The ARIA Chain Protocol

Index

1. Introduction p.1
2. Mathematical Model p.2
2.1 State Representation p.2
2.2 Block Representation p.2
2.3 Transaction Structure p.2
2.4 Deterministic Execution p.3
3. Tokenomics and Economic Model p.3
3.1 Monetary Policy p.3
3.2 Transaction Fees p.3
3.3 Staking Rewards p.3
3.4 Slashing and Penalties p.3
3.5 Treasury Mechanism p.3
4. Consensus Protocol p.4
4.1 Validator Selection p.4
4.2 Block Production p.4
4.3 Finality Layer p.4
4.4 Byzantine Fault Tolerance p.4
4.5 Synchronization and Recovery p.4
4.6 Determinism and Fairness p.4
5. Governance Framework p.4
5.1 Proposal Creation p.5
5.2 Voting and Quorum Rules p.5
5.3 Execution of Approved Proposals p.5
5.4 Treasury Governance p.5
5.5 Upgrade and Version Management p.5
5.6 Transparency and Accountability p.5
6. State Model and Accounts p.5
6.1 Global State p.5
6.2 Account Structure p.6
6.3 State Transition Logic p.6
6.4 Deterministic Properties p.6
6.5 State Integrity and Verification p.6
7. Transaction Lifecycle p.6
7.1 Transaction Creation p.6
7.2 Propagation and Mempool p.6
7.3 Block Inclusion p.7
7.4 Transaction Validation p.7
7.5 Fee Deduction and Distribution p.7
7.6 Confirmation and Finality p.7
7.7 Determinism and Replay Protection p.7
8. Cryptography and Security p.7
8.1 Key Generation and Identity p.7
8.2 Digital Signatures p.8
8.3 Hashing and State Verification p.8
8.4 Merkle Proofs p.8
8.5 Validator Authentication p.8
8.6 Slashing and Misbehavior Detection p.8
8.7 Network Encryption p.8
8.8 Security Guarantees p.8
9. Network and Communication p.8
9.1 Peer Discovery p.8
9.2 Message Propagation p.9
9.3 Block Propagation p.9
9.4 Transaction Distribution p.9
9.5 Synchronization p.9
9.6 Network Reliability p.9
9.7 Security and Privacy p.9
9.8 Finality Synchronization p.9
10. Runtime Implementation p.9
10.1 WebAssembly Execution p.9
10.2 Modular Design p.10
10.3 Deterministic Logic p.10
10.4 Genesis Configuration p.10
10.5 Runtime Upgrades p.10
10.6 Constants and Parameters p.10
10.7 Testing and Verification p.10
11. Appendix p.10
12. System Parameters p.10
12.1 Governance Example p.11
12.2 Example of Slashing Event p.11
12.3 Example of State Update p.11
12.4 Testnet and Simulation p.11
12.5 Notation Reference p.11

1. Introduction

Aria Chain is a deterministic blockchain that follows the Nominated Proof-of-Stake (nPoS) model.
Its design focuses on three principles: predictable execution, verifiable state changes, and mathematically defined economics.
The chain operates as a state machine. The current state depends only on the previous state and the new block added to the network. Each node runs the same deterministic logic inside the WebAssembly runtime to produce the same result.

$$\sigma_k = \Gamma\,(\sigma_{k-1}, B_k)$$

This ensures that no matter who runs the node, the final outcome of block execution will always match.
Aria's rules and parameters are defined inside the runtime. Nothing critical depends on external configuration.
The token supply is limited to 200,000,000 ARIA.
Each transaction has a fixed cost of 5 ARIA.

Validator rewards follows:

20% to validators
80% to the treasury

If a validator acts dishonestly, a fixed penalty of 50,000 ARIA is applied, which equals about 5% of the required staking deposit.
The network runs on the Adaptive Consensus Protocol (ACP). This protocol combines Proof-of-Stake efficiency with Byzantine fault tolerance. Validators are chosen through a nomination system. Blocks are produced by time slots, and finality is achieved by confirmation rounds between validators.
The protocol remains secure as long as less than one-third of validators behave maliciously.
Governance decisions follow a minimal quorum rule:
at least three validators must vote, and at least two must approve for a proposal to pass. This rule ensures fairness and fault tolerance even if some validators are offline.
In summary, Aria Chain is a mathematically defined blockchain. Every transaction, reward, and governance action follows deterministic logic. There is only one valid result for each block — and that result can always be verified by anyone.

2. Mathematical Model

Aria Chain defines its operation as a deterministic state machine.
Every change in the system is a function of two inputs: the previous state and the new block received by the network.

