phase 2 → Phase 3 → Phase 4 progression while in Round 1 — that is, V had not yet escalated to Round 2 at the time of its Round 1 commit. A commit vote in Round 2 requires V to have entered Round 2 via step 14 (Section 5.3) and completed the Round 2 progression. But the no-dual-round-participation invariant (Section 5.3) states that once V escalates, it reinitializes local state and sets round = 2, discarding all Round 1 state. There is no protocol path that returns V to Round 1. Therefore V cannot have produced a Round 1 commit after escalating, and it cannot have escalated before producing the Round 1 commit (because it completed Phase 4 in Round 1). Gossip relay of delayed Round 1 messages after escalation (Section 5.3) does not create new votes — it retransmits votes already signed by their original senders. V would need to have been simultaneously in Round 1 (to produce c_1's commit) and Round 2 (to produce c_2's commit), which violates the invariant. Contradiction. ∎

This covers all conflict sub-cases: OK vs. NIL conflicts (where c_1 = (OK, p) and c_2 = (NIL, ⊥)), equivocation conflicts (where c_1 = (OK, p_1) and c_2 = (OK, p_2) with $p_1 \neq p_2$, which a Byzantine proposer may attempt by sending different proposals to different validators), and cross-round conflicts (where c_1 is certified in Round 1 and c_2 in Round 2).

Structural independence. Theorem 1 depends on the quorum intersection arithmetic (Proposition 1, Lemma 1), the single-value binding rule (Lemma 2), the deterministic hash chaining from acknowledgment through precommit to commit, and message authentication (A2). All four properties hold regardless of the number of validators, the deposit distribution, or network timing. The TLA+ model checking results (Section 9) verify Agreement exhaustively over bounded configurations; Theorem 1 extends the reasoning to the general

case.

8.2 Safety under Asynchrony (Theorem 2)

Theorem 2 (Safety under Asynchrony). Theorem 1 holds regardless of network timing, including during periods of asynchrony before GST.

Proof. The proof of Theorem 1 relies on Assumptions A2–A5 and Proposition 1. None of these depend on message delivery timing: Proposition 1 is a number-theoretic property of the quorum threshold; Lemma 1 (quorum intersection) is a counting argument over deposit weights; Lemma 2 (single-value binding) depends on the protocol's local decision rule and message authentication, not on when messages arrive. The partial synchrony assumption (A1) is not invoked. Therefore, even if messages are arbitrarily delayed, reordered, or lost, no two honest validators can both collect commit quorums for conflicting certified values — the quorum intersection always contains an honest validator who could not have committed to both values. Safety holds at all times, including before GST. ∎

Note. During asynchrony, liveness may be violated (the chain may stall), but safety is never compromised. Pending transactions remain in the mempool during a stall and are included once block production resumes.

8.3 Liveness Proof (Theorem 3: Termination)

Theorem 3 (Termination). Under Assumptions A1–A5, after GST every block height is finalized within bounded time.

Proof. The proof proceeds in two parts: (i) Round 1 either finalizes or all honest validators escalate to Round 2; (ii) Round 2 finalizes within bounded time.

(i) Round 1 terminates or escalates. After GST, each phase has a configurable timeout. If the proposer is honest and reachable, it broadcasts the proposal, which arrives at all honest validators within Δ. Honest validators vote OK and proceed through all phases; finalization occurs within 4Δ (one round-trip per phase). If the proposer is Byzantine, offline, or unreachable, honest validators do not receive a valid proposal within the proposal timeout and vote NIL. If the NIL acknowledgment quorum is reached, the block finalizes as NIL in Round 1 within the phase timeouts. If Round 1 does not reach quorum in either the ACK or Precommit phase before the respective timeout fires, validators escalate to Round 2 (Section 5.3). Each phase timeout is finite, so Round 1 terminates or triggers escalation in bounded time.

(ii) Round 2 terminates within 3Δ of the last honest entry. In Round 2, all honest validators vote NIL unconditionally. Let t_last be the wall-clock time at which the last honest validator enters Round 2. By A1, all honest ACK/NIL votes are delivered by t_last + Δ. Since honest deposit exceeds 2/3 of total deposit (by A3, Byzantine deposit < 1/3), the NIL acknowledgment threshold is reached by t_last + Δ. Each honest validator that reaches the acknowledgment threshold sends a precommit, delivered by t_last + 2Δ. Each honest validator that reaches the precommit threshold sends a commit, delivered by t_last + 3Δ. All honest validators finalize by t_last + 3Δ.

The total worst-case per-block latency is bounded by the Round 1 timeout plus 3Δ. Honest validators may enter Round 2 at different wall-clock times (depending on when each validator's local timeout fires), but convergence is guaranteed because all Round 2 NIL votes are compatible and eventually delivered. ∎

Timeout configuration. The protocol does not require that timeouts match Δ exactly; it requires only that after GST, timeouts are long enough for honest messages to arrive. If timeouts are shorter than Δ, Round 1 may time out unnecessarily, but Round 2 still guarantees progress under the partial synchrony assumption.

8.4 FLP Impossibility

The FLP impossibility result [4] proves that no deterministic consensus protocol can guarantee both safety and liveness in a purely asynchronous network where even a single process may fail. In a fully asynchronous model, messages may be delayed indefinitely, making it impossible to distinguish a crashed node from a slow one.

QuantumCoin's partial synchrony assumption (A1) places the protocol outside FLP's scope. Under partial synchrony, liveness is achievable alongside safety, consistent with [5]. The protocol does not solve or circumvent FLP; it operates under strictly stronger network assumptions. As shown in Theorem 2, safety holds even during asynchronous periods (before GST); only liveness requires the partial synchrony assumption.

8.5 Normalization-Aware Safety (Corollary 2)

Corollary 2 (Normalization-aware safety). Under Assumptions A2–A5, if the pre-normalization Byzantine deposit fraction f and the aggregate honest clipped deposit fraction h_clip satisfy f / (1 − h_clip) < 1/3, then no two honest validators finalize at the same block height with different certified values.

Proof. By Proposition 2 (Section 12), the post-normalization Byzantine fraction f' ≤ f / (1 − h_clip). If f / (1 − h_clip) < 1/3, then f' < 1/3, so Assumption A3 is satisfied (post-normalization Byzantine deposit < 1/3). Theorem 1 then applies directly. ∎

This corollary connects the safety theorem (Section 8.1) to the normalization mechanism (Section 12), providing an observable, pre-normalization sufficient condition for safety. The constraint f / (1 − h_clip) < 1/3 is not enforced on-chain because Byzantine identity is unknown to the protocol. Operators should monitor pre-normalization deposit distributions to ensure the constraint is not violated — particularly when a small number of honest validators hold a large fraction of total deposit, which increases h_clip and narrows the safety margin.

---

9. Formal Verification

The consensus protocol has been formally specified in TLA+ and verified using the TLC model checker [7]. The specification models the protocol's message phases, deposit-weighted voting, two-round escalation, proposal identity tracking (hash-binding), and Byzantine faults including vote equivocation and proposer equivocation (sending different proposals to different validators). Full verification results, state-space metrics, and counterexample analysis are documented in [8].

Model-specification gap. The TLA+ specification is a model of the consensus protocol, not a direct translation of the production Go implementation. The model abstracts away several aspects of the full protocol: deposit normalization (Section 12), the offline penalty system (Section 11), the three-pass committee selection algorithm (Section 10.2), timestamp validation (Section 5.5), and network topology (gossip propagation). These abstractions mean that the TLC verification confirms properties of the idealized model, not of the implementation

directly. The verified properties (safety, liveness, fault tolerance boundary) hold under the model's assumptions; extending verification to cover the abstracted mechanisms or performing code-level verification against the Go implementation is future work.

9.1 TLA+ Specification

The specification models the following Byzantine behaviors nondeterministically:

- **Proposer equivocation**: a Byzantine proposer delivers different proposal identities to different honest validators, attempting to create competing OK quorums for different blocks.
- **Vote equivocation**: a Byzantine validator sends votes for all (voteType, proposalId) combinations for the same round, so different honest peers can form quorums for different certified values.
- **Selective proposal delivery**: a Byzantine proposer delivers the proposal to an arbitrary subset of validators.
- **Arbitrary voting**: Byzantine validators send votes for all (voteType, proposalId) combinations regardless of preconditions.
- **Phase skipping**: Byzantine validators advance through protocol phases without waiting for thresholds.
- **Crash or abstention**: a Byzantine validator stops participating entirely. This is implicitly modeled: Byzantine actions are nondeterministic, so a validator that never acts is a valid behavior.

Honest validators are modeled with weak fairness (WF), which guarantees that if an action is continuously enabled, it is eventually taken. This encodes the partial synchrony assumption: honest validators' messages are eventually delivered.

9.2 Safety Properties

The following invariants are checked exhaustively across all reachable states. Agreement and Validity are safety properties in the distributed systems sense. TypeOK, Round2Consistency, and CommitIntegrity are protocol correctness invariants that verify the implementation adheres to design rules.

Property	Definition
TypeOK	All variables remain within their declared types.
Agreement	No two honest validators finalize at the same block height with different blocks. Checks both vote type and proposal identity, covering OK-vs-NIL and OK(blockA)-vs-OK(blockB) conflicts from an equivocating proposer.
Validity	If all honest validators voted OK, no honest validator finalizes as NIL. (This is a protocol-specific consistency property, not the standard BFT validity notion. Standard BFT validity — "if all correct processes propose value v, all correct processes decide v" — is inapplicable because validators do not propose; they vote in response to a single proposer's proposal, and any validator may vote NIL on a proposal timeout. The decision domain is asymmetric. The stated property captures the analogous guarantee: unanimous OK votes by honest validators are not overridden by a NIL finalization.)
Round2Consistency	Any honest finalization in Round 2 has vote type NIL.

CommitIntegrity	No honest validator finalizes without two-thirds commit deposit weight for its specific certified value (vote type and proposal identity).

9.3 Liveness Properties

The following temporal property is verified:

Property	Definition
Termination	All honest validators eventually finalize.

9.4 Fault Tolerance Boundary

The model checker was run on three configurations to precisely characterize the fault tolerance boundary:

Configuration	Byzantine Deposit	Expected Result	Actual Result
Safe	25% (1 of 4 equal-deposit validators)	All properties pass	Pass (43,911 states)
Near-boundary	33% (1 validator holding 33% of total deposit)	All properties pass	Pass (43,911 states)
Unsafe	34% (2 validators, combined 34% of total deposit)	Safety violation	Agreement violated (expected)

The theoretical fault tolerance bound is strictly less than 1/3 (33.33...%) of total deposit. The model checker's deposit discretization does not permit testing at exactly 1/3, so the Near-boundary configuration tests at 33% — the closest representable value below the bound — and passes. The Unsafe configuration, at 34% (the closest representable value above the bound), produces a counterexample in which two honest validators finalize with conflicting blocks (different vote types and/or different proposal identities). The detailed counterexample trace and state-space metrics for all three configurations are documented in [8].

Generalization beyond bounded instances. TLC model checking verifies properties exhaustively over the configured state space (4 validators, 2 abstract proposal identities, specific deposit distributions) but does not constitute a proof for the general unbounded case. The safety invariant verified by TLC — that no two honest validators finalize with conflicting blocks (differing in vote type or proposal identity) — is structurally independent of the number of validators: it depends only on the quorum intersection arithmetic and the single-value-per-round rule for honest validators (Section 8.1, Lemmas 1–2), which scale uniformly with any number of validators, any deposit distribution, and any number of proposal identities. The pen-and-paper safety proof in Section 8.1 (Theorem 1) extends the reasoning to the general case.

