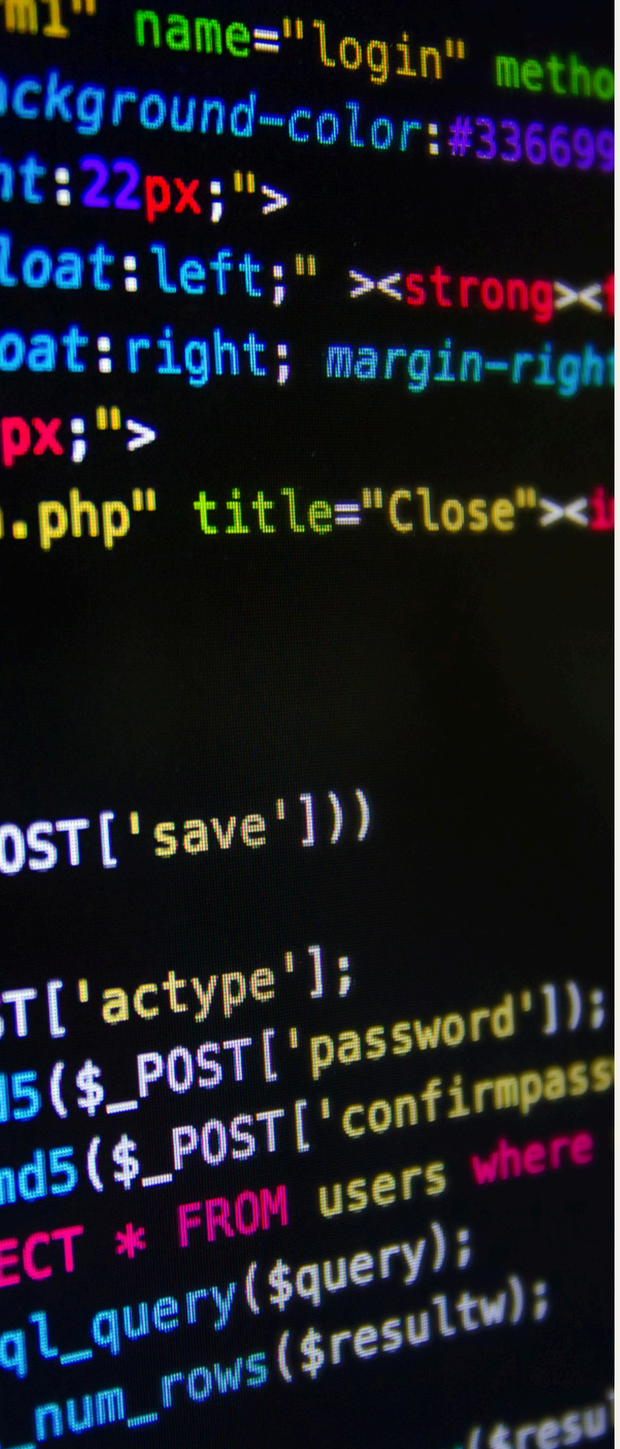
SAFEONECHAIN

DEVELOPER
PROTOCOL
SPECIFICATION
(DPS)

(SAFO)

SAFO

Version: DPS–v1.0
Status: Canonical Implementation Reference
Code Status: Code Freeze v1.0
Audit Status: External Audit Kickoff
Audience: Core Developers, Protocol Engineers, Auditors, Node Operators

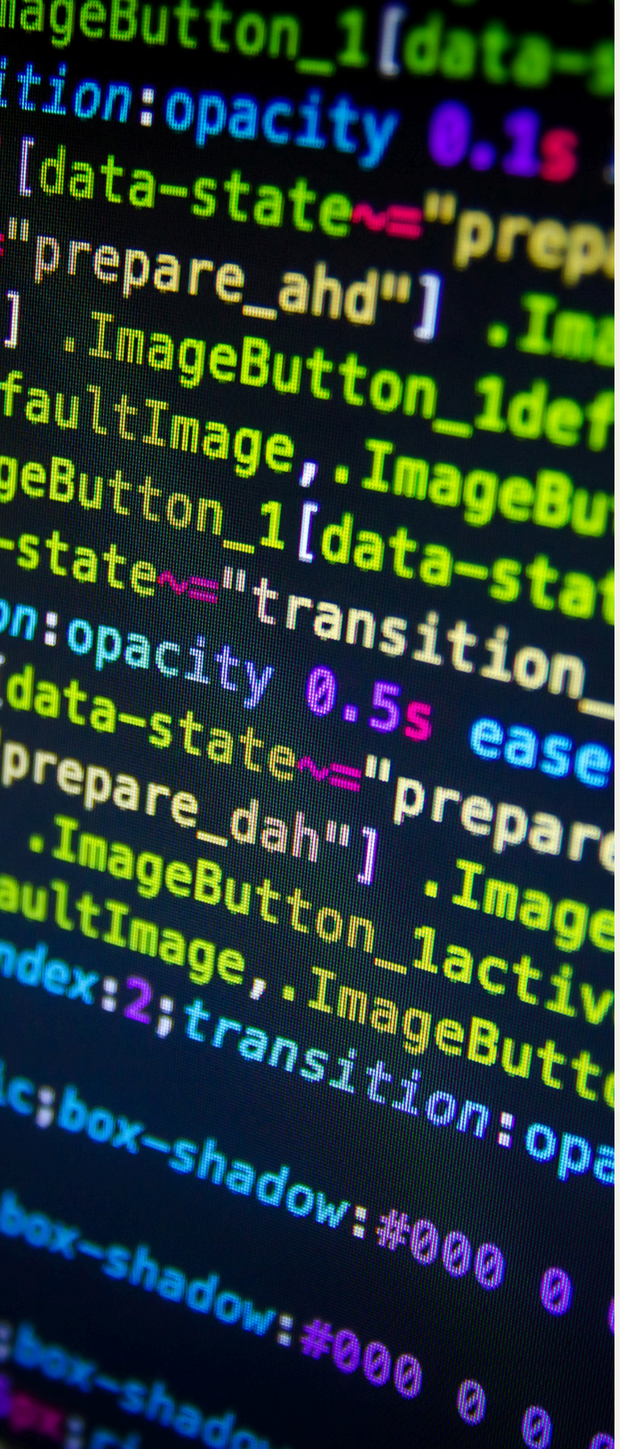


PURPOSE OF THIS DOCUMENT (DEVELOPER)

This document defines how SafeOneChain is implemented.
If there is a conflict between:

- interpretation
- blog posts
- marketing material
- secondary summaries

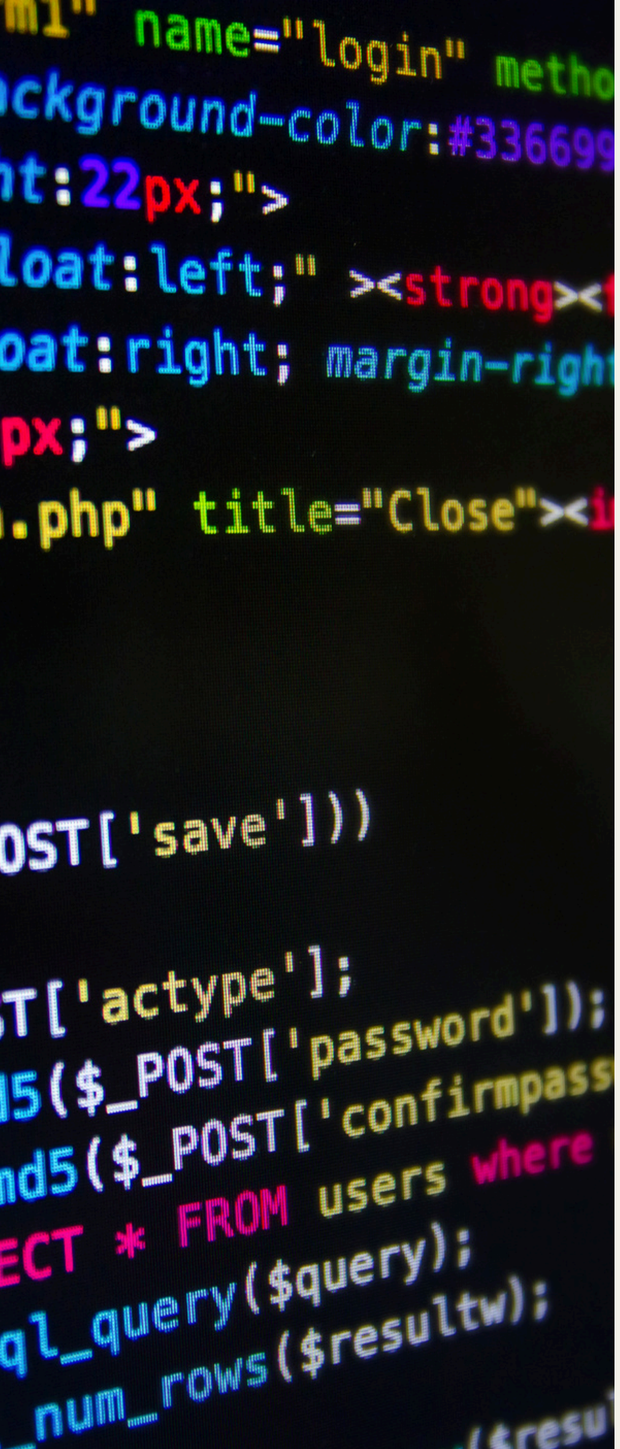
this document wins.
Any implementation claiming SAFO compatibility must conform exactly to this specification.



REPOSITORY LAYOUT (NORMATIVE)

```
rusk/
├── consensus/ # PoA-BFT state machine
├── extradata/ # CommitProof encoding/decoding
├── execution/ # EVM integration
├── networking/ # Permissioned P2P
├── rpc/ # JSON-RPC (eth_* + safo_*)
├── governance/ # System contract bindings
├── evidence/ # Evidence detection & verification
├── storage/ # Block & state persistence
├── node/ # Node lifecycle
├── config/ # Genesis & validator config
└── bin/ # Node binary
```

No module may bypass another module's authority.



CORE PROTOCOL CONSTANTS (FROZEN)

// consensus/constants.rs

```
pub const COMMIT_QUORUM_NUMERATOR: u64 = 2;
pub const COMMIT_QUORUM_DENOMINATOR: u64 = 3;
pub const EXTRA_DATA_MAGIC: [u8; 4] = *b"SAFO";
pub const EXTRA_DATA_VERSION_V1: u8 = 0x01;
pub const MAX_VALIDATORS: usize = 100;
pub const SIGNATURE_SIZE: usize = 65;
pub const VALIDATOR_ID_SIZE: usize = 20;
```

These constants are protocol-critical.Changing them requires a new protocol version and re-audit.

PoA-BFT Consensus – Minimal Implementable State Machine

State Definitions

```
// consensus/state.rs
pub enum ConsensusStep {
 Propose,
 Prevote,
 Precommit,
 Commit,
}
```

Only one state may be active per block height.

Consensus Context

```
// consensus/context.rs
pub struct ConsensusContext {
 pub height: u64,
 pub round: u32,
 pub step: ConsensusStep,
 pub proposer: ValidatorId,
 pub votes: VoteSet,
}
```

Vote Structure

```
// consensus/vote.rs
pub struct Vote {
 pub height: u64,
 pub round: u32,
 pub block_hash: Option<Hash>,
 pub validator: ValidatorId,
 pub signature: Signature,
}
```

A validator may never emit two Precommit votes for the same height.

Commit Condition (Normative)

```
// consensus/quorum.rs
pub fn has_commit_quorum(
 votes: &Vec<Vote>,
 validator_count: usize,
) -> bool {
 votes.len() >= (validator_count * 2 / 3) + 1
}
```

If this returns true, the block must be finalized.

extraData CommitProof v1 — Encoding & Verification

Raw Binary Layout (Exact)

0..4	MAGIC = "SAFO"
4	VERSION = 0x01
5..9	ROUND (u32, BE)
9..41	VALSET_HASH (32 bytes)
41..43	SIG_COUNT (u16, BE)
43..*	SIGNATURE ENTRIES