We can describe this relationship as:

state_new = apply_transition(state_prev, block_new)

Here:

state_prev represents the entire global state before a block is applied.
block_new contains all transactions, validator data, and consensus metadata.
state_new is the verified result after executing the block.
apply_transition() is the deterministic function that updates balances, staking data, governance records, and all other state values.

Each node in the network independently runs the same apply_transition() logic within the WebAssembly runtime.
If both the input state and the block data are valid, all nodes reach the same result.
This property guarantees determinism and eliminates any possibility of divergent chains caused by local variations.

2.1 State Representation

The global state contains all accounts, staking pools, governance proposals, and system constants.

We can express it conceptually as:

```
state = {
  accounts: { address: balance, ... },
  staking: { validator_id: { stake, reward, slash }, ... },
  governance: { proposals, votes },
  treasury: { balance },
  runtime_constants: { fee, apr, slash_penalty }
}
```

Each account and validator is stored with a verifiable identifier.
Merkle-style hashing is used to ensure that any change can be proven without revealing the full state.

$$h_{root} = H\,(h_1 \,||\, h_2 \,||\, \ldots \,||\, h_n)$$

Nodes can verify inclusion with compact proofs, allowing light clients to confirm balances or validator data efficiently.

2.2 Block Representation

Each block is a compact container that defines what changes the next state should reflect.
It includes:

```
block = {
  header: { parent_hash, number, state_root, timestamp, validator_signature },
  transactions: [tx1, tx2, tx3, ...],
  metadata: { validator_set, slot, finality_votes }
}
```

The header connects the block to its parent through the parent_hash, ensuring continuity of the chain.
The state_root field is a hash representing the entire state after the block's execution.

$$state_{root} = H\,(state_{new})$$

This allows any node to verify that the transition was performed correctly.

2.3 Transaction Structure

A transaction is the smallest executable unit of state change.

It is defined as:

```
transaction = {
  from: address,
  to: address,
  amount: number,
  fee: 5,
  nonce: number,
  signature: bytes
}
```

Each transaction must be signed by the sender, validated against account balances, and processed in order.
The runtime ensures that the same transaction cannot be applied twice by checking the nonce value.

2.4 Deterministic Execution

When a node processes a block, it executes each transaction sequentially.
No randomness, time-based logic, or external data can affect the result.
This guarantees that all honest nodes produce identical state_new outputs.

If a validator or node produces a different result, its block will be rejected during verification because the computed state_root will not match the expected value.

$$verify\ (B_h) = true \Leftrightarrow H\ (apply_transition(\sigma_{h-1}, B_h)) = state_{root_expected}$$

3. Tokenomics and Economic Model

Aria Chain operates with a fixed total supply of 200,000,000 ARIA. The supply is non-inflationary and fully pre-allocated across several categories to ensure long-term stability and decentralization.

The distribution is as follows:

Coco Blockchain Lab receives 100,000,000 ARIA, which represents 50% of the total supply.
This portion secures validator control and maintains network security.
Early Investors (Launchpad) receive 30,000,000 ARIA, or 15% of the total supply.
These tokens are used for initial distribution and funding during the early network phase.
Founders receive 50,000,000 ARIA, equal to 25%.
This ensures long-term leadership and alignment with the project's goals.
Listing and Liquidity Pool holds 10,000,000 ARIA, equal to 5%.
These tokens are used to provide liquidity for exchange listings and market stability.
The development team receives 5,000,000 ARIA, also 2.5%.
This allocation is reserved for development incentives and research contributions.
Marketing receives 5,000,000 ARIA, another 2.5%.
This portion funds global awareness, partnerships, and user adoption efforts.

The allocation model is designed to maintain network stability and align incentives for sustainable growth.

3.1 Monetary Policy

Aria has a permanently capped supply of 200 million ARIA.

No new tokens can ever be minted. Inflation is zero by design.

All staking rewards and validator incentives are paid from existing allocations, ensuring long-term predictability and transparent monetary control.
At genesis, 200 million ARIA are released as the initial circulating supply.

3.2 Transaction Fees

Each transaction on Aria Chain requires a fixed fee of 5 ARIA.

This fee is distributed in two parts: 20 percent is given to active validators as compensation for network maintenance, and 80 percent is sent to the treasury wallet to fund ecosystem development.

$$F_{tx} = 5\ ARIA$$
$$F_{validator} = 0.2 \times F_{tx}$$
$$F_{treasury} = 0.8 \times F_{tx}$$

The fixed fee structure simplifies user experience and ensures predictable network costs.

3.3 Staking Rewards

Validators and nominators receive staking rewards at a fixed annual rate of 4 percent.
Rewards are distributed per epoch and are proportional to the amount of tokens staked by each participant.
This system encourages consistent validator participation and long-term staking commitment.