Scope of the current specification. See the "Model-specification gap" paragraph at the beginning of this section for a summary of what the TLA+ model abstracts away and the implications for verified properties.

---

10. Validator and Proposer Selection

10.1 Consensus Randomness

Validator and proposer selection require a deterministic pseudorandom source that all validators compute identically and that is resistant to manipulation by the current block's proposer.

Taxonomy of consensus randomness approaches. Existing proof-of-stake protocols employ two primary approaches to consensus randomness, each with a characteristic vulnerability:

- **Accumulator-based (e.g., RANDAO [2]).** Each block proposer contributes a reveal (e.g., a BLS signature) that is mixed into a running accumulator. The randomness seed is the accumulator state at a designated point. Vulnerability: the last contributor to the accumulator can choose between two (or more) outcomes by withholding their reveal — the "last-revealer" bias problem. Recent work [9] has shown this extends to forking attacks that selectively exclude honest proposers' reveals.
- **VRF-based (e.g., Algorand [14]).** Each validator privately evaluates a Verifiable Random Function keyed to their secret key. The VRF output determines committee membership, which is revealed only when the validator publishes its proof. Vulnerability: the current-slot participant can evaluate the VRF privately and choose whether to participate (a "grindable abort" decision). Committee membership is unpredictable externally, but the VRF evaluator retains a unilateral choice over a binary outcome (participate or abstain).

QuantumCoin introduces a third approach — **committed-history randomness** — that avoids both vulnerability classes. The randomness seed for block N is derived entirely from on-chain data committed at block N − 512,000, hundreds of thousands of blocks in the past. No current-block participant contributes any input to the seed: there is no reveal to withhold (eliminating last-revealer bias) and no private function to selectively evaluate (eliminating grindable abort). The trade-off is that the seed is publicly predictable once its inputs are known — adversarial predictability replaces adversarial influence. The protocol bounds the utility of prediction through the 512,000-block lookback (any validator registration change invalidates predictions), the block-number mixing (committee composition changes at every height even with a static seed), and the 32,000-block withdrawal delay (Section 10.5, limiting the rate of validator-count grinding).

The randomness source for block N is defined as:

```
ConsensusContext(N) = Keccak256( BlockParentHash(N − X) ‖ ValidatorCount(N) )
```

where:

- X = 512,000 (the lookback constant; see below for bootstrap behavior).
- BlockParentHash(N − X) is the parent hash field of block N − X (i.e., the hash of block N − X − 1), as recorded on-chain.
- ValidatorCount(N) is the total number of registered validators (before filtering) as of the parent state of block N. The consensus context smart contract stores BlockParentHash as a 32-byte bytes32 value. The final hash is computed off-chain in the consensus layer (Go implementation) as Keccak256(BlockParentHash ‖ ASCII(ValidatorCount)), where ASCII(ValidatorCount) is the validator count's base-10 decimal string representation (e.g., the integer 1000 is encoded as the 4-byte ASCII string "1000"). The concatenation is

therefore a 32-byte hash followed by a variable-length ASCII string. This encoding is unambiguous because BlockParentHash is fixed at 32 bytes and the ASCII suffix encodes a non-negative integer whose string length is uniquely determined by its value.
- ‖ denotes concatenation.
- Keccak256 [11] is the hash function (the pre-FIPS-202 variant of SHA-3, as used by Ethereum; see also [12] for the NIST-standardized SHA-3 family). While the protocol uses post-quantum signatures for message authentication (see [10]), Keccak256 itself is not a digital signature scheme — it is a hash function. Grover's algorithm reduces the effective preimage resistance of a 256-bit hash to approximately 128 bits, which remains within the security margin considered adequate by NIST for post-quantum applications. The randomness derivation does not depend on collision resistance alone; a successful attack would require computing preimages or second preimages, not merely collisions.

Lookback justification. The 512,000-block lookback corresponds to approximately 35 days at the target 6-second block period, or 53 to 178 days at observed mainnet block times (9-30 seconds). This provides a window of several weeks to months during which any validator registration or deregistration event invalidates an attacker's prediction of future committee composition. The primary manipulation resistance comes from the temporal separation: a proposer at block N cannot influence the parent hash that was committed 512,000 blocks in the past.

Bootstrap phase. For blocks before the chain has accumulated 512,000 blocks of consensus-context history, a shorter lookback of 64,000 blocks is used, corresponding to approximately 4 to 22 days depending on block times. The reduced lookback provides lower manipulation resistance during chain infancy, when the validator set is typically more volatile and registration events more frequent. The full 512,000-block lookback activates at a fixed configuration-defined block height. After that point, all blocks use the full lookback. Note that the bootstrap phase compounds two independent risk factors: the shorter lookback and the smaller validator set (fewer registered validators). A smaller validator set reduces the combinatorial space of possible committees, making it easier for an adversary with even modest deposit to appear disproportionately often. The protocol accepts this elevated risk as inherent to chain infancy, where the validator set has not yet grown to its steady-state size.

Block-number entropy. The consensus context is the base seed, but it is not used alone. When computing the per-validator hashes that determine committee membership (Section 10.2) and proposer ordering (Section 10.3), the current block number N is concatenated with the consensus context and each validator's address before hashing. This ensures that even if the consensus context is identical for two different blocks — which occurs when neither the lookback parent hash nor the validator count has changed over 512,000 blocks — the resulting validator ordering and committee composition differ at every block height. Without this measure, a static validator set would produce the same committee and proposer sequence periodically; the block number breaks this degeneracy.

10.2 Validator Selection

For each block, a committee of up to 128 validators is selected from the full registered set, which may contain any number of validators [3]. The 128 limit is a per-block communication-complexity bound, not a cap on network participation. All registered validators remain eligible for selection in future blocks, subject to the offline weight reduction and exclusion rules (Section 11.3, Section 11.5); committee membership rotates at every

block height. The committee is computed from on-chain state at the parent block and is fixed for the duration of that block's consensus; validator registrations or deregistrations that occur while consensus is in progress take effect at the next block. Because the committee is a subset of the registered set and voting is deposit-weighted, committee selection cannot amplify the Byzantine deposit fraction (see Section 7, "Network-level to committee-level translation").

Comparison with other committee-based protocols. This rotating committee approach is architecturally comparable to Algorand's [14] per-step sortition committees, which similarly select a bounded subset from an unbounded participant set via VRF-based cryptographic sortition (committee sizes are parameterized per step and vary by protocol phase). The key difference is the selection mechanism: Algorand uses private VRF evaluation (committee membership is unknown until revealed), while QuantumCoin uses deterministic selection from on-chain state (committee membership is publicly computable). Both approaches bound per-block communication complexity while permitting unbounded network participation. Ethereum uses a different model: the full active validator set is divided into per-slot beacon committees across each 32-slot epoch, with up to 64 committees per slot and a target minimum of 128 validators per committee. Each validator attests once per epoch in its assigned slot. This guarantees every active validator exactly one attestation opportunity per epoch, distributing participation uniformly across slots, at the cost of higher aggregate communication overhead compared to QuantumCoin's single 128-validator committee per block.

If fewer than 128 validators are registered and eligible, all are selected. If more than 128 are registered, the consensus randomness (Section 10.1) is used to select a subset through a three-pass algorithm designed to balance deposit representation with decentralization.

The selection process applies the following filters in order:

1. **Minimum deposit.** Validators with deposit below the minimum threshold are excluded.
2. **Offline weight reduction and exclusion.** Each remaining validator's effective deposit is reduced according to its NilBlockCount (see Section 11.3). Validators whose effective deposit falls below the minimum threshold after reduction are excluded. At NilBlockCount = 50, the penalty reaches 100% and effective deposit is zero, triggering permanent exclusion from the filtered validator set (see Section 11.5).
3. **Committee selection.** If more than 128 eligible validators remain, a three-pass deterministic algorithm reduces the set to 128 while targeting a high fraction of total network deposit in the committee:

- **Pass 1 — Deposit-weighted selection (up to 42 slots).** Validators are sorted by deposit in descending order. The algorithm selects validators from the top of this list until either 42 validators have been selected or the cumulative deposit of selected validators exceeds a target fraction of the total network deposit — whichever comes first. The current protocol rule sets this target at **85%**. The theoretical minimum for a two-thirds committee quorum to represent a simple majority of network deposit is 76% (since 76% × 67% ≈ 51%). The protocol uses 85% rather than 76% to provide margin for the effects of deposit normalization (Section 12), which can reduce effective honest deposit through clipping and redistribution. At 85%, a two-thirds quorum within the committee controls at least 85% × 67% ≈ 57% of the entire network's deposit — well above a simple majority even after normalization effects. This is a best-effort target: if deposit is widely distributed across many validators, the 42-slot cap may be reached before the deposit threshold. In

practice, the design significantly increases the fraction of total network deposit represented in the committee compared to uniform random selection.

- **Pass 2 — Coin-flip weighted random selection (up to 84 − Pass1Count additional slots).** The remaining validators, still in deposit-descending order from Pass 1, are each subjected to a cryptographic coin flip: two Keccak256 [11] hashes are computed — H1 = Keccak256(context ‖ address ‖ blockNumber) and H2 = Keccak256(address ‖ context ‖ blockNumber) — and the validator is included if H1 > H2. The two-hash comparison assumes that Keccak256 behaves as a random oracle: when modeled as an ideal random function, H1 and H2 are independently and uniformly distributed over the output space regardless of input structure, making the comparison unbiased (P(H1 > H2) = P(H1 < H2) ≈ 50%, with a negligible probability of equality for 256-bit outputs). This is a standard cryptographic modeling assumption; it is not a provable property of Keccak256 itself but is consistent with the random oracle model widely used in cryptographic protocol analysis. The total committee size after Pass 2 is capped at 84; if Pass 1 selected k validators, Pass 2 adds up to 84 − k. Processing in deposit-descending order means validators with more deposit are evaluated first and are more likely to fill available slots before the cap is reached.
- **Pass 3 — Pseudorandom fill (128 − Pass2Count remaining slots).** All remaining validators are sorted by a purely pseudorandom hash derived from the consensus context, the validator's address, and the current block number. Validators are added in hash order until the committee reaches 128 members. The number of Pass 3 slots is exactly 128 minus the number of validators selected in Passes 1 and 2 (typically around 44 when both passes fill their allocations). This pass ensures that validators with smaller deposits have a fair opportunity for inclusion regardless of deposit size.

The three-pass design addresses a tension present in single-mechanism committee selection algorithms: ensuring that the committee controls a large fraction of total network deposit (security) while simultaneously ensuring that small-deposit validators are not permanently excluded (decentralization). Most committee-based protocols use a single selection mechanism — Algorand uses VRF-based sortition (probabilistic, deposit-weighted), Ethereum divides validators into per-slot beacon committees via RANDAO-based shuffling (each validator attests once per epoch in an assigned slot), and Tendermint uses weighted round-robin proposer selection with an application-controlled validator set that can update at each block height. QuantumCoin's three-pass algorithm is a bounded-committee algorithm that compositionally combines three distinct selection strategies (deterministic deposit-ranked, probabilistic coin-flip, and pseudorandom fill), each targeting a different portion of the deposit distribution. This hybrid design targets capturing at least 85% of network deposit (via Pass 1) while rotating remaining seats among the full validator population (via Passes 2 and 3). Committee membership rotates at every block height because the pseudorandom hashes depend on the block number (Section 10.1).