Each signature entry:

| VALIDATOR_ID (20 bytes) |
SIGNATURE (65 bytes) |

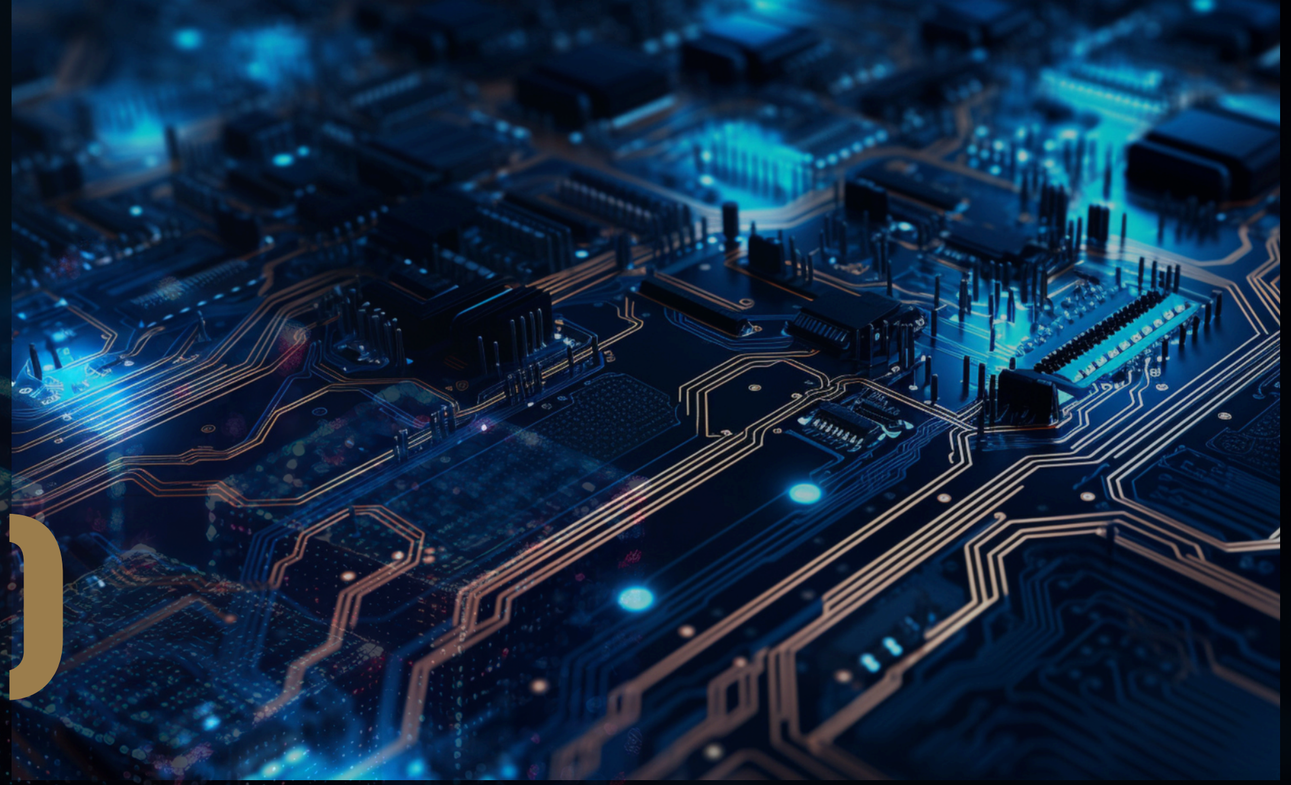
Encoding

```
// extradata/encode.rs
pub fn encode_commit_proof(
 round: u32,
 valset_hash: [u8; 32],
 signatures: Vec<(ValidatorId,
Signature)>
) -> Vec<u8> {
 let mut out = Vec::new();
out.extend_from_slice(&EXTRA_DATA
_MAGIC);
 out.push(EXTRA_DATA_VERSION_V1);
out.extend_from_slice(&round.to_be_b
ytes());
 out.extend_from_slice(&valset_hash);
 out.extend_from_slice(&
(signatures.len() as u16).to_be_bytes());
```

Verification

```
// extradata/verify.rs
pub fn verify_commit_proof(
 header_hash_no_extradata: Hash,
 proof: CommitProof,
 validator_set: &ValidatorSet
) -> Result<(), Error> {
 ensure!(proof.signatures.len() >=
quorum(validator_set.len()));
 for (id, sig) in proof.signatures {
 let pubkey =
validator_set.get_pubkey(&id)?;
verify_secp256k1(header_hash_no_extrad
ata, sig, pubkey)?;
 }
 Ok(())
}
```

EVIDENCE PIPELINE V1 (DOUBLE-SIGN)



EVIDENCE STRUCTURE

```
// evidence/types.rs

pub struct DoubleSignEvidence {
 pub validator: ValidatorId,
 pub height: u64,
 pub vote_a: Vote,
 pub vote_b: Vote,
}
```

THE REFERENCE CLIENT IS:

```
// evidence/detect.rs

pub fn detect_double_sign(votes: &[Vote])
-> Vec<DoubleSignEvidence> {
 // group by validator + height
}
```

Evidence is never auto-enforced.

SYSTEM CONTRACTS V1 (SOLIDITY)

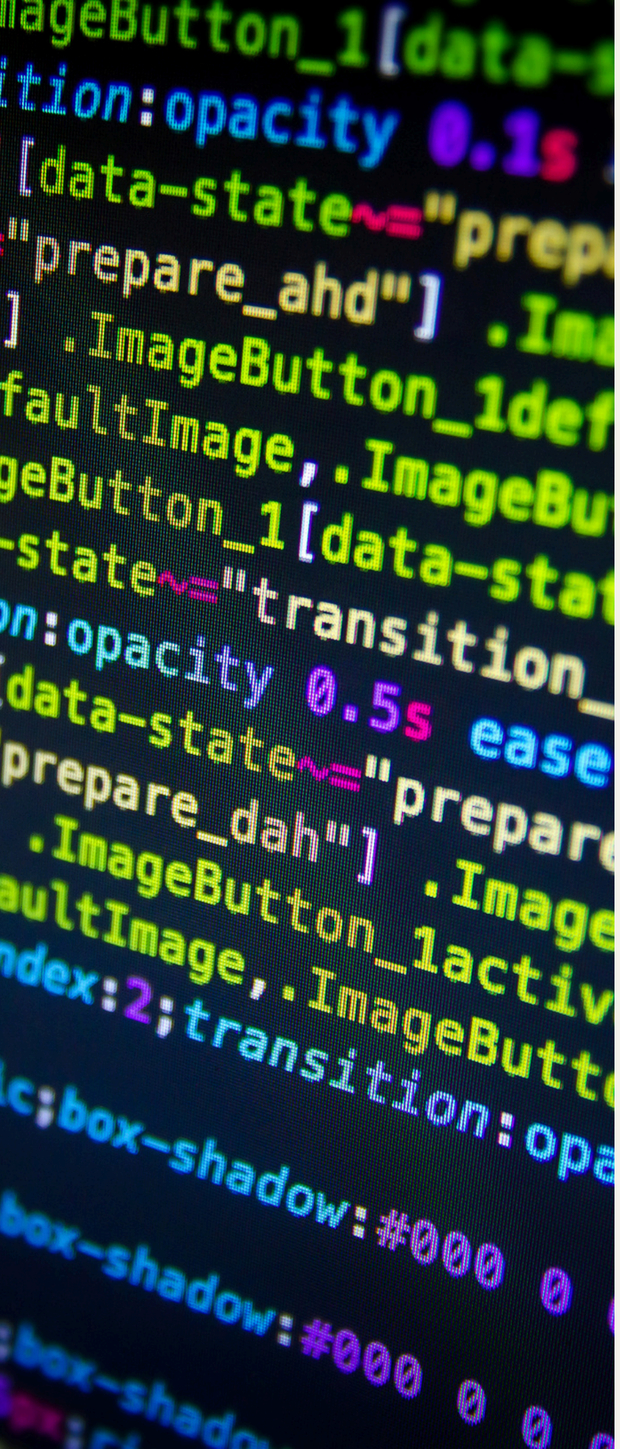
VALIDATORREGISTRY.SOL

```
contract ValidatorRegistry {
 event ValidatorAdded(address validator);
 event ValidatorPaused(address validator);
 event ValidatorRemoved(address validator);
 mapping(address => bool) public active;
 function addValidator(address v) external
onlyGovernance {
 active[v] = true;
 emit ValidatorAdded(v);
 }
}
```