3.4 Slashing and Penalties

To maintain security, Aria applies a strict slashing rule.
If a validator is found to act maliciously or violate consensus rules, a penalty of 50,000 ARIA is applied.
This value equals approximately five percent of the minimum validator deposit.

$$S_{penalty} = 50{,}000\ ARIA = 0.05 \times Deposit_{min}$$

The slashed tokens are permanently transferred to the treasury wallet to support collective security and discourage misconduct.

3.5 Treasury Mechanism

The treasury wallet accumulates funds from three sources: transaction fees, slashing penalties, and unclaimed staking rewards.

Treasury funds are released only through on-chain governance decisions approved by validator votes.

This ensures transparency, predictable economics, and verifiable funding for future development.

4. Consensus Protocol

Aria Chain uses the Adaptive Consensus Protocol, abbreviated as ACP. This protocol combines the efficiency of Proof of Stake with the fault tolerance of Byzantine systems. It ensures that every block produced on the network is both valid and final under a deterministic set of rules.

$$\sigma_h = \Gamma\,(\sigma_{h-1}, B_h)$$

The consensus process is built around nominated validators. Token holders can nominate trusted validators who are responsible for block production and finalization. The election process selects a limited number of validators based on total stake and reputation metrics.
Each validator must lock a minimum staking amount to be eligible for participation. Nominators delegate their tokens to validators to strengthen their election chances. Both validators and nominators receive periodic rewards as long as they remain active and behave honestly.

4.1 Validator Selection

Validator selection occurs at the beginning of each era. The system evaluates all staking positions and chooses a subset of validators who meet the minimum criteria for stake amount and reliability.
Eras are predefined time intervals that include multiple epochs. At the end of each era, rewards are calculated and distributed.
The nomination system ensures decentralization by allowing token holders to influence which nodes are trusted to secure the chain. This mechanism prevents power concentration and ensures fair validator rotation.

4.2 Block Production

Block production in Aria Chain follows a slot-based mechanism similar to probabilistic leader election. Time is divided into discrete slots, and one validator is randomly assigned to propose a block during each slot.

$$P\,(V_i \rightarrow S_k) = \frac{w_i}{\Sigma_j w_j}$$

The random selection is derived from a verifiable randomness function seeded from previous blocks.
Each proposed block includes transactions, staking updates, and governance actions. Once produced, the block is broadcast to all peers for verification.
Validators check that the block is valid, signatures are correct, and the transactions respect all runtime rules. If the block passes validation, it becomes part of the canonical chain.

4.3 Finality Layer

While block production is probabilistic, finality is deterministic. The ACP protocol includes a finality layer inspired by GRANDPA-style voting. Validators collectively sign and confirm completed chains rather than individual blocks.
Once two-thirds of active validators confirm a chain head, that block and all its ancestors become final and irreversible.

$$\Sigma\,(votes_{validators}) \geq \frac{2}{3} \times N$$

This hybrid approach allows high throughput and fast block generation while ensuring absolute finality after confirmation.
Even if temporary forks occur due to network delay, the finality layer guarantees that only one canonical chain can exist after confirmation.

4.4 Byzantine Fault Tolerance

ACP is designed to tolerate up to one-third of malicious validators without losing safety or liveness.
If fewer than one-third of validators act dishonestly, the protocol continues to operate correctly, and no conflicting finalized chains can appear.
In case of malicious behavior such as double signing or invalid block production, the slashing mechanism applies automatically, deducting 50,000 ARIA from the offending validator.

4.5 Synchronization and Recovery

Nodes joining the network can synchronize by downloading the latest finalized block and verifying all state proofs.
Since each block contains a verifiable state root, light clients and new nodes can independently confirm network correctness without processing the full history.
In case of temporary partition or validator downtime, the ACP protocol ensures recovery once quorum is re-established.

4.6 Determinism and Fairness

The Adaptive Consensus Protocol ensures determinism through strict input equality.
Given the same validator set, slot schedule, and transaction pool, all honest nodes always compute the same block outcome.
Fairness is maintained by rotating validator assignments and randomizing slot selection, ensuring that no single validator can control block production frequency.
In summary, the Adaptive Consensus Protocol gives Aria Chain three key properties: deterministic execution, provable finality, and resilience against Byzantine failures. It achieves these properties without relying on external randomness or probabilistic settlement, making Aria suitable for both financial and governance-critical applications.