Selection probability by scenario. The following tables illustrate how the three-pass algorithm distributes committee membership and proposer probability across four representative scenarios. Within the committee, each member has equal probability of becoming a proposer (1/128); the overall proposer probability for a validator is its committee selection probability multiplied by 1/128. These values have been verified by Monte Carlo simulation over thousands of blocks (see filterValidators_test.go). Approximation caveat: the per-pass probabilities listed below are approximate. Pass 2's early-stopping seat cap means

that a validator's selection probability depends on how many validators before it (in deposit-descending order) passed their coin flip; Pass 3's pool composition depends on the random Pass 2 outcome. The tabulated values reflect the expected-case behavior observed in simulation, not exact closed-form probabilities.

Scenario A — Widely distributed deposit (1,000 validators). Pass 1 reaches the 42-slot cap before hitting the deposit threshold.

Pass	Validators considered	Slots filled	Prob. of committee selection	Overall prob. of being block proposer
Pass 1	1,000	42	100%	1/128 = 0.78125%
Pass 2	~84	Up to 42	~50%	0.5 × 1/128 ≈ 0.39063%
Pass 3	916	~44	~4.8% (44/916)	44/916 × 1/128 ≈ 0.03753%

Pass 2 considers validators in deposit-descending order (excluding Pass 1 selections); with a ~50% coin flip, ~84 candidates are evaluated before the 42-seat cap is reached. Pass 3 considers all 916 remaining unselected validators (1,000 − 42 Pass 1 − 42 Pass 2). Pass 2 candidates who lost their coin flip remain eligible for Pass 3.

Scenario B — Highly concentrated deposit (1,000 validators). The top 18 validators hold more than 85% of total deposit; Pass 1 stops at the deposit threshold.

Pass	Validators considered	Slots filled	Prob. of committee selection	Overall prob. of being block proposer
Pass 1	1,000	18	100%	1/128 = 0.78125%
Pass 2	~132	Up to 66	~50%	0.5 × 1/128 ≈ 0.39063%
Pass 3	916	~44	~4.8% (44/916)	44/916 × 1/128 ≈ 0.03753%

Pass 2 considers validators in deposit-descending order (excluding Pass 1 selections); with a ~50% coin flip, ~132 candidates are evaluated before the 66-seat cap is reached. Pass 3 considers all 916 remaining unselected validators (1,000 − 18 Pass 1 − 66 Pass 2). Pass 2 candidates who lost their coin flip remain eligible for Pass 3.

Scenario C — Large validator set (100,000 validators). Pass 1 fills all 42 slots. Pass 3 validators have very low individual selection probability but rotate into the committee over time.

Pass	Validators considered	Slots filled	Prob. of committee selection	Overall prob. of being block proposer
Pass 1	100,000	42	100%	1/128 = 0.78125%
Pass 2	~84	Up to 42	~50%	0.5 × 1/128 ≈ 0.39063%
Pass 3	99,916	~44	~0.04% (44/99,916)	44/99,916 × 1/128 ≈ 0.00034%

Pass 2 considers validators in deposit-descending order (excluding Pass 1 selections); with a ~50% coin flip, ~84 candidates are evaluated before the 42-seat cap is reached. Pass 3 considers all 99,916 remaining unselected validators (100,000 − 42 Pass 1 − 42 Pass 2). Pass 2 candidates who lost their coin flip remain eligible for Pass 3.

Scenario D — Small validator set (128 or fewer eligible validators). All validators are selected;

the three-pass algorithm is not invoked.

Pass	Validators considered	Slots filled	Prob. of committee selection	Overall prob. of being block proposer
All	N	N (all eligible)	100%	1/N

Comparison with equal-weight-per-validator schemes. In protocols where each committee member has equal voting weight regardless of deposit, a two-thirds committee quorum may represent an arbitrarily small fraction of total network deposit if the committee is sampled uniformly. QuantumCoin's deposit-weighted voting combined with the Pass 1 deposit-representation target means committee consensus is far more likely to reflect majority economic backing at the network level.

Proposition 3 (Committee representativeness bound). Let F denote the Byzantine fraction of total network deposit ($F < 1/3$). Under the three-pass selection algorithm, the expected Byzantine fraction of the committee's pre-normalization deposit is at most F.

Proof. Committee selection does not create deposit — it selects a subset of existing validators and preserves their deposit weights. An attacker controlling fraction F of total network deposit controls at most deposit $F \times D\_total$ across all their identities. In Pass 1, validators are selected deterministically by deposit rank; any Byzantine validators selected contribute at most their actual deposit (deterministic, no variance). In Pass 2, each remaining validator passes an independent coin flip (~50% probability); the expected Byzantine deposit selected is at most F times the expected total Pass 2 deposit. In Pass 3, validators are sorted by a pseudorandom hash; the selected subset is a uniform random sample of remaining validators, with expected Byzantine fraction F. Since each pass selects validators with their actual deposit weights and no pass creates or amplifies deposit, the expected aggregate Byzantine deposit in the committee cannot exceed $F \times D\_total$. The expected committee Byzantine fraction is therefore at most F. ∎

Note. This is an expectation bound. For any specific block, the realized committee Byzantine fraction may exceed F due to sampling variance in Passes 2 and 3. Proposition 4 below provides a tail bound on the probability of such deviations. Post-normalization, the effective Byzantine fraction may increase further due to redistribution (see Proposition 2, Section 12). Monte Carlo simulations (see filterValidators_test.go) confirm that the empirical committee Byzantine fraction matches the network-level fraction in expectation across thousands of blocks.

Proposition 4 (Per-block committee Byzantine fraction tail bound). Let F denote the network-level Byzantine deposit fraction ($F < 1/3$). Let B_1 denote the Byzantine deposit selected in Pass 1 (deterministic), and let B_2, B_3 denote the Byzantine deposit selected in Passes 2 and 3 respectively (random). Let D_c denote the total committee deposit. The probability that the committee Byzantine fraction exceeds 1/3 satisfies:

$$\Pr[(B_1 + B_2 + B_3) / D_c > 1/3] \leq \Pr[B_2 > E[B_2] + \varepsilon_2] + \Pr[B_3 > E[B_3] + \varepsilon_3]$$

where $\varepsilon_2 + \varepsilon_3 = (1/3 - F) \times E[D_c] - (B_1 - F \times E[D_1])$, and D_1 is the Pass 1 total deposit.

Proof sketch. Pass 1 is deterministic: Byzantine deposit B_1 is fixed for a given deposit distribution. In Pass 2, each validator is independently selected with probability ~1/2. Let X_i be the indicator for validator i being selected, and d_i its deposit. The Pass 2 Byzantine

deposit is $B_2 = \Sigma_{i \in Byz} X_i \times d_i$, a sum of independent random variables bounded in $[0, d_i]$. By the Hoeffding bound: $Pr[B_2 > E[B_2] + \varepsilon_2] \leq \exp(-2\varepsilon_2^2 / \Sigma_{i \in Byz} d_i^2)$. Qualification: Pass 2 processes validators in deposit-descending order and stops when the cumulative seat count reaches $84 - Pass1Count$. This early-stopping truncation means that not all Pass 2 candidates are evaluated independently; validators processed after the cap is reached are deterministically excluded. The Hoeffding bound, which assumes all coin flips are independent, is therefore conservative (an upper bound) when the seat cap binds — the actual Pass 2 Byzantine deposit variance is lower than the independent-flip model predicts. In Pass 3, validators are selected by pseudorandom hash ordering (equivalent to sampling without replacement from the remaining pool). The Pass 3 Byzantine deposit B_3 follows a hypergeometric-like distribution. By the standard hypergeometric tail bound: $Pr[B_3 > E[B_3] + \varepsilon_3] \leq \exp(-2\varepsilon_3^2 / (s_3 \times d\_max^2))$, where s_3 is the number of Pass 3 slots and d_max is the maximum single-validator deposit in the Pass 3 pool. Combining via the union bound yields the stated inequality. The bound is tightest (smallest deviation probability) when the Byzantine deposit is distributed across many small-deposit identities (reducing d_max and Σd_i^2) — which is also the Sybil-heavy scenario most relevant to security analysis. ∎

Methodological note. Proposition 4 is a heuristic approximation, not a rigorous concentration theorem. The Hoeffding bound assumes independent coin flips, but Pass 2's early-stopping seat cap introduces dependence (validators processed after the cap binds are deterministically excluded). The union bound over Passes 2 and 3 is loose when the two deviation events are correlated. Furthermore, the treatment of Pass 3 as hypergeometric sampling is approximate because the Pass 3 pool composition depends on the (random) Pass 2 outcome. The bound is therefore conservative (an upper bound on the true deviation probability) and is intended to provide practical guidance on the negligibility of per-block Byzantine fraction exceedances, not a formally tight characterization. Monte Carlo validation (see below) confirms that the bound is consistent with empirical behavior.

Practical significance. For a network with $F = 0.20$ and 1,000 registered validators, the probability of the per-block committee Byzantine fraction exceeding 1/3 is negligible ($< 10^{-6}$) under typical deposit distributions. Monte Carlo simulations over thousands of blocks (see filterValidators_test.go) confirm that the empirical committee Byzantine fraction never exceeds the network-level fraction F in any observed block, consistent with the exponential tail decay predicted by the concentration inequalities.

The full selection procedure is specified in [6].

Sybil resistance. The minimum deposit requirement provides economic Sybil resistance. Splitting a large deposit across many minimum-deposit identities does not increase total voting weight (all quorum thresholds are deposit-weighted, not validator-count-weighted) and incurs registration gas costs for each identity. The three-pass algorithm's deposit-descending processing further reduces the benefit of splitting: a single high-deposit validator is more likely to be selected in Pass 1 or Pass 2 than multiple low-deposit fragments competing for Pass 3 slots.

If the combined deposit of all selected validators is too low to form a valid quorum (below the protocol's minimum block deposit requirement of 500 billion coins, which is 0.5% of the max supply), or if the total deposit across all registered validators is below this minimum, block production is stalled. Since validator registrations require on-chain transactions, recovery from this state requires off-chain coordination (e.g., a protocol upgrade). This guard also prevents the degenerate case where penalties and exclusions reduce total effective deposit to

zero.

10.3 Proposer Selection

For each round within a block, a single proposer is selected from the filtered validator set. The selection is deterministic, using the consensus randomness, the round number, and the current block number as inputs. Proposer selection is not weighted by deposit beyond the initial filtering — once a validator is in the filtered set, each validator has an equal probability of being selected as proposer, determined by a pseudorandom ordering derived from the consensus context hash mixed with each validator's address, the round number, and the block number (see Section 10.1, "Block-number entropy").

Validators that are currently deferred from proposing due to offline penalties (Section 11.4) are excluded from proposer selection.

An entity that splits its deposit across multiple identities does not increase its total deposit-weighted voting power (all quorum thresholds are deposit-weighted). However, because proposer selection is uniform per committee seat, an entity that obtains k seats in the committee has a per-block proposer probability of k/128, compared to 1/128 for a single-identity entity with the same total deposit.