GOVERNANCECONTROLLER.SOL

```
contract GovernanceController {
 uint256 public quorum;
 function approve(bytes32 action) external
onlyValidator {
 // quorum tracking
 }
}
```

No contract has a single-admin escape hatch.



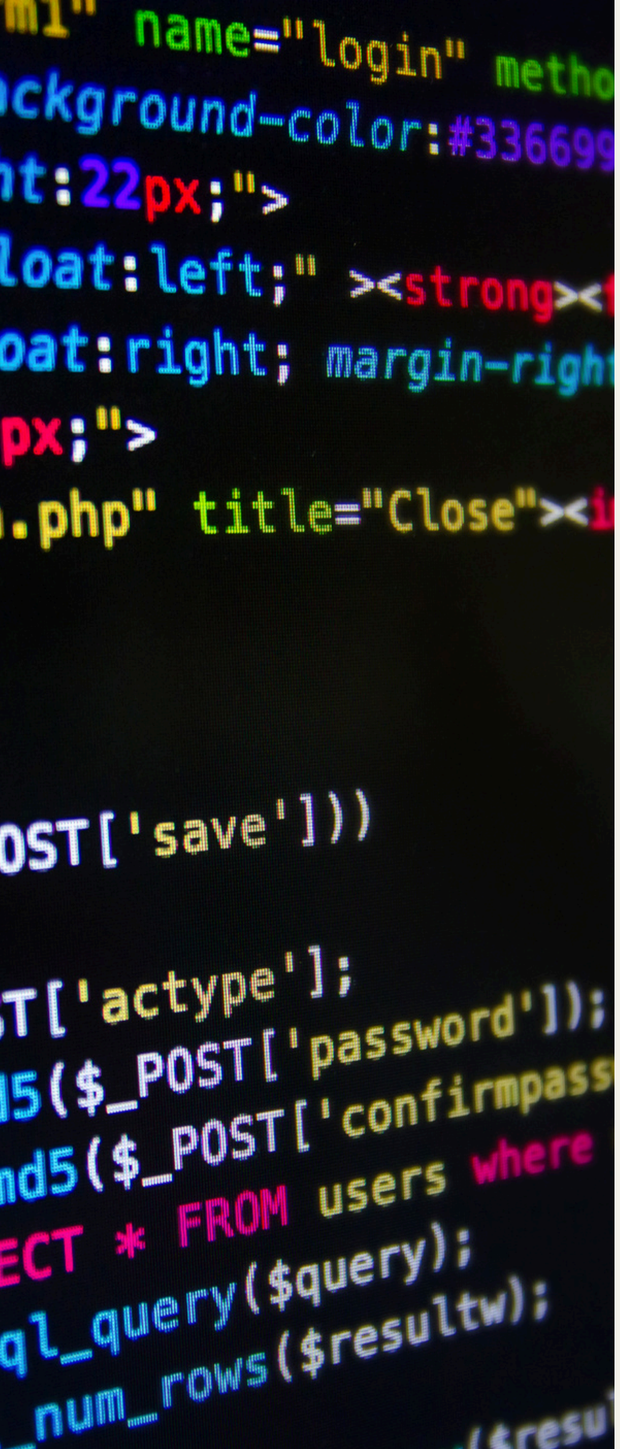
RUST ↔ SOLIDITY BINDINGS

```
// governance/bindings.rs

abigen!(
    ValidatorRegistry,
    "abi/ValidatorRegistry.json"
);
```

Bindings are:

- static
- versioned
- compile-time checked



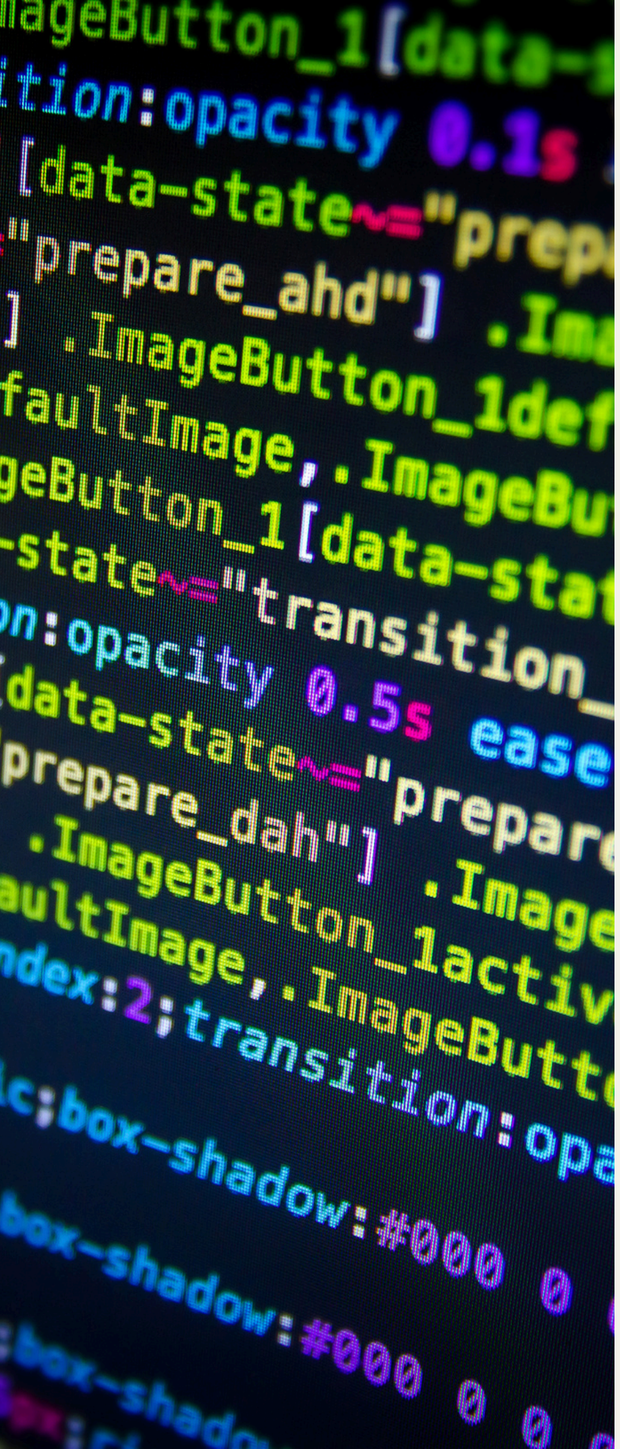
LOCAL MULTI-VALIDATOR HARNESS (DOCKER-COMPOSE)

```
version: "3.9"
services:
  v1:
    image: safo/rusk
    environment:
      - VALIDATOR_ID=1
  v2:
    image: safo/rusk
    environment:
      - VALIDATOR_ID=2
```