5. Governance Framework

Aria Chain uses an on-chain governance system that defines how proposals are created, discussed, and approved. The entire process takes place inside the blockchain runtime, making every governance decision transparent and verifiable by any participant.
The goal of the governance framework is to keep the network stable while allowing controlled evolution. All protocol upgrades, parameter adjustments, and treasury expenditures must pass through governance voting before they take effect.

5.1 Proposal Creation

Any validator can create a governance proposal.
A proposal can include actions such as upgrading the runtime, adjusting staking parameters, funding a treasury project, or changing economic constants.
When a proposal is submitted, it is recorded on-chain and enters the proposal queue. Each proposal includes the validator's identifier, a description of the action, and a unique reference ID.
To prevent spam and ensure commitment, submitting a proposal requires a small deposit. The deposit is returned if the proposal is accepted or rejected through the official process. It is forfeited if the proposer fails to follow through with the required validation steps.

5.2 Voting and Quorum Rules

Voting is restricted to active validators. Each validator has one equal vote, regardless of stake size.
For a proposal to pass, at least three validators must participate in the voting process. Among them, a minimum of two must vote in favor of the proposal.
This rule can be expressed in plain terms:
"Three validators must vote, and at least two must approve."
This represents a minimum 50 percent plus one approval requirement. The quorum rule ensures that no single validator can pass a change and that every decision has collective consent.
Voting occurs over a defined period measured in epochs. After the voting window closes, the proposal status becomes either "approved," "rejected," or "expired."
Only approved proposals move to the execution phase.

5.3 Execution of Approved Proposals

Once a proposal is approved, the runtime automatically schedules it for execution.
Depending on the proposal type, execution may involve updating runtime parameters, distributing treasury funds, or deploying a new runtime version.
The deterministic runtime guarantees that execution happens exactly as defined in the proposal, without manual intervention.
All governance actions are logged permanently in the chain history.

This provides full accountability and makes it possible to verify every protocol change over time.

5.4 Treasury Governance

The treasury wallet is controlled by the governance system.
Funds in the treasury can only be spent through approved proposals.

Common treasury actions include funding validator incentives, development grants, and community growth programs.

Each spending request must specify the recipient address, amount, and purpose.

The treasury module automatically enforces that no funds can leave the wallet without a valid, approved governance record.

5.5 Upgrade and Version Management

Aria supports seamless runtime upgrades through governance decisions.
When an upgrade proposal is approved, validators collectively deploy the new runtime version during the next epoch transition.
The runtime version number is increased, and all nodes verify the integrity of the new code before accepting blocks.
This mechanism eliminates the need for hard forks, allowing the network to evolve safely without chain splits.

5.6 Transparency and Accountability

All governance data is public. Anyone can view proposals, votes, and execution results directly from the chain explorer.
Validator identities, voting behavior, and final decisions are permanently stored on-chain, making the entire governance process open to review and audit.
In summary, Aria's governance system ensures that all network changes are transparent, rule-based, and collectively verified.
It provides a self-contained model where decision-making, treasury management, and runtime upgrades are fully decentralized yet mathematically predictable.

6. State Model and Accounts

Aria Chain defines its entire system as a global state that evolves over time.
The state contains every account, validator, staking record, governance proposal, and treasury balance.
Each new block updates the state deterministically, meaning that all nodes processing the same block always reach the same result.

6.1 Global State

The global state is a structured set of records stored in the blockchain database.
It represents the complete status of the network at a given block height.
Every block execution replaces the previous state with a new version that reflects all valid transactions and system updates.
Conceptually, the state can be described as containing the following components:

$$\sigma = \{(a,(b,n,s,r,p,t)) \mid a \in A\}$$

A list of user accounts with balances and nonces
A list of validators and their staking information

The treasury balance and its transaction history
Governance proposals and voting records
Runtime constants such as fees, reward rates, and slashing parameters
Every value is verifiable and cryptographically hashed to ensure that the full state can be confirmed by anyone, including light clients that do not store complete history.

6.2 Account Structure

Each account is identified by a unique address.
An account contains several key fields that define its role and participation in the system.

The main fields are:

address: a unique identifier derived from the public key
balance: the total amount of ARIA tokens owned by the account
nonce: a sequential number that prevents replay of transactions
stake: the amount locked for validator nomination or self-bonding
role: the account type, which can be user, validator, or treasury
reward: accumulated rewards pending withdrawal

User accounts are free to transfer tokens, participate in governance, or nominate validators.
Validator accounts have additional privileges and responsibilities, including block production and vote signing.
The treasury account is a special on-chain entity that holds network funds and operates only through approved governance actions.