Remark 1 (Sybil proposer advantage). An entity holding total deposit D_a that splits into m equal identities (each with deposit D_a/m) trades deterministic committee membership for probabilistic multi-seat selection. When D_a is large enough for Pass 1 selection, splitting generally reduces the expected number of committee seats; however, if fragments retain Pass 2 eligibility (i.e., each fragment's deposit is still high enough to be evaluated before the Pass 2 seat cap binds), splitting can increase the expected proposer frequency.

Informal argument. Consider two strategies for an entity with total deposit D_a: (A) register as a single validator, or (B) split into m validators with D_a/m each. Under strategy A, if D_a is large enough for Pass 1 selection (one of the top 42 by deposit), the entity is deterministically selected and has proposer probability 1/128. Under strategy B, each fragment has a smaller deposit and may fall below the Pass 1 threshold. If all m fragments fall to Pass 2, each has ~50% selection probability, yielding expected committee seats m/2 and expected proposer probability m/(2 × 128). For m > 2, this exceeds 1/128: the Sybil entity gains proposer frequency at the cost of per-proposal deposit weight (relevant for normalization and redistribution capture) and the loss of deterministic Pass 1 selection. This gain is contingent on all m fragments being processed before the Pass 2 seat cap binds; if the cap is reached before some fragments are evaluated, those fragments are deterministically excluded, reducing the effective gain. If D_a is small enough that the single identity is already in Pass 2 or Pass 3, splitting provides no systematic advantage: m coin flips replace one coin flip, and the expected number of committee seats scales linearly with m only in Pass 2, while Pass 3 selection probability per identity is inversely proportional to the remaining validator count. Additionally, each Sybil identity incurs registration gas costs and minimum deposit lock-up.

The practical gain is mitigated by the three-pass selection algorithm: Pass 1 selects the top 42 validators by deposit, so splitting a large deposit into small fragments moves identities from the deterministic Pass 1 into the probabilistic Pass 2/3 pools. While this can increase expected proposer frequency as described above, it sacrifices guaranteed committee membership and deposit weight. Additionally, the proposer role carries direct economic reward (block reward plus 50% of transaction fees, per Section 11.7), but these rewards are contingent on

successful proposal acceptance and do not scale with deposit weight.

10.4 Resistance to Validator Set Manipulation

A common attack vector in proof-of-stake protocols is for the current proposer to manipulate block parameters (timestamp, transaction inclusion, state changes) to influence the validator set of future blocks. QuantumCoin mitigates this through three mechanisms:

Lookback-based randomness. The consensus context for block N depends on the parent hash of block N − 512,000. This hash was determined hundreds of thousands of blocks in the past by a different proposer. The current proposer has no ability to alter it. A proposer at block N could in principle attempt to influence the consensus context of block N + 512,000, but any new validator registration between blocks N and N + 512,000 changes the validator count input, altering the prediction. The consensus context is stored as a field in the blockchain's state when each block is finalized, so nodes do not need to have the actual lookback block available locally — they read the pre-computed value from the current state.

Validator count entropy. The total registered validator count is mixed into the randomness hash. This count changes whenever any validator registers or deregisters. Since the proposer cannot control other participants' registration decisions, the randomness source includes an input that is outside any single proposer's control. The validator count is a slowly-changing integer and does not provide high entropy in isolation — an attacker could enumerate all plausible counts and pre-compute the resulting selections. Its value is that any change at all invalidates the entire prediction. The primary manipulation resistance comes from the 512,000-block lookback, not from the validator count alone. Attempting to change the validator count by registering or deregistering validators is further constrained by the minimum deposit requirement per validator and the 32,000-block withdrawal delay (Section 10.5), which make count-enumeration attacks capital-intensive and rate-limited.

Block-number mixing. In addition to the consensus context, the current block number is concatenated into every per-validator hash used during committee selection and proposer ordering. This provides a defense against periodicity: if the validator set is completely static (no registrations or deregistrations) for 512,000 consecutive blocks, the consensus context would repeat because both its inputs — the lookback parent hash and the validator count — would cycle. The block number is strictly monotonic and therefore never repeats, guaranteeing that the committee composition and proposer ordering are unique at every block height regardless of validator set turnover.

Comparison with RANDAO. Ethereum's beacon chain uses a RANDAO accumulator [2] for proposer and committee selection randomness. Each block proposer contributes a BLS signature over the current epoch number, which is mixed into a global RANDAO value via XOR. The RANDAO seed at the end of epoch N determines validator duties for epoch N + 2 (a two-epoch lookahead).

This design has a known bias vector: a proposer assigned to the last slot of an epoch can choose to either publish its block (mixing in its RANDAO reveal) or withhold it (forfeiting the block reward but preserving the current RANDAO value). This gives the proposer a binary choice over two possible RANDAO outcomes for the next epoch — commonly referred to as "one bit of influence." Recent research [9] has identified a more severe variant, the RANDAO forking attack, in which an attacker selectively forks out honest proposers' blocks to control the RANDAO output without the cost of withholding its own blocks.

QuantumCoin's approach avoids both classes of RANDAO bias. The randomness inputs (parent hash from 512,000 blocks ago and current validator count) are already committed on-chain and cannot be selectively revealed or withheld by the current proposer. There is no accumulator that the proposer can influence by choosing whether to publish a block.

The trade-off is adversarial predictability: since both inputs are deterministic and publicly observable, the proposer for a future block can in principle be predicted by any observer, including potential attackers, once both inputs are known. However, the validator count input changes with every registration or deregistration event, limiting the practical prediction horizon. A proposer at block N could attempt to influence the consensus context of block N + 512,000, but any validator registration change in the intervening 512,000 blocks alters the randomness, making long-range manipulation impractical. Additionally, because committee membership rotates at every block height (Section 10.2), a predicted proposer for block N+k faces a different committee than the current block, limiting the utility of proposer targeting for consensus manipulation.

Long-range attacks. As with all proof-of-stake protocols, clients bootstrapping from genesis must obtain a trusted checkpoint or rely on social consensus to identify the canonical chain. A historical attacker who controlled more than 1/3 of deposit at some past block could in principle create a competing fork from that point. The protocol's immediate deterministic finality mitigates this for clients that observe finalization in real time: once a block is finalized, it is irreversible by design, and no fork-choice rule exists that could prefer an alternative chain. Bootstrapping clients that cannot verify finalization in real time should use a recent trusted checkpoint.

10.5 Withdrawal Delay and Re-Registration Restrictions

A potential attack on deposit-weighted randomness is rapid withdraw-reenter cycling: a validator withdraws its deposit and re-registers (possibly under a different address) to change the registered validator count, which is an input to the consensus randomness hash (Section 10.1). By iterating over different validator-count values, an attacker could search for a count that biases proposer selection favorably — a form of short-range randomness grinding.

The staking contract mitigates this through two mechanisms:

1. **Withdrawal delay.** All withdrawals (partial or full) require a mandatory delay of 32,000 blocks between initiation and completion. During this period, the deposit remains locked and the validator count is unchanged. This delay bounds the rate at which any single entity can cycle through validator-count values: at most one withdraw-reenter cycle per 32,000 blocks, compared to the 512,000-block randomness lookback window (Section 10.1). An attacker would need approximately 16 cycles (512,000 / 32,000) to span one full lookback period, each cycle requiring a fresh minimum deposit.
2. **One-time address binding.** The staking contract permanently records every validator and depositor address that has ever been registered. The registration function rejects any address — validator or depositor — that has previously been registered, even after full withdrawal. Re-entry therefore requires a fresh address pair with no prior staking history. Combined with the minimum deposit requirement, this converts each re-entry attempt into a capital-intensive operation requiring both new addresses and a new minimum deposit lock.

Together, these mechanisms ensure that validator-count grinding is rate-limited by the

withdrawal delay and capital-limited by the minimum deposit, making it economically impractical relative to the marginal influence on proposer selection that a single-unit validator-count change provides (see Section 10.4, "Validator count entropy").

---

11. Offline Validator Penalties

The protocol applies a graduated penalty scheme to validators whose proposals are not received by the network. The penalties are designed to progressively reduce the influence of unreliable validators while avoiding punishing validators for network-wide issues.

11.1 Round Attribution

Penalties are applied **only when a block is finalized at Round 1.** The rationale:

- **Round 1 finalization** indicates that a sufficient supermajority of validators received the proposal (or did not receive it in time and voted NIL) and reached consensus within a single round. If a supermajority voted NIL, this indicates that the **proposer** was likely offline or unreachable — an individual failure.
- **Round 2 finalization** indicates that Round 1 failed to reach the two-thirds threshold for any vote type, meaning honest validators were split between OK and NIL. This is characteristic of a **network-level** issue (e.g., partial connectivity, high latency) not necessarily an individual proposer failure. No penalties are applied.

This distinction prevents penalizing proposers for conditions outside their control. A consequence is that a Byzantine coalition controlling just under 1/3 of deposit could strategically split votes to force Round 2, shielding a co-conspirator proposer from penalties. This is a known tradeoff: attributing penalties at Round 2 would risk penalizing honest proposers during genuine network degradation, which would be more harmful to the protocol's long-term health than the penalty-avoidance attack.

11.2 Deposit Deduction

When a block is finalized at Round 1 with vote type NIL, the proposer whose proposal was not received is identified. A deployment-configured fixed amount is deducted from that proposer's on-chain deposit. The deducted amount is transferred to address(0) (the zero address), permanently removing the coins from circulation. The staking contract records the cumulative slash amount in a per-depositor ledger (_depositorSlashings), which is subtracted from the depositor's balance when computing the net deposit available for withdrawal. This is a direct economic penalty. Consistent with the round attribution principle (Section 11.1), no deposit deduction occurs when a block finalizes at Round 2, because Round 2 finalization indicates a network-level issue rather than individual proposer fault.

11.3 Progressive Voting Weight Reduction

Each Round 1 NIL finalization increments an on-chain counter (NilBlockCount) for the attributed proposer. This counter drives a progressive reduction of the validator's effective voting weight:

```
EffectiveDeposit = Deposit × (1 − NilBlockCount × 2%)    if NilBlockCount > 2
EffectiveDeposit = Deposit                                if NilBlockCount ≤ 2
```

- The first 2 nil blocks incur no weight reduction (grace period). A validator with NilBlockCount of 1 or 2 retains its full deposit.
- Starting at NilBlockCount = 3, the total reduction is NilBlockCount × 2 percentage points. At NilBlockCount = 3 the total reduction is 6%; at NilBlockCount = 4 it is 8%; at NilBlockCount = 10 it is 20%.
- At NilBlockCount = 50, the reduction is 50 × 2% = 100%, reducing the effective deposit to zero. Because the penalty percentage reaches 100%, the implementation returns zero effective deposit directly (a penaltyPercent >= 100 guard), and the validator is excluded from the filtered set (Section 11.5). The formula's natural zero-crossing is at NilBlockCount = 50.

This reduction applies during validator filtering (Section 10.2) and affects the validator's contribution to all quorum calculations. The weight reduction is not a permanent change to the on-chain deposit — it is computed dynamically during each block's validator selection.

Boundary discontinuity. The grace period creates a discrete jump at NilBlockCount = 3: effective deposit drops from 100% (at NilBlockCount = 2) to 94% (at NilBlockCount = 3), a 6-percentage-point cliff. After this boundary, each additional NIL incurs a marginal cost of only 2 percentage points. This is a deliberate design choice: the first two NIL attributions are treated as transient (possibly caused by brief connectivity issues) and incur no weight penalty; the third attribution signals a pattern and triggers the cumulative penalty for all three. The cliff creates a strong incentive to resolve connectivity issues after the first NIL attribution, before reaching the threshold. A validator at NilBlockCount = 2 that successfully proposes an OK block resets to zero (Section 11.6), avoiding the cliff entirely.