The harness must reproduce:

- quorum finality
- proposer failure

double-sign evidence



SAFO * RPC METHODS (EXAMPLES)

safo_getFinalityProof

```
{
  "jsonrpc": "2.0",
  "method": "safo_getFinalityProof",
  "params": ["0xBLOCKHASH"],
  "id": 1
}
```

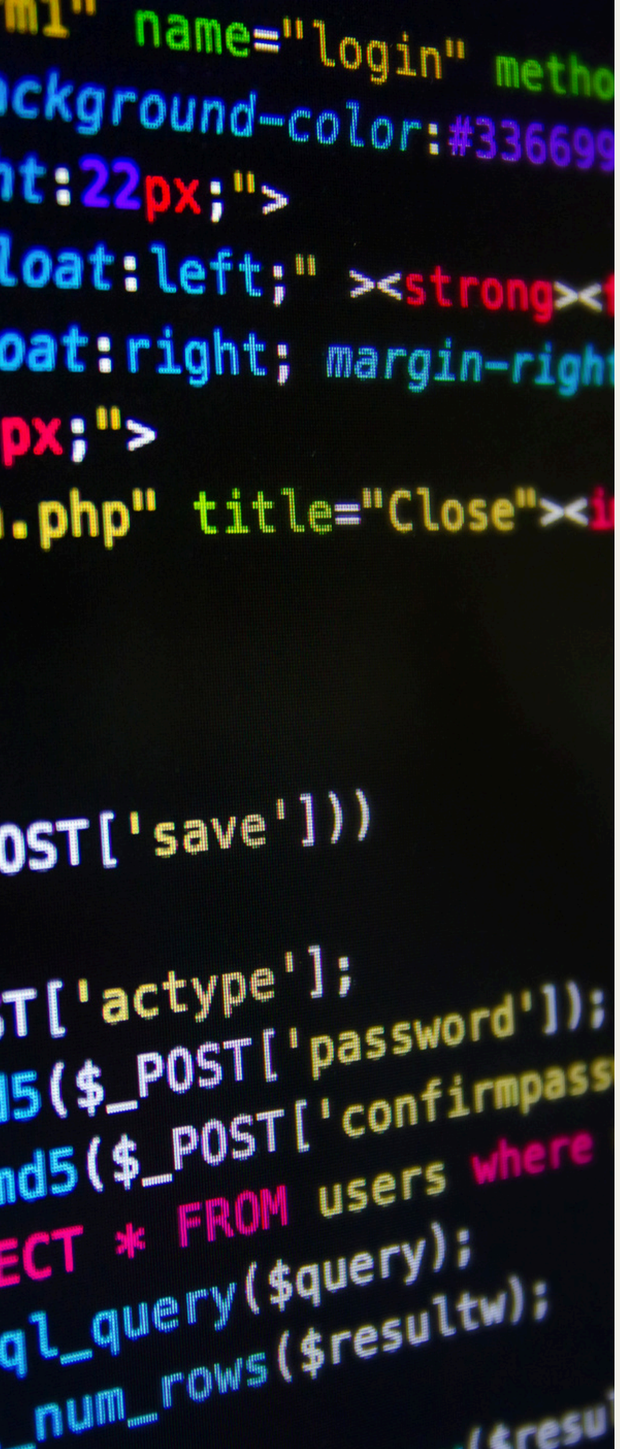
Response includes:

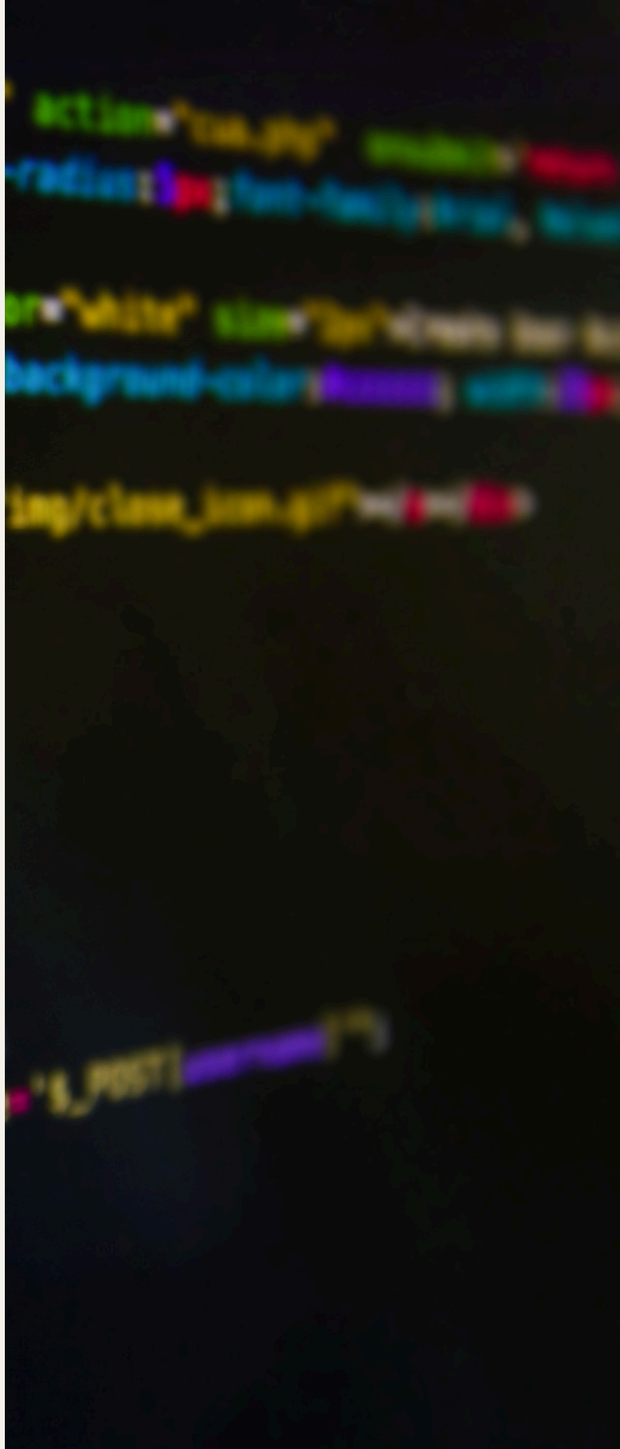
- raw commit proof bytes
- validator set hash