6.3 State Transition Logic

Whenever a new block is applied, the state transition logic executes the following sequence of steps:

Verify that the block is signed by an active validator.
Validate all transactions in the block according to nonce and balance rules.
Deduct transaction fees from each sender and transfer them to the fee distribution module.
Update balances for all transfers.
Apply staking and reward updates for validators.
Process governance proposals that have reached their voting deadline.
Update treasury records and allocate funds when necessary.
Recalculate the final state and store its root hash in the block header.

This deterministic sequence ensures that the outcome of every block is fully reproducible and independent of local node configuration.

6.4 Deterministic Properties

The Aria state model is strictly deterministic.
No random, time-dependent, or external input is allowed to affect execution.
Given the same previous state and the same block, all honest nodes will produce the exact same next state.
This principle guarantees that Aria's blockchain remains consistent and verifiable across all nodes globally.
If a node produces a different result, its block will be automatically rejected by peers because its computed state root will not match the expected value recorded in the block header.

6.5 State Integrity and Verification

The state database uses hash-based verification.
Each part of the state (accounts, staking, governance, treasury) is hashed and combined into a single root value.

$$R = H\,(H_1 \parallel H_2 \parallel \ldots \parallel H_n)$$

This allows the network to confirm correctness without needing to transmit the entire dataset.
In summary, the Aria state model defines every component of the network as verifiable and deterministic data.
All balances, votes, and validator records exist within the same unified state, and every block updates that state according to strict and reproducible logic.

7. Transaction Lifecycle

Every transaction in Aria Chain follows a fully deterministic process from submission to final state inclusion.
The design ensures that all nodes, validators, and users see the same result for every valid transaction.
No external factors, timing differences, or random data can change the outcome once a transaction is accepted.

7.1 Transaction Creation

A transaction is created by a user who wants to transfer tokens, stake, vote, or perform another runtime action.

Each transaction includes the following basic fields:

from: the sender's account address
to: the receiver's address
amount: the number of tokens transferred
fee: the fixed transaction cost of ARIA
nonce: a unique sequential number from the sender's account
signature: the sender's digital signature verifying ownership

The wallet or client software signs the transaction using the sender's private key.
Once signed, the transaction is immutable and can be broadcast to the network.

7.2 Propagation and Mempool

When a node receives a transaction, it first validates its format and digital signature.
If valid, it is temporarily stored in the node's transaction pool, also called the mempool.

Each validator continuously collects pending transactions from its local mempool to include in the next block.
Transactions in the mempool are sorted by arrival time.
Because Aria uses a fixed fee, there is no fee-based priority competition.
This simplifies scheduling and ensures predictable inclusion for all users.
Invalid or duplicate transactions are immediately discarded.
A transaction is considered invalid if the sender's balance is insufficient or the nonce does not match the expected sequence.

7.3 Block Inclusion

During block production, the validator responsible for the current slot selects transactions from its mempool and executes them one by one inside the runtime.
Each transaction updates the system state by modifying account balances, stakes, or governance data.
After all transactions are executed, the validator calculates the new state root and signs the block.
The block is then broadcast to all other validators and nodes for verification.
If all checks pass, the block becomes part of the canonical chain.
The transactions inside it are now permanently recorded and cannot be reversed.

7.4 Transaction Validation

Before accepting a transaction, every node performs the following checks:

Verify that the transaction is properly signed.
Verify that the sender has a sufficient balance to cover both the transfer and the fee.
Verify that the nonce matches the sender's last known transaction count.
Verify that the transaction has not been included in a previous block.
Verify that the transaction does not violate any runtime rules such as staking limits or treasury restrictions.

Only transactions that pass all validation rules are accepted and applied.

7.5 Fee Deduction and Distribution

For each transaction, exactly 5 ARIA are deducted as the network fee.

This fee is split into two parts:

20 percent goes to the validator who produced the block
80 percent goes to the treasury account

The fee deduction happens automatically during block execution.
The validator does not manually collect or process fees, ensuring that all rewards and treasury inflows remain transparent and verifiable.

7.6 Confirmation and Finality

Once a transaction is included in a block, it becomes visible on the network.
However, it is considered fully final only after the block is confirmed by the finality mechanism of the Adaptive Consensus Protocol.
After finality is reached, the transaction can never be reverted or altered.
Wallets and explorers can safely report the transaction as confirmed after this point.

7.7 Determinism and Replay Protection

Each transaction includes a unique nonce that prevents replay attacks.

$$Valid\ (T) \Leftrightarrow Tnonce = Account\ (T.from).nonce + 1$$

If a transaction is submitted twice, only the first one with the correct nonce is accepted.
Subsequent duplicates are automatically rejected by all nodes.
The fixed-fee and deterministic execution ensure that transaction processing is predictable and cannot be influenced by network congestion or validator preference.
In summary, every transaction in Aria Chain follows a strict and transparent path:
it is signed, verified, included in a block, executed deterministically, and finalized through consensus.
Once finalized, it becomes an immutable part of the blockchain history.