11.4 Proposer Deferral

Validators with a history of being offline when selected as proposer are deferred from proposer eligibility for a number of blocks that increases exponentially with the offline count:

```
SlotsMissed = floor(NilBlockCount / 2)    (integer division; capped at 16)
BlockDelay = 2^SlotsMissed                (capped at 65,536)
if NilBlockCount > 1:
    BlockDelay = BlockDelay + 3,600       (additive minimum deferral)
```

The validator may not be selected as proposer until at least BlockDelay blocks have elapsed since their last nil block attribution. The additive 3,600-block term ensures that even validators with low NilBlockCount (2 or 3) are deferred for a meaningful period once they have missed more than one proposal. This ensures that repeatedly offline validators are excluded from proposer rotation for progressively longer periods.

Deferral cliff. The additive 3,600-block floor creates a sharp transition at NilBlockCount = 2: deferral jumps from 1 block (at NilBlockCount = 1) to 3,602 blocks (at NilBlockCount = 2), corresponding to approximately 9 to 30 hours at observed block times. This is intentional: a single NIL attribution is treated as a transient event warranting minimal deferral, while a second attribution indicates a pattern of unavailability that warrants a meaningful exclusion period. The jump ensures that validators with recurring connectivity problems are removed from proposer rotation for long enough to allow the operator to diagnose and resolve the issue, rather than repeatedly wasting proposer slots on an unreachable node. They are however still eligible to be validators, albeit with the voting weight decay described in Section 11.3.

11.5 Validator Exclusion

At NilBlockCount = 50, the progressive weight reduction (Section 11.3) reduces the validator's effective deposit to zero (since 50 × 2% = 100%). The implementation enforces this via a penaltyPercent >= 100 guard that returns zero effective deposit. A zero effective deposit falls below the minimum deposit threshold, causing exclusion from the filtered validator set by Section 10.2 step 2. This exclusion is permanent for the lifetime of the current registration: since NilBlockCount can only be reset by successfully proposing an OK block (Section 11.6), and the validator cannot be selected as proposer while excluded from the filtered set, there is no recovery path once NilBlockCount reaches 50. The validator must deregister and re-register with a fresh deposit to participate again. Upon deregistration, the original deposit (minus any amounts previously deducted under Section 11.2) is returned to the validator.

Honest validator exclusion under prolonged partition. The penalty system does not distinguish between a genuinely malicious validator and an honestly operated validator that is unreachable due to a prolonged network partition. If an honest validator is repeatedly selected as proposer during a partition, it will accumulate NilBlockCount increments and eventually reach permanent exclusion at NilBlockCount = 50 — despite having no Byzantine intent. This is a deliberate design trade-off: the protocol prioritizes protecting the network's liveness and throughput over accommodating validators with persistent connectivity failures. Honest validators operating behind unreliable network links bear the risk of exclusion; upon partition recovery, they must deregister and re-register (incurring no economic loss beyond the deposit deductions already applied) to resume participation.

Protocol-level tolerance of targeted DDoS attacks. An adversary may attempt to DDoS a specific validator to make it unreachable, causing NIL block finalizations, triggering penalties, and eventually causing the validator to be removed from the eligible validator set. The penalty system's gradualism provides built-in resilience against this attack vector. The voting weight reduction is incremental (2% per NIL attribution beyond the grace period), taking 50 NIL block attributions to reach permanent exclusion. Furthermore, after each NIL attribution, the proposer deferral mechanism (Section 11.4) excludes the targeted validator from proposer selection for a progressively longer period (exponential backoff plus an additive 3,600-block floor after the second attribution). This deferral window — ranging from hours to days at observed block times — gives the validator operator time to detect the attack, mitigate it, and restore connectivity before the next proposer selection. If the validator recovers and successfully proposes an OK block before reaching NilBlockCount = 50, its NilBlockCount is reset to zero (Section 11.6), restoring full voting weight and clearing the proposer deferral. Deposit deductions already applied under Section 11.2 are not reversed.

11.6 Recovery

A validator's nil block counter is **reset to zero** when it successfully proposes a block that is finalized with vote type OK at Round 1. This provides a recovery path for validators with NilBlockCount below 50: once a validator demonstrates that it is back online and reachable by successfully producing a block, its NilBlockCount is reset to zero, restoring full voting weight and clearing the proposer deferral. Deposit deductions already applied under Section 11.2 are permanent and are not reversed by this reset. Validators whose NilBlockCount has reached 50 cannot recover through this mechanism because their zero effective deposit prevents them from being selected (see Section 11.5).

11.7 Elimination of Passive Staking Incentives

In many proof-of-stake protocols, validators accumulate staking yield proportional to their locked deposit irrespective of operational contribution to consensus — a property variously termed passive staking, free-riding, or the lazy validator problem. Under such designs, the economically rational strategy is to lock capital and collect yield without incurring the operational cost of running a validator node, degrading the effective set of active consensus participants.

QuantumCoin eliminates this incentive class through two reinforcing mechanisms:

1. Work-contingent reward distribution. The protocol distributes block rewards only to the proposer of a block that finalizes with vote type OK. The block reward comprises an inflation component computed by a halving schedule driven by block number: the nominal annual reward rate starts at 5% of total supply (assuming the nominal 6-second block period) and halves every 4 nominal years, where one nominal year is defined as 5,256,000 blocks. Because observed mainnet block times are typically 9–30 seconds rather than the nominal 6 seconds, the effective wall-clock halving interval is proportionally longer (e.g., at an observed mean block time of 24 seconds, the effective wall-clock time to reach the first halving is approximately 16 years rather than 4). In addition, the proposer receives 50% of the aggregate transaction fees (gas cost) included in the block; the remaining 50% is burned. Blocks that finalize with vote type NIL — including all Round 2 blocks and Round 1 blocks where the proposer's proposal was not accepted — produce zero reward. Non-proposer committee members receive no compensation for voting. Consequently, the expected per-block reward for a registered validator is conditioned on (a) selection into the per-block committee (Section 10.2), (b) selection as proposer within that committee (Section 10.3), and (c) successful proposal acceptance by a two-thirds deposit-weighted quorum.

2. Proposer-attributed penalty escalation. When a block finalizes as NIL at Round 1, the protocol attributes the failure to the designated proposer and applies the graduated penalty scheme specified in Sections 11.1 through 11.6:

- Deposit deduction (Section 11.2): a fixed amount (deployment-configured; 100 coins on mainnet) is deducted from the proposer's on-chain deposit per Round 1 NIL attribution.
- Progressive voting weight reduction (Section 11.3): effective deposit is reduced by NilBlockCount × 2 percentage points (after a 2-block grace period), directly diminishing the validator's contribution to quorum calculations and its probability of committee selection in deposit-weighted passes (Section 10.2, Pass 1).
- Proposer deferral (Section 11.4): the validator is excluded from proposer eligibility for an exponentially increasing number of blocks.
- Permanent exclusion (Section 11.5): at NilBlockCount = 50, effective deposit is reduced to zero (since 50 × 2% = 100%) and the validator is removed from the filtered set with no on-chain recovery path.

Proposition 6 (Offline penalty trajectory). An offline validator is permanently excluded after at most 50 proposer selections, with monotonically decreasing effective deposit.

Proof. Let n denote the validator's NilBlockCount (initially 0). Each Round 1 NIL attribution increments n by 1 and applies:

- Deposit deduction: 100 coins (fixed per attribution).
- Effective deposit multiplier: $\max(0, 1 - n \times 0.02)$ for $n > 2$; multiplier is 1 for $n \leq 2$.

The effective deposit as a function of n is $E(n) = (D_0 - 100n) \times \max(0, 1 - n \times 0.02)$ for $n > 2$, and $E(n) = D_0 - 100n$ for $n \leq 2$, where D_0 is the initial deposit. The multiplier reaches zero at $n = 50$ (since $1 - 50 \times 0.02 = 0$), and the implementation returns zero effective deposit directly when the penalty percentage reaches or exceeds 100%. At $n = 50$, effective deposit is zero regardless of D_0. The sequence $E(0), E(1), \ldots, E(50)$ is strictly decreasing for all $n \geq 3$ (since both the deduction and the multiplier decrease), so recovery is impossible without producing an OK block. ∎

Expected value analysis. Consider a validator V that registers a deposit of 100,000,000 coins (20× the minimum) and subsequently ceases operation. Each time V is deterministically selected as block proposer (probability 1/128 per block in which V is a committee member), its proposal will not be received by the network, producing a Round 1 NIL finalization attributed to V. Let p_s denote the per-block probability of V being selected into the committee. The expected number of blocks until first proposer selection is 128 / p_s. After each attributed NIL, V incurs a fixed deposit deduction of 100 coins and its NilBlockCount increments, triggering progressive voting weight reduction (Section 11.3). While the cumulative deposit deduction is modest in absolute terms (50 × 100 = 5,000 coins, or 0.005% of the initial deposit), the dominant economic damage is the progressive weight reduction: at NilBlockCount = 3 the validator's effective deposit is reduced by 6%, at NilBlockCount = 10 by 20%, and at NilBlockCount = 25 by 50%. This weight reduction directly diminishes the validator's probability of committee selection in deposit-weighted passes (Section 10.2, Pass 1), compounding the penalty trajectory. The process is monotonically worsening: no mechanism exists to recover NilBlockCount without successfully proposing an OK block, which an offline validator cannot do. At NilBlockCount = 50, effective deposit is reduced to zero (since 50 × 2% = 100%), and V is permanently excluded from the validator set (see Proposition 6). Validators with smaller deposits may be excluded before NilBlockCount = 50 if the combination of deposit deductions and weight reduction causes their effective deposit to fall below the minimum threshold earlier. Upon deregistration, V recovers its original deposit minus the cumulative deductions (5,000 coins), but the opportunity cost of exclusion — forfeited block rewards and transaction fees over the penalty accumulation period — constitutes the primary economic loss.

Scope of penalty attribution. Penalties are attributed exclusively to the proposer role. A validator selected as a non-proposer committee member that fails to cast votes is not individually penalized; the protocol tolerates such omissions provided the participating deposit exceeds the two-thirds quorum threshold. This design targets the failure mode with direct impact on block production — proposer unavailability — rather than attempting to penalize all forms of non-participation, which would require additional attribution mechanisms and risk penalizing validators experiencing transient network conditions indistinguishable from Byzantine behavior.

Interaction with deposit normalization. Deposit normalization (Section 12) affects voting weight but not proposer selection probability, since proposer selection is uniform across committee seats (Section 10.3). A validator whose effective deposit is clipped from 30% to 10% retains the same 1/128 proposer probability but contributes less voting weight; conversely, a redistribution recipient gains voting weight without gaining proposer probability.

Proposition 5 (Incentive compatibility under normalization). Under the current reward structure, an online validator's expected per-block reward is independent of its post-normalization deposit weight and depends only on committee selection probability and

proposer selection probability. Therefore, normalization does not alter the incentive to remain online.