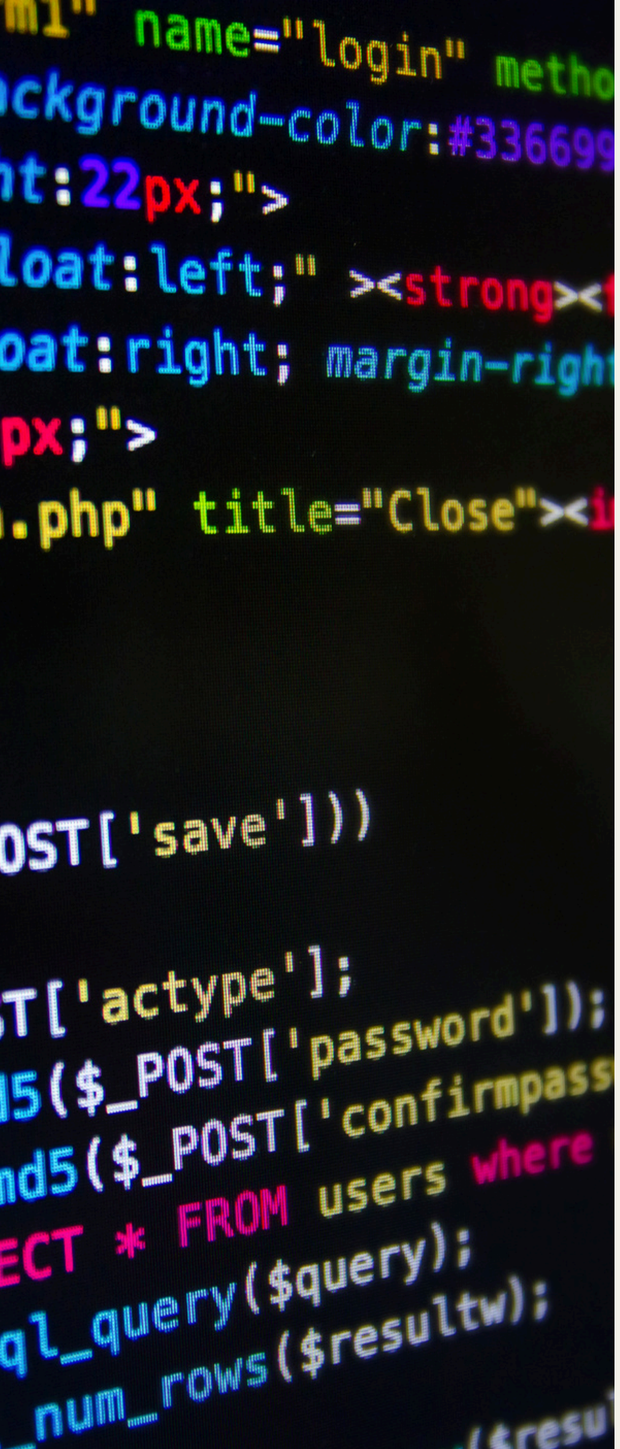
verification status

NODE OPERATIONS (DEVELOPER)

- validators must use remote signers
- keys are rotatable
- upgrades require governance approval





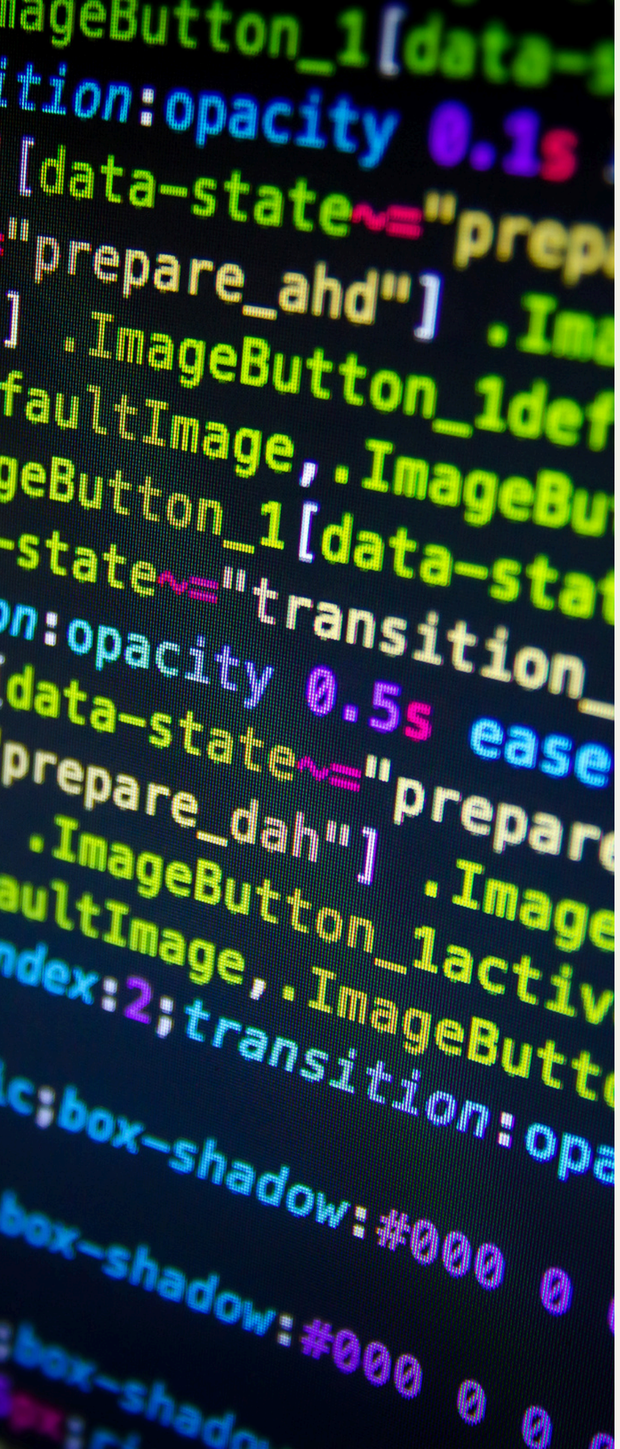


NETWORKING STACK OVERVIEW

SafeOneChain networking is permissioned, identity-bound, and role-aware.