8. Cryptography and Security

Aria Chain relies on modern cryptographic primitives to guarantee authenticity, integrity, and non-repudiation across the network.
All actions in the system, from transaction signatures to validator consensus, are backed by verifiable cryptographic proofs.

8.1 Key Generation and Identity

Every participant in Aria Chain owns a key pair consisting of a private key and a public key.
The private key is used to sign messages, while the public key is used to verify those signatures.
Each account address is derived from the public key through a hashing process to make it compact and tamper-resistant.

Validators are identified by their public keys and must register them on-chain before participating in consensus.
Key rotation is supported, allowing validators to replace their signing keys through governance or staking updates without losing their reputation or history.
Users must keep their private keys secure.
Loss or exposure of a private key gives full control of the corresponding account to whoever possesses it.

8.2 Digital Signatures

Every transaction and block in Aria Chain is signed using a digital signature algorithm.

$$\sigma = Sign\ (PrivKey, m)$$
$$Verify\ (PubKey, m, \sigma) = true$$

The protocol supports both Ed25519 and Sr25519 signature schemes for flexibility and strong cryptographic guarantees.
Signatures provide three security properties: authenticity, integrity, and non-repudiation.
Authenticity ensures that the sender truly authorized the message.
Integrity ensures that the message content has not been modified.
Non-repudiation guarantees that the sender cannot later deny creating the message.
Nodes automatically verify signatures before accepting any transaction or block.
If a signature does not match the public key, the message is rejected immediately.

8.3 Hashing and State Verification

Aria uses cryptographic hash functions to link and verify all blocks and state data.
Each block contains a hash of its parent block, forming an unbroken chain of trust.
The state of the blockchain is also represented by a root hash that summarizes all accounts, staking records, and governance data.
When a new block is produced, the validator calculates the new state hash based on all updates in that block.
Other nodes verify this hash independently.
If any node produces a different hash, its block will be rejected because the state transition is not identical to the expected result.

8.4 Merkle Proofs

To make verification efficient, Aria uses a Merkle proof system.
Merkle proofs allow any node or light client to verify that a specific piece of data, such as an account balance or validator record, exists in the current state without downloading the entire database.

Each piece of state data is hashed and combined into a tree structure.
By providing a small sequence of intermediate hashes, a node can prove inclusion of a record to any verifier.
This structure ensures data integrity with minimal bandwidth usage.

8.5 Validator Authentication

Validators use their registered public keys to sign blocks and consensus messages.

When a block is proposed, all other validators verify its signature before accepting it.
This mechanism prevents identity spoofing and ensures that only legitimate validators can produce blocks.
Each validator must also use separate keys for block production and network communication.
This separation minimizes the risk of compromise and allows key revocation without disrupting network operations.

8.6 Slashing and Misbehavior Detection

Security enforcement in Aria is strengthened by slashing rules.
If a validator signs two conflicting blocks or attempts to manipulate the network, the protocol automatically triggers a slashing event.
A fixed penalty of fifty thousand ARIA is deducted from the validator's stake.
The slashed amount is transferred to the treasury wallet.
All evidence of misbehavior is permanently stored on-chain, making the process transparent and verifiable.

8.7 Network Encryption

All peer-to-peer communication between nodes is encrypted using session-based keys.
Handshake protocols ensure mutual authentication and prevent replay or man-in-the-middle attacks.
Data exchange between peers, including blocks, transactions, and consensus messages, is protected from tampering during transmission.

8.8 Security Guarantees

The combined cryptographic design provides the following guarantees:

Only valid and authorized transactions can modify the blockchain state.
No node can alter the data without detection.
All blocks are traceable to their validator signatures.
All account and staking data can be verified independently.
Consensus remains secure even if up to one-third of validators act maliciously.

These guarantees ensure that Aria Chain remains secure, auditable, and tamper-proof under all operating conditions.

9. Network and Communication

Aria Chain operates as a fully decentralized peer-to-peer network.
There are no central servers or authorities. Every node communicates directly with others to exchange blocks, transactions, and consensus messages.
The network layer ensures that all participants have the same view of the blockchain over time, even under high load or partial connectivity.

9.1 Peer Discovery

When a node starts, it connects to known bootstrap peers to discover other nodes in the network.

Once connected, it exchanges peer lists and builds a view of the network topology.
Each node maintains a small set of active peers for efficient communication and periodically updates this list to replace inactive ones.
The discovery process is based on a distributed hash table model.
This design allows new nodes to join without requiring any centralized registry.