Argument. Block rewards (inflation + 50% fees) are paid exclusively to the proposer of an OK-finalized block (Section 11.7, item 1). The reward amount is determined by the halving schedule and the block's transaction fees, not by the proposer's deposit weight. Proposer selection is uniform across committee seats (1/128 per member, Section 10.3). Thus the expected per-block proposer reward for committee member i is R/128, where R is the expected block reward, regardless of i's post-normalization deposit. Normalization affects which validators are selected into the committee (via Pass 1's deposit-ranking) and their voting weight within the committee, but not their proposer probability once selected. Since reward is contingent on proposer selection and online presence (not on voting weight), the incentive to remain online and propose blocks is preserved for both clipped and non-clipped validators. ∎

This asymmetry does not undermine the passive-staking argument — both clipped and non-clipped validators must still be online and produce accepted proposals to earn rewards. The normalization mechanism does, however, create a secondary incentive asymmetry: a clipped validator contributes disproportionately less voting weight per unit of locked capital compared to its pre-normalization share, which may affect rational deposit-sizing decisions. Analysis of deposit-sizing equilibria under normalization is deferred to a future economic analysis document.

Non-proposer voting incentives. Although non-proposer committee members receive no direct per-vote compensation, they retain a strong indirect economic incentive to vote promptly. Block rewards are paid only when blocks finalize with vote type OK; if non-proposers withhold votes, the acknowledgment phase cannot reach the two-thirds quorum, Round 1 fails, and the block finalizes as NIL with zero reward for all participants. Chronic non-voting degrades the network's OK-block production rate, reducing every validator's expected future proposer revenue — including the non-voter's own. The incentive to vote is therefore not altruistic but self-interested: each validator's expected return from its future proposer selections depends on the chain maintaining a high OK-block finalization rate, which requires prompt voting by all honest committee members. This iterated-game argument assumes validators with a long time horizon; a validator intending to deregister imminently has a weaker indirect incentive. The protocol's primary defense against non-voting is structural rather than economic: the two-thirds quorum threshold tolerates up to one-third of committee deposit abstaining without affecting finalization.

---

12. Deposit Normalization and Redistribution

To prevent a small number of large validators from dominating quorum formation, the protocol applies deposit normalization to the full eligible validator deposit map (all validators that passed minimum-deposit and penalty filtering), not only to the 128-member committee. In the implementation, normalization is applied after committee selection (Section 10.2, step 3) but operates on the full eligible deposit map, not just the selected committee members. The committee members' deposits are then read from the already-normalized map. This means the 10% cap and redistribution are computed against the total deposit of all eligible validators, ensuring consistent deposit values whether a validator is selected into the

committee or not. Normalization is applied only when the eligible set contains at least 12 validators. The threshold of 12 is a conservative margin: with fewer than ~10 equal-deposit validators, every validator necessarily holds ≥ 10% of total deposit in the eligible set, making the cap universally binding rather than targeted at outliers. The threshold of 12 provides headroom for uneven deposit distributions where some validators hold less than 10% even in small sets. All quorum thresholds for the block are then computed against the post-normalization deposits of the selected committee members.

Deposit cap. Any validator whose effective deposit exceeds 10% of the filtered set's total deposit is clipped to that 10% threshold. The excess is removed from that validator's effective deposit for the current block. This adjustment affects only the validator's effective deposit for quorum calculations in the current block; no actual coins are transferred or deducted from the validator's on-chain deposit.

Redistribution. The clipped excess deposit is redistributed among validators that have zero nil blocks (validators with a clean participation record), in proportion to each recipient's effective deposit after penalties. As with the cap, this redistribution is a per-block effective deposit adjustment — no coins change hands. Validators with NilBlockCount > 0 are excluded from receiving any redistribution. If no validator has a zero nil block count, no redistribution occurs and the excess is simply discarded; the total effective deposit for that block is reduced accordingly. Normalization is a single pass: after redistribution, recipients may hold more than 10% of the original total. The cap targets disproportionately large depositors, not the post-redistribution state.

Fault tolerance under normalization. The 1/3 fault tolerance bound applies to post-normalization effective deposits, since all quorum calculations use post-normalization values. Normalization can alter the effective Byzantine fraction in either direction: clipping a large honest validator reduces honest effective deposit, while redistribution to a Byzantine validator with a clean record (NilBlockCount = 0) increases its effective weight. The protocol's safety assumption is that Byzantine validators control less than 1/3 of the **post-normalization committee effective deposit**. Operators should be aware that the pre-normalization and post-normalization Byzantine fractions may differ.

Worst-case amplification. Consider a set of 12 validators with total pre-normalization deposit of 1000 units: one honest validator holds 600 units (60%), 2 Byzantine validators hold 100 units each (200 units, 20%), and 9 honest validators hold the remaining 200 units (~22.2 each, 20%). All 12 validators have NilBlockCount = 0. The large honest validator is clipped to 10% of the pre-normalization total = 100 units; the excess of 500 units is redistributed proportionally among all validators with NilBlockCount = 0 (all 12), in proportion to their effective deposit after clipping. The redistribution denominator (denoted nonOfflineCoinsAfterReduction in the implementation) is the sum of all NilBlockCount = 0 deposits after the clipping pass: 100 (clipped honest) + 200 (Byzantine) + 200 (other honest) = 500 units. Each validator receives excess × (its post-clipping deposit / 500). Byzantine validators receive 500 × (200/500) = 200 units; the clipped honest validator receives 500 × (100/500) = 100 units; the 9 non-clipped honest validators receive 500 × (200/500) = 200 units. Post-normalization deposits: clipped honest = 100 + 100 = 200, Byzantine = 200 + 200 = 400, non-clipped honest = 200 + 200 = 400. Total = 1000 (excess is fully redistributed). **Post-normalization Byzantine fraction = 400/1000 = 40.0%**, which exceeds the 1/3 (33.3%) safety threshold. Pre-normalization Byzantine fraction was 20%. The scenario requires a specific deposit distribution (one dominant honest validator with ≥60% of total

deposit) and Byzantine validators maintaining clean participation records (NilBlockCount = 0), but it is not precluded by the protocol.

Proposition 2 (Normalization amplification upper bound). Let f denote the pre-normalization Byzantine deposit fraction (f < 1/3), let h_clip denote the aggregate pre-normalization deposit fraction held by honest validators that are clipped, and let k denote the number of clipped honest validators. All Byzantine validators have NilBlockCount = 0 (worst case). Let E = (h_clip − 0.1k) × D denote the total clipped excess (where D is total deposit and 0.1 is the per-validator cap fraction). Let R = (1 − h_clip + 0.1k) × D denote the total deposit of the redistribution-eligible pool (all non-excess deposit held by NilBlockCount = 0 validators). The post-normalization Byzantine fraction f' satisfies:

```
f' = f + E × f / R = f × (1 + E / R)
```

When the cap retention 0.1k is negligible relative to h_clip (i.e., few validators hold very large deposits), this simplifies to the upper bound:

```
f' ≤ f / (1 − h_clip)
```

Derivation. Byzantine deposit is fD. The clipped excess E is redistributed proportionally among all validators with NilBlockCount = 0, in proportion to their effective deposit. The redistribution-eligible pool has total deposit R. Byzantine validators hold fD of the pool, so they receive E × (fD / R). Post-normalization Byzantine deposit is fD + E × fD / R = fD × (1 + E/R). Dividing by D yields f' = f × (1 + E/R). Since E = (h_clip − 0.1k)D and R = (1 − h_clip + 0.1k)D, we get f' = f × (1 + (h_clip − 0.1k)/(1 − h_clip + 0.1k)). In the limiting case where 0.1k → 0 (each clipped validator's cap is negligible), E → h_clip × D and R → (1 − h_clip) × D, giving f' → f / (1 − h_clip). Since 0.1k ≥ 0, the exact f' is at most f / (1 − h_clip), so the simplified formula is an upper bound. ∎

Verification against numerical example. In the worst-case example above: f = 0.20, h_clip = 0.60 (the single honest validator holds 600/1000 = 60%), k = 1. E = (0.60 − 0.10) × 1000 = 500. R = (1 − 0.60 + 0.10) × 1000 = 500. (Note: R includes the clipped validator's retained 100 units, plus the 200 Byzantine and 200 non-clipped honest units.) f' = 0.20 × (1 + 500/500) = 0.20 × 2 = 0.40. The numerical example confirms this: the redistribution pool has total post-clipping deposit of 500 (100 clipped-honest + 200 Byzantine + 200 non-clipped honest); Byzantine share = 200/500 = 0.40; Byzantine redistribution = 500 × 0.40 = 200; post-normalization Byzantine deposit = 200 + 200 = 400; Byzantine fraction = 400/1000 = 0.40, matching the formula exactly. The clipped validator's retained cap participates in the redistribution basis (as implemented), which is why the denominator is 500, not 400. The upper bound f/(1 − h_clip) = 0.20/0.40 = 0.50 remains valid and conservative. For safety (f' < 1/3), the sufficient condition is f / (1 − h_clip) < 1/3, equivalently h_clip < 1 − 3f. At f = 0.20, safety requires h_clip < 0.40; at f = 0.10, safety requires h_clip < 0.70.

Risk assessment. There is no on-chain mechanism to detect or prevent the post-normalization Byzantine fraction from exceeding 1/3, because Byzantine identity is unknown to the protocol. Proposition 2 provides a closed-form bound: the normalization mechanism is safe when h_clip < 1 − 3f, where f is the pre-normalization Byzantine fraction and h_clip is the aggregate honest clipped deposit. When the pre-normalization Byzantine fraction is well below 1/3 (e.g., f ≤ 0.10), the protocol tolerates substantial honest deposit concentration (h_clip up to 70%) without violating the safety bound. As f approaches 1/3, the tolerance for honest deposit concentration shrinks to zero. Operators deploying the protocol

should be aware that normalization creates a gap between the pre-normalization and post-normalization fault tolerance bounds, and that the protocol's safety guarantee (Theorem 1) is expressed in terms of the latter.

Fresh-validator redistribution capture. Newly registered validators start with NilBlockCount = 0 and immediately qualify as redistribution recipients. An attacker can observe on-chain state, predict when a large validator will be clipped, and register multiple minimum-deposit Sybil identities in the preceding block to capture a proportional share of the redistributed excess. This attack does not affect safety or liveness — the attacker gains voting weight, not control over finalization (assuming the < 1/3 bound holds post-normalization). However, it dilutes the redistribution share of honest validators with clean records, partially undermining the mechanism's intended function as a reliability incentive. The attack's economic cost is the registration gas fee per Sybil identity plus the minimum deposit lock-up; the gain is a one-block share of the redistribution pool proportional to total Sybil deposit. In practice, the gain is bounded by the Sybil identities' fraction of the redistribution pool, which decreases as the pool grows. A minimum-tenure requirement (e.g., a validator must have participated in at least one block as a committee member before qualifying for redistribution) would close this vector but is not currently implemented.

Redistribution pool exhaustion. In a mature network, if all validators have NilBlockCount > 0, the eligible redistribution pool is empty and the clipped excess is discarded, reducing total effective deposit. This does not affect the 67% quorum ratio, which is computed against the total effective deposit (post-normalization), not the pre-normalization total.

This mechanism achieves two goals:

1. **Decentralization of voting power.** Every validator whose deposit exceeds 10% of the pre-normalization total is clipped to that threshold. This raises the number of colluding entities required to dominate or stall the network: without the cap, a single validator holding more than 33% of deposit could block consensus by withholding votes; with the cap, at least four colluding validators are required to reach the same blocking power. After redistribution, reliable validators with clean records may exceed 10% of the original total, but this increase is spread among multiple recipients rather than concentrated in a few large depositors.
2. **Incentive for uptime.** The redistribution exclusively benefits validators with a clean record. Validators that have been offline receive no share of the redistributed excess, compounding the penalty from the progressive weight reduction (Section 11.3). Reliable validators gain proportionally more voting weight, while unreliable validators lose both their own weight and their share of redistributed excess.