There is no peer discovery.
All peers are explicitly configured or admitted via governance.

NODE IDENTITY



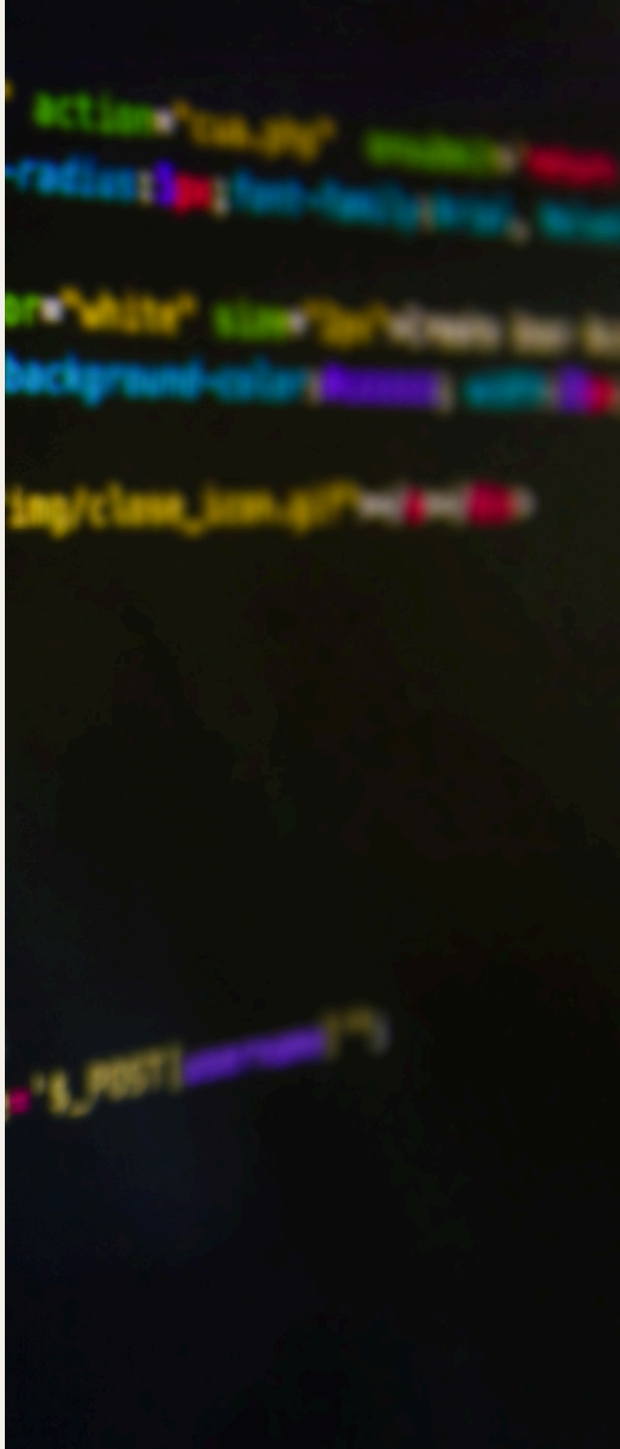
Each node has:

```
Pub struct NodeIdentity {
 Pub node_id: [u8; 32],
 Pub validator_id: Option<ValidatorId>,
 Pub role: NodeRole,
}
Pub enum NodeRole {
 Validator,
 FullNode,
 Auditor,
}
```

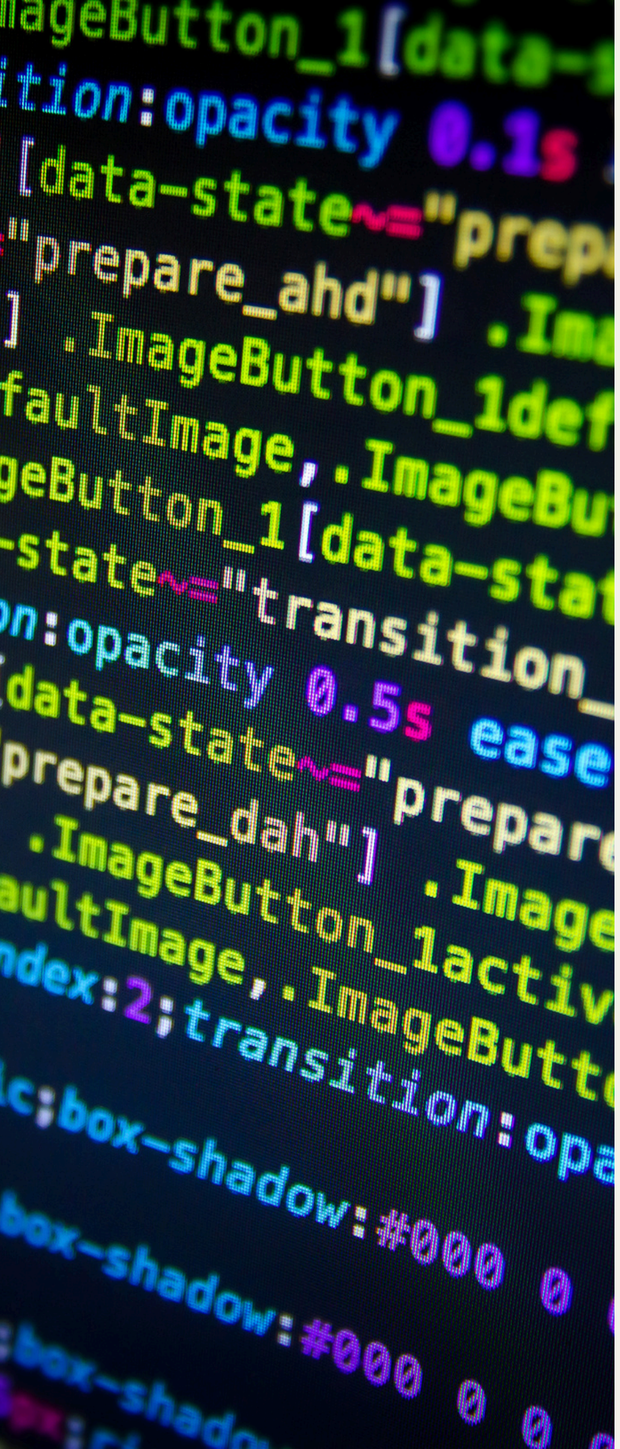
Validator → consensus + gossip

FullNode → sync + RPC

Auditor → read-only verification



HANDSHAKE PROTOCOL



Handshake must complete before any data exchange.

Handshake steps:

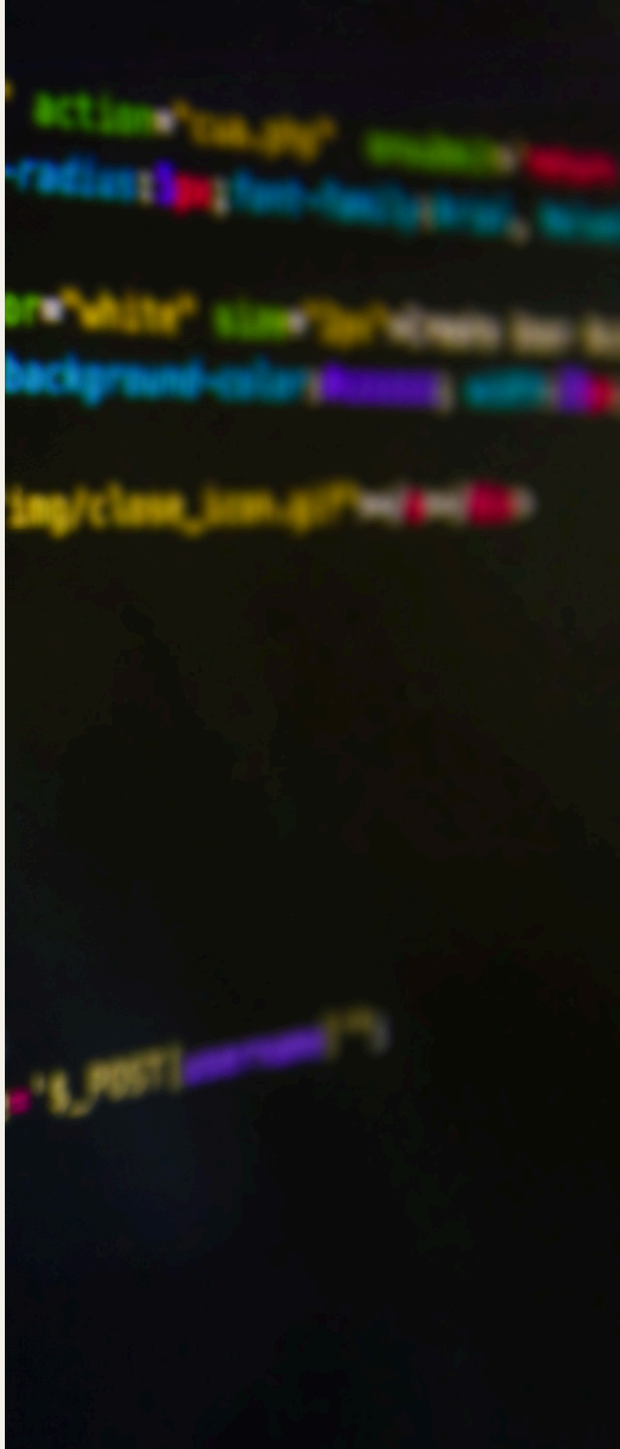
1. Node ID exchange

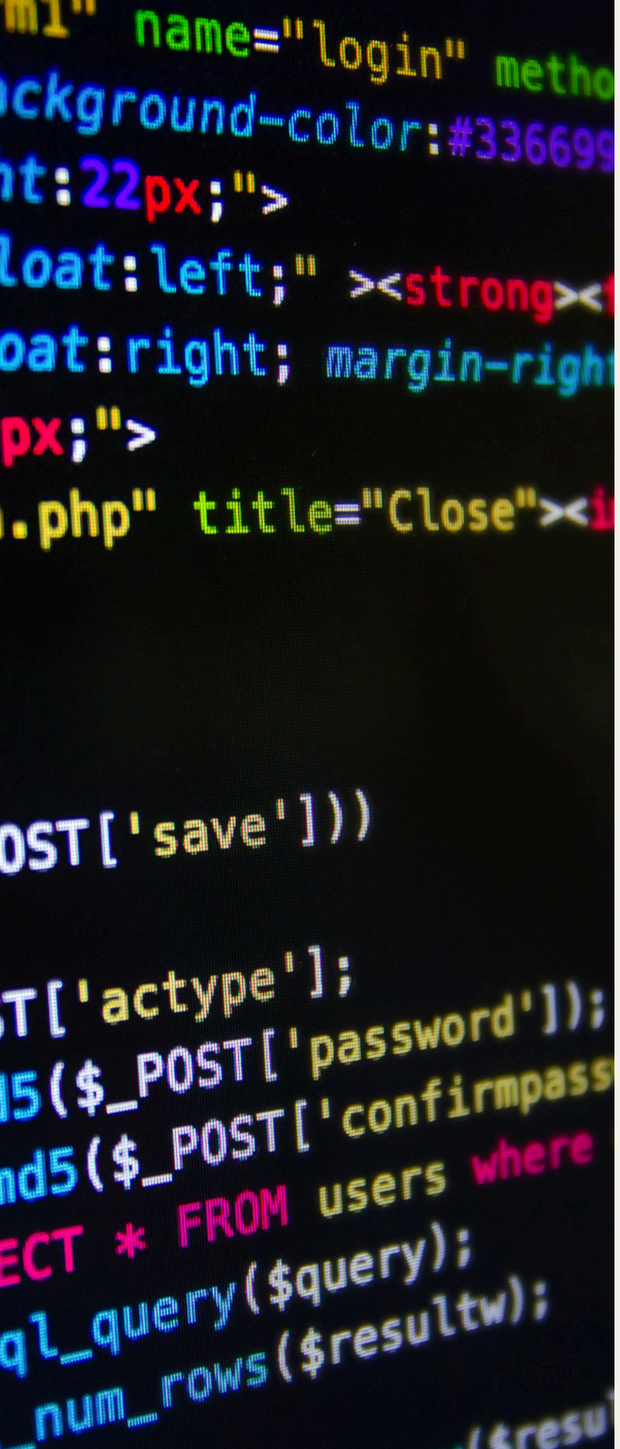
2. Role declaration

3. Signature challenge

4. Validator authorization check (if applicable)

5. Connection acceptance or rejection