9.2 Message Propagation

All messages in the network follow a gossip protocol.
When a node receives new information, such as a transaction or block, it forwards it to a random subset of its peers.

$$\forall\ N_i \in Network : P\ (N_i\ receives\ M) \rightarrow 1ast \rightarrow \infty$$

Those peers then forward it again, ensuring rapid distribution without excessive bandwidth use.
The gossip layer guarantees that every valid message eventually reaches all active nodes, while duplicates and invalid messages are filtered out automatically.
Each message is verified upon arrival, and only verified data is relayed further.

9.3 Block Propagation

When a validator produces a new block, it immediately broadcasts it to its connected peers.
Each receiving node validates the block by checking its signature, parent reference, and state hash.
If the block passes verification, it becomes part of the node's local chain and is relayed to additional peers.
To prevent network overload, Aria nodes apply a relay limit, meaning each node only forwards a given block a limited number of times.
This ensures that propagation is fast but controlled.

9.4 Transaction Distribution

Transactions are distributed across the network in the same way as blocks.
When a node receives a new transaction, it verifies the signature and checks for duplicate nonces.
If valid, it stores the transaction in its local pool and relays it to peers.
Invalid transactions are ignored and not propagated.
The deterministic fixed-fee model ensures that all transactions are treated equally.
No priority or bidding mechanism exists, which eliminates mempool congestion attacks.

9.5 Synchronization

When a new node joins the network, it begins by requesting the latest block headers from its peers.
It verifies each header's linkage to ensure the chain is valid.
After confirming the most recent finalized block, the node downloads missing blocks in sequence until it reaches full synchronization.
The sync process includes two stages:
fast sync, which downloads headers and proofs to quickly reach the current state, and full sync, which reconstructs the complete state from block execution.
Nodes can choose either mode depending on available resources.

9.6 Network Reliability

Aria's communication layer is built for fault tolerance.
If peers disconnect or go offline, the node automatically discovers and connects to new ones.
Block and transaction data are retransmitted until confirmation is received, minimizing the risk of data loss.
The protocol uses timeouts and retry logic to handle temporary failures.
This design keeps the network consistent even under unstable conditions or partial network splits.

9.7 Security and Privacy

All peer connections are encrypted using session-based keys.
Handshake protocols verify peer identities and prevent malicious nodes from impersonating others.
Each message includes integrity checks to detect any tampering during transmission.
Nodes never reveal private keys or internal state data.
Only necessary metadata is shared for synchronization and consensus, protecting user privacy.

9.8 Finality Synchronization

Once a block becomes finalized through the consensus layer, all nodes are notified through a dedicated message type.
Nodes that were behind can quickly update to the finalized chain head without replaying the full history.

$$\sigma_{local} \rightarrow \sigma_{final}\ if\ hash_{head}\ (local) = hash_{head}\ (final)$$

This mechanism keeps the network synchronized and ensures that all participants agree on the same final state.
Aria's network architecture ensures decentralized communication, fast propagation, and high reliability.
Through encrypted gossip, automatic discovery, and deterministic validation, every node maintains the same consistent and verifiable view of the blockchain.

10. Runtime Implementation

The runtime is the core logic of Aria Chain.
It defines how the blockchain processes transactions, updates state, distributes rewards, and enforces rules.
All runtime code is executed inside a WebAssembly (WASM) environment, ensuring determinism and platform independence.

10.1 WebAssembly Execution

Aria's runtime is compiled into WebAssembly to guarantee that every node executes exactly the same instructions.

WASM provides a sandboxed execution model that isolates runtime logic from the operating system.
This prevents non-deterministic behavior and ensures that results are identical across all nodes regardless of hardware or software differences.
When a block is received, the node loads the runtime code and executes each transaction through the WASM interface.
The output is the new state of the chain, which is then verified against the expected hash recorded in the block header.

10.2 Modular Design

The runtime is built from independent modules called pallets.
Each pallet handles a specific part of the blockchain logic, such as balances, staking, governance, or treasury.
This modular structure allows developers to modify or extend functionality without affecting the rest of the system.

The main runtime modules in Aria include:

Balances pallet: manages user accounts and token transfers
Staking pallet: handles validator selection and reward distribution
Governance pallet: manages proposals and voting
Treasury pallet: controls fund allocation
Consensus pallet: integrates validator scheduling and block production rules

These modules interact through well-defined interfaces, ensuring that updates remain compatible and predictable.