---

13. MEV Analysis and Transaction Ordering

Miner/Maximum Extractable Value (MEV). The block proposer has full discretion over which transactions to include or exclude from a block and receives 50% of transaction fees (Section 11.7), creating a direct economic interest in fee-maximizing inclusion. The protocol does not currently implement proposer-builder separation, commit-reveal schemes, or other MEV mitigation mechanisms for transaction inclusion. Proposer predictability (Section 10.4) may enable pre-positioning by attackers who anticipate the next proposer. Inclusion-level MEV mitigation is a known open problem in blockchain protocol design and is deferred to a

future protocol enhancement. However, the protocol does mitigate ordering-level MEV (front-running, sandwich attacks) through deterministic round-robin transaction execution ordering, described in detail below.

Round-robin execution ordering and sandwich attack mitigation. Most blockchain protocols grant the block producer full control over transaction execution order, making ordering-level MEV (front-running, sandwich attacks) a proposer privilege. QuantumCoin separates transaction selection (proposer-controlled) from transaction execution order (protocol-controlled), a design choice that is distinct from both Ethereum's proposer-ordered execution and Flashbots-style proposer-builder separation (which delegates ordering to a specialized builder rather than fixing it algorithmically). Although the proposer controls which transactions appear in a block, the execution order of included transactions is determined by a deterministic round-robin algorithm that the proposer cannot override. Transactions are grouped by sender address. Addresses are sorted by Keccak256(parentHash, senderAddress) — a deterministic but per-block-unpredictable ordering derived from the parent block hash. The execution cursor then interleaves transactions across addresses in cyclic round-robin fashion: one transaction (at the lowest pending nonce) from each address in sorted order, then a second transaction from each address that has one, and so on. The ordering is cyclic — after the last address in the sorted sequence, the cursor wraps back to the first address for the next round of transactions. This means every address in the sorted list has exactly two neighbors (the last address is adjacent to the first), and no address occupies a privileged "endpoint" position. This interleaving has the following consequences for sandwich attacks:

- **Disrupted adjacency.** A sandwich attack requires three transactions to execute in a specific sequence: attacker front-run → victim swap → attacker back-run. If the attacker and victim use different addresses (the typical case), their transactions are placed in separate round-robin groups separated by all other senders' transactions. The attacker cannot guarantee that their front-running and back-running transactions execute immediately before and after the victim's transaction.
- **Unpredictable sort position.** The per-block sort key depends on the parent hash, which is not known until the previous block is finalized. An attacker cannot pre-compute which address-sort position will place their transactions adjacent to a victim's in a future block.
- **Multi-address attack limitation.** An attacker could attempt to use multiple addresses to increase the probability of landing adjacent to the victim. The following proposition formalizes the bound.

Proposition 7 (Sandwich attack probability bound). Assume Keccak256 behaves as a random oracle (uniform, independent outputs). Let $n \geq 3$ be the number of distinct sender addresses in a block, and let $k \leq n - 1$ be the number of addresses controlled by the attacker. The round-robin execution ordering treats the sorted address list as cyclic (the last address wraps to the first), so every address has exactly two neighbors. The probability that the attacker controls both the address sorted immediately before and the address sorted immediately after a target victim address is:

```
P_sandwich = k(k − 1) / ((n − 1)(n − 2))
```

which satisfies $P_{sandwich} \leq (k / (n - 1))^2$ for all $n \geq 3$, $k \leq n - 1$.

Proof. Under the random oracle model, the sort keys Keccak256(parentHash, address) are independently and uniformly distributed over $\{0,1\}^{256}$ for distinct addresses. The probability of collision is negligible (2^{-256}), so with probability approaching 1 the n addresses have a

unique total ordering. Because the execution order is cyclic, every position (including the first and last in the sorted sequence) has exactly two neighbors. The victim address occupies a uniformly random position in this cyclic ordering. There are $n - 1$ candidates for the slot immediately before the victim; the probability that a specific attacker address occupies this slot is $1/(n - 1)$. Given that one attacker address occupies the slot before the victim, there are $n - 2$ candidates for the slot immediately after the victim; the probability that a different attacker address occupies this slot is $(k - 1)/(n - 2)$. Summing over all k choices for the "before" address: $P_{sandwich} = k \times (1/(n - 1)) \times (k - 1)/(n - 2) = k(k - 1)/((n - 1)(n - 2))$. Since $(k - 1)/(n - 2) \leq k/(n - 1)$ (cross-multiplying: $(k - 1)(n - 1) = kn - k - n + 1 \leq k(n - 2) = kn - 2k$ iff $k \leq n - 1$, which is given), we have $P_{sandwich} = [k/(n - 1)] \times [(k - 1)/(n - 2)] \leq [k/(n - 1)]^2 = (k/(n - 1))^2$. ∎

Example. With $k = 5$ attacker addresses among $n = 100$ total senders: $P_{sandwich} = 5 \times 4 / (99 \times 98) \approx 0.0021$, or about 0.2%. The upper bound gives $(5/99)^2 \approx 0.0026$. Both decrease quadratically with the number of distinct senders in the block.

Residual MEV surface. The round-robin execution order mitigates ordering-based MEV (sandwich attacks, front-running) but does not address inclusion-based MEV: the proposer can still selectively include or exclude transactions based on their content. Additionally, in blocks with very few distinct senders, the round-robin interleaving provides less separation between any two addresses. The proposer itself, as a transaction sender, participates in the same round-robin ordering and cannot self-preferentially reorder its own transactions relative to others.

Hash-only proposals and last-moment injection cost. Block proposals in QuantumCoin contain only the transaction hashes, not full transaction bodies. Receiving validators verify a proposal by checking that every referenced transaction hash exists in their local mempool (having arrived via normal gossip prior to the proposal). If a validator cannot locate one or more referenced transactions, it must wait for them to arrive or vote NIL when its proposal timeout expires. This creates a timing constraint on proposer-injected MEV transactions: if the proposer creates a transaction at proposal time (e.g., a sandwich or back-running transaction) and includes its hash in the proposal without having previously broadcast the transaction to the network, other validators may not find it in their mempools immediately, due to the time taken for network propagation of the transaction. Validators that cannot resolve the hash before timeout will vote NIL. If sufficiently many NIL votes accumulate, the block finalizes as NIL at Round 1, costing the proposer the block reward and incurring graduated penalties (Section 11). The proposer therefore faces a tradeoff: injecting an unseen transaction increases NIL-vote probability, risking both reward forfeiture and penalty attribution. This is a minor timing-based constraint rather than a robust MEV defense — a patient proposer can broadcast the MEV transaction to the gossip network shortly before proposing, allowing it to reach validator mempools in time — but it raises the cost of spontaneous last-moment transaction injection relative to protocols where full transaction bodies are included in proposals and validators encounter them for the first time inside the block.

Remark 2 (NIL-block persistence bound). Under partial synchrony (after GST), a Byzantine coalition controlling deposit fraction $f < 1/3$ cannot cause an unbounded sequence of NIL-finalized blocks.

Informal argument. A NIL finalization at Round 1 requires either (a) the proposer to be offline/Byzantine, or (b) insufficient OK votes within the timeout. Case (a): proposer selection rotates deterministically at each block (Section 10.3). With $f < 1/3$, the expected fraction of

blocks with a Byzantine proposer is at most f < 1/3. After each Round 1 NIL attributed to a Byzantine proposer, that proposer accumulates penalties (Proposition 6) and is eventually excluded. The attacker must register new identities (incurring minimum deposit costs and the 32,000-block withdrawal delay per Section 10.5) to sustain proposer disruption. Case (b): with honest deposit > 2/3 and message delivery within Δ after GST, honest validators' OK votes arrive before timeout, so Round 1 succeeds whenever the proposer is honest and reachable. If Round 1 fails despite an honest proposer (due to network jitter within Δ of the timeout), Round 2 guarantees finalization (Theorem 3). In the worst case, the attacker can cause at most a fraction f of blocks to finalize as NIL by having Byzantine proposers abstain, but this fraction is bounded and decreasing as attacker proposers are excluded.

Deferred analyses and their interactions. This paper provides formal propositions for the normalization amplification bound (Proposition 2, Section 12), committee representativeness in expectation (Proposition 3, Section 10.2), per-block committee Byzantine fraction tail bound (Proposition 4, Section 10.2), incentive compatibility (Proposition 5, Section 11.7), offline penalty trajectory (Proposition 6, Section 11.7), and sandwich attack probability (Proposition 7, above), as well as informal arguments for Sybil proposer advantage (Remark 1, Section 10.3) and NIL-block persistence (Remark 2, above). Several aspects remain deferred to future publications: deposit-sizing equilibria under normalization and a comprehensive threat model covering economic attacks, Eclipse attacks, and targeted DoS. These deferred items are not independent: a Sybil attacker who can influence committee composition to gain disproportionate proposer probability, combined with the ability to target predicted proposers with DoS, has a compounding advantage that no single analysis captures. A comprehensive threat model covering economic attacks (bribery, inclusion-level MEV), network-level adversaries (Eclipse attacks, targeted DoS against predicted proposers), and detailed performance analysis is planned for a separate security analysis document. The protocol's round-robin execution ordering and hash-only proposals provide structural mitigations for ordering-level MEV (see above), but inclusion-level MEV — where the proposer selectively includes or excludes transactions — remains an open problem.

---

14. Summary and Novel Contributions

The QuantumCoin consensus protocol is a deposit-weighted BFT protocol providing immediate deterministic finality for every block. Its key properties are:

- **Explicit system model** (Section 6.1) with five numbered assumptions (partial synchrony, authenticated channels, Byzantine bound, deposit-weighted voting, single proposer per round) that are referenced by all formal results.
- **Multi-phase consensus** — four phases for the proposer (Proposal, Acknowledgment, Precommit, Commit) and three phases for non-proposer validators (Acknowledgment, Precommit, Commit) — with a two-thirds deposit-weighted quorum threshold, derived from the standard BFT quorum intersection requirement.
- **Byzantine fault tolerance** for strictly less than one-third of the per-block committee's effective deposit, encompassing equivocation, selective delivery, arbitrary voting, phase skipping, and crash faults.
- **Safety independent of timing** (Theorems 1–2, Section 8): no two honest validators finalize conflicting blocks, regardless of network conditions or message delivery timing (given the < 1/3 fault tolerance bound).

- **Liveness under partial synchrony** (Theorem 3, Section 8): all honest validators eventually finalize, given bounded message delay after an unknown stabilization time.
- **Two-round design** with a forced empty-block fallback that guarantees progress by construction under the partial synchrony assumption, avoiding unbounded retry loops.
- **Lookback-based randomness** (512,000 blocks) for validator and proposer selection, resistant to proposer manipulation.
- **Graduated offline penalties** — deposit deduction, progressive voting weight reduction, proposer deferral, and validator exclusion — attributed only when individual proposer failure is distinguishable from network-wide issues (Round 1 vs. Round 2).
- **Unbounded validator registration** with a bounded per-block committee (up to 128). Any number of validators may register by locking a deposit; the 128 limit is a per-block communication-complexity bound, not a network membership cap. Committee membership rotates at every block height from the eligible registered set (after applying offline penalty exclusions per Section 11.5).
- **Deposit normalization** clipping any disproportionately large validator's effective deposit to 10% of the pre-normalization total, with single-pass redistribution of the clipped excess voting weight to validators with clean participation records (no actual coins are transferred).
- **Withdrawal delay and one-time address binding** (Section 10.5) rate-limiting and capital-limiting validator-count grinding attacks on the consensus randomness.
- **Ordering-level MEV mitigation** (Section 13) via deterministic round-robin transaction execution ordering (by Keccak256(parentHash, senderAddress)) and hash-only block proposals, which disrupt sandwich attacks and raise the cost of last-moment transaction injection.
- **Formal proofs and verification**: pen-and-paper safety proof (Theorem 1), asynchrony safety proof (Theorem 2), and bounded liveness proof (Theorem 3) in Section 8; complementary TLA+ model checking [7][8] confirming properties exhaustively over bounded configurations, with the fault tolerance boundary demonstrated at 33% (pass) and 34% (violation).

The protocol's per-block message complexity is O(n^2) where n is the committee size (up to 128), characteristic of PBFT-family protocols.

Novel contributions. The protocol's contribution combines a well-understood BFT foundation with five mechanisms that, to the authors' knowledge, are not present in existing deployed proof-of-stake protocols:

- **Committed-history randomness** (Section 10.1) that eliminates both last-revealer bias and grindable-abort vulnerabilities by deriving consensus randomness entirely from on-chain data committed 512,000 blocks in the past.
- **Protocol-enforced round-robin transaction execution ordering** (Section 13) that algorithmically separates transaction selection from execution order, mitigating sandwich attacks without relying on proposer-builder separation.
- **Three-pass bounded-committee selection algorithm** (Section 10.2) that compositionally combines deposit-ranked, coin-flip, and pseudorandom selection to guarantee both high deposit representation and small-validator inclusion.
- **Deposit normalization** (Section 12) that clips any validator's effective deposit exceeding 10% of the eligible set's total deposit and redistributes the clipped excess voting weight exclusively to validators with a clean participation record — no actual coins

are transferred; this is a per-block adjustment to quorum calculation weights only. In deposit-weighted protocols (Cosmos, Algorand, Solana), no deployed mechanism dynamically caps a single validator's quorum influence as a percentage of total deposit and redistributes the excess based on participation record.

- **Graduated offline penalty system** (Section 11) that combines fixed deposit deductions, progressive voting weight reduction (2% per attributed NIL block), exponential proposer deferral, and permanent exclusion at a defined threshold (NilBlockCount = 50) — with penalties attributed only when individual proposer failure is distinguishable from network-wide issues (Round 1 vs. Round 2). Existing protocols use either flat penalties with binary jailing (Cosmos), reward-loss-only models (Ethereum under normal conditions), or no offline penalties at all (Solana, Algorand). QuantumCoin's multi-dimensional graduated trajectory provides a continuous incentive gradient rather than a binary threshold.

These mechanisms are integrated alongside production post-quantum cryptography.

Throughput benchmarks under representative network conditions are planned for a future publication.

---

15. References

1. M. Castro and B. Liskov, "Practical Byzantine Fault Tolerance," Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI), 1999. https://pmg.csail.mit.edu/papers/osdi99.pdf
2. RANDAO. https://github.com/randao/randao
3. C. Liang, "Minimum Committee Size Explained" (supplementary exposition). https://medium.com/@chihchengliang/minimum-committee-size-explained-67047111fa20
4. M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of Distributed Consensus with One Faulty Process," Journal of the ACM, vol. 32, no. 2, pp. 374–382, April 1985. https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf
5. C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," Journal of the ACM, vol. 35, no. 2, pp. 288–323, April 1988. https://groups.csail.mit.edu/tds/papers/Lynch/jacm88.pdf
6. QuantumCoin Consensus Protocol Specification (step-by-step protocol description, glossary, fault model, and design rationale). https://github.com/quantumcoinproject/quantum-coin-go/blob/main/consensus/proofofstake/README.md
7. QuantumCoin Consensus TLA+ Specification (formal TLA+ model, Byzantine behaviors modeled, properties verified, and model configurations). https://github.com/quantumcoinproject/quantum-coin-go/blob/main/consensus/proofofstake/tla/README.m
8. QuantumCoin TLA+ Verification Report (TLC model checking results, state-space metrics, fault tolerance boundary analysis, and counterexample narratives). https://github.com/quantumcoinproject/quantum-coin-go/blob/main/consensus/proofofstake/tla/tla-report.m
9. Á. Nagy, J. Tapolcai, I. A. Seres, and B. Ladóczki, "Forking the RANDAO: Manipulating Ethereum's Distributed Randomness Beacon," Cryptology ePrint Archive, Report 2025/037, 2025. https://eprint.iacr.org/2025/037
10. QuantumCoin Post-Quantum Cryptography Documentation (hybrid signature scheme

specification, algorithm selection rationale, and key management). https://quantumcoin.org/whitepapers/Quantum-Coin-Blockchain-Quantum-Resistance-Whitepaper.pdf

11. G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche, "The Keccak Reference," Version 3.0, January 2011. https://keccak.team/files/Keccak-reference-3.0.pdf
12. National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Federal Information Processing Standards Publication 202, August 2015. https://doi.org/10.6028/NIST.FIPS.202
13. V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining GHOST and Casper," arXiv preprint arXiv:2003.03052, 2020. https://arxiv.org/abs/2003.03052
14. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine Agreements for Cryptocurrencies," Proceedings of the 26th Symposium on Operating Systems Principles (SOSP), 2017. https://dl.acm.org/doi/10.1145/3132747.3132757
15. M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus with Linearity and Responsiveness," Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC), 2019. https://dl.acm.org/doi/10.1145/3293611.3331591
16. Flashbots, "MEV-Boost: Proposer-Builder Separation for Ethereum," 2022. https://boost.flashbots.net/
17. Ethereum Foundation, "Ethereum Consensus Specifications — Phase 0: The Beacon Chain" (committee structure, validator duties, protocol constants). https://ethereum.github.io/consensus-specs/specs/phase0/beacon-chain/
18. CometBFT Contributors, "Proposer Selection Procedure," CometBFT Documentation, v0.38 (weighted round-robin algorithm, validator set update semantics). https://docs.cometbft.com/v0.38/spec/consensus/proposer-selection
19. E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," arXiv preprint arXiv:1807.04938, 2018 (Tendermint consensus algorithm with proofs of safety and termination). https://arxiv.org/abs/1807.04938

---

16. Appendix A: Consensus Steps

The following 14 steps describe the complete per-block consensus flow. All vote thresholds refer to 67% of the filtered validator set's total deposit (see Section 6.2). Round escalation (step 14) applies only from Round 1; at most 2 rounds are permitted per block.

Step 1 — Interrupt. At any point, if a valid finalized block is received from the network for the current block number, abort the current consensus and go to step 2 for the next block. A block is considered validly finalized if it is accompanied by a commit certificate: a set of commit votes for a single commit hash whose signers' combined deposit weight meets or exceeds the two-thirds threshold of the filtered validator set's total deposit.

Step 2 — New block begins. The consensus handler is invoked with the parent hash of the last finalized block.

Step 3 — Select validator set. Deterministically select and filter the set of validators for this block based on on-chain deposits (Section 10.2), apply deposit normalization (Section 12) if the filtered set contains at least 12 validators, and compute quorum thresholds (Section 6.2) against post-normalization deposits.

Step 4 — Initialize Round 1. Set round = 1 and state = WAITING_FOR_PROPOSAL.

Step 5 — Select proposer. Deterministically compute the proposer for the current round from the filtered validator set (Section 10.3), applying proposer deferral rules (Section 11.4) to exclude validators with recent nil-block attributions.

Step 6 — Check proposer role. Is this node the selected proposer?

6.1) Yes (proposer path):

6.1.1) Construct a PROPOSAL message referencing selected pending transactions by their hashes (or zero transactions if round = 2).

6.1.2) Broadcast the PROPOSAL to all validators including self.

6.1.3) Go to step 7.

6.2) No (non-proposer path):

6.2.1) Go to step 7.

Step 7 — Evaluate proposal receipt. Was a valid PROPOSAL received before the proposal timeout? A proposal is valid if it is correctly signed, matches the expected round and parent hash, and every referenced transaction hash can be resolved against the validator's local mempool. An honest validator accepts the first valid proposal it receives for a given round and ignores any subsequent proposals for the same round.

7.1) Yes, proposal received (first valid proposal for this round):

If round = 1: broadcast an ACK_PROPOSAL with vote type = OK and the proposal's hash to all validators.

If round = 2: broadcast an ACK_PROPOSAL with vote type = NIL and the deterministic nil-proposal hash to all validators.

7.2) No, proposal timed out:

Broadcast an ACK_PROPOSAL with vote type = NIL to all validators.

Step 8 — Wait for acknowledgment threshold. Collect ACK_PROPOSAL votes from validators until the 67% weighted deposit threshold is reached for a single (vote type, proposal hash) pair. Votes for different proposal hashes do not combine, even if they share the same vote type.

Step 9 — Evaluate acknowledgment threshold. Has the 67% threshold been reached?

9.1) Yes:

Broadcast a PRECOMMIT vote to all validators. Go to step 10.

9.2) No:

If round = 1: if the acknowledgment timeout is exceeded, or if validators already participating in Round 2 hold enough deposit that Round 1 can never reach the 67% threshold, go to step 14.

If round = 2: remain waiting (no further rounds exist).

Step 10 — Wait for precommit threshold. Collect PRECOMMIT votes from validators until the 67% weighted deposit threshold is reached.

Step 11 — Evaluate precommit threshold. Has the 67% threshold been reached?

11.1) Yes:

Broadcast a COMMIT vote to all validators. Go to step 12.

11.2) No:

If round = 1: if the precommit timeout is exceeded **and** validators already participating in Round 2 hold enough deposit that Round 1 can never reach the 67% threshold, go to step 14. (Note: this uses AND, unlike Step 9.2 which uses OR. The asymmetry is intentional; see Section 5.3 for the rationale and for a proof that this AND condition cannot produce a deadlock — the state in which all honest validators are in Round 1 precommit with no Round 2 participants is unreachable under Assumption A3.)

If round = 2: remain waiting (no further rounds exist).

Step 12 — Wait for commit threshold. Collect COMMIT votes from validators until the 67% weighted deposit threshold is reached.

Step 13 — Evaluate commit threshold. Has the 67% threshold been reached?

13.1) Yes:

Block is finalized. Apply block rewards (Section 11.7) if the finalized vote type is OK. If the finalized vote type is NIL and round = 1, apply proposer penalties (Sections 11.2-11.6). Produce the block and go to step 2 for the next block.

13.2) No:

Remain waiting, unless step 1 is triggered.

Step 14 — Round escalation (only from Round 1; at most 2 rounds per block; see Section 5.3 for the no-dual-round-participation invariant and transition triggers):

14.1) Set round = 2 and reinitialize local consensus state. Round 2 is a forced empty-block round (Section 5.4): the proposer MUST include zero transactions in the PROPOSAL, and all validators MUST vote NIL in their ACK_PROPOSAL. This produces an empty block, guaranteeing the chain makes progress. Because the block finalizes at Round 2, no penalties are applied to the Round 1 proposer (Section 11.1: penalties apply only to blocks finalized at Round 1).

14.2) Go to step 5 with round = 2.