10.3 Deterministic Logic

The runtime executes strictly deterministic operations.
All arithmetic, storage, and execution paths are fully defined and contain no external randomness.
System time, network data, or external APIs are never used inside the runtime.
Given the same inputs, every node must always produce the same result.
This deterministic guarantee is the foundation of Aria's state consistency across the network.

10.4 Genesis Configuration

The genesis configuration defines the initial state of the blockchain.
It includes account balances, validator lists, treasury allocation, and runtime constants such as transaction fees and reward rates.
Once the genesis block is created, it becomes the starting point for all future state transitions.
All parameters in the genesis configuration are verifiable and stored permanently in the chain's history.
Changing them requires an approved governance proposal and a runtime upgrade.

10.5 Runtime Upgrades

Aria supports seamless upgrades without hard forks.
When the governance system approves an upgrade proposal, the new runtime code is uploaded to the chain.
Validators automatically switch to the new version at the beginning of the next epoch.
This design allows the network to evolve safely while keeping all nodes synchronized.
The version number of the runtime increases with each upgrade, and nodes verify the integrity of the new code before accepting it.

10.6 Constants and Parameters

The runtime stores all critical values as on-chain constants.
These include transaction fees, slashing penalties, reward rates, and staking limits.

$$C = \{F_{tx}, P_{slash}, R_{reward}, L_{stake}\}$$

By keeping these values inside the runtime, Aria ensures that no external configuration or off-chain dependency can alter network behavior.
When governance decides to change a constant, the update is performed through a runtime upgrade, preserving determinism and traceability.

10.7 Testing and Verification

Before deployment, every runtime version undergoes deterministic simulation testing.
The system runs predefined transaction sequences on multiple environments and checks that all nodes produce identical outputs.

$$\forall\ E_1, E_2 : \Gamma_E_1(\sigma, B) = \Gamma_E_2(\sigma, B)$$

This method verifies correctness and prevents non-deterministic errors from entering production.
Runtime logic is also auditable, as its source code is open and verifiable.
Anyone can compile the code to WASM and confirm that it matches the version running on the network.

11. Appendix

The appendix provides supporting information for the Aria Chain Yellow Paper.
It includes parameter definitions, key constants, governance examples, and testing notes.
This section serves as a reference for developers, auditors, and researchers who need a complete view of the system's operational details.

12. System Parameters

The following constants define the operational limits and economics of Aria Chain:

Total token supply: 200,000,000 ARIA
Fixed transaction fee: 5 ARIA
Stake rate: 4 percent annually

Fee distribution: 20 percent to validators, 80 percent to treasury
Slashing penalty: 50,000 ARIA
Minimum validator quorum for proposals: 3 validators
Minimum approval votes: 2 yes votes
Consensus tolerance: up to one third of validators may fail without breaking safety

All parameters are enforced by the runtime and cannot be changed without an approved governance proposal and runtime upgrade.

12.1 Governance Example

Example scenario of an upgrade proposal:

A validator submits a proposal to increase the staking reward rate from 4 percent to 5 percent.
The proposal is recorded on-chain and requires at least three validators to vote.
Two validators vote yes, one votes no.
The proposal passes because it meets the quorum and majority rule.
The new reward rate is scheduled for activation at the next epoch transition.
The runtime upgrade occurs automatically and is verified by all nodes.

This example illustrates the deterministic and rule-based nature of governance within the Aria protocol.

12.2 Example of Slashing Event

If a validator signs two conflicting blocks during the same slot, a slashing event is triggered.
The system deducts 50,000 ARIA from the validator's stake and transfers the amount to the treasury wallet.
The validator's reputation score decreases, and the validator may be temporarily suspended from block production.
All details of the incident are recorded permanently in the chain history.

12.3 Example of State Update

Example of a single transaction flow:
A user sends 1000 ARIA to another address.
The transaction includes a fixed fee of 5 ARIA.
Validators process the transaction and update both accounts in the state.
The sender's balance decreases by 1005 ARIA, the receiver's increases by 1000 ARIA.
One ARIA is credited to the block producer, and four ARIA are transferred to the treasury.
The new state is hashed and stored in the next block header.
This example demonstrates the deterministic accounting model of Aria Chain.

12.4 Testnet and Simulation

Before mainnet deployment, all versions of Aria Chain undergo a full simulation on the test network.
The testnet reproduces block production, validator rotation, staking rewards, and slashing conditions under controlled parameters.
The goal of testing is to confirm deterministic behavior, correct fee handling, and proper finality performance.
Simulation results are published in public reports, allowing verification of the system's stability before activation on mainnet.

12.5 Notation Reference

Simple variable naming conventions used in this document:
a: account address
b: account balance
n: account nonce
s: staking amount
r: reward balance
v: validator identifier
t: transaction
blk: block
st: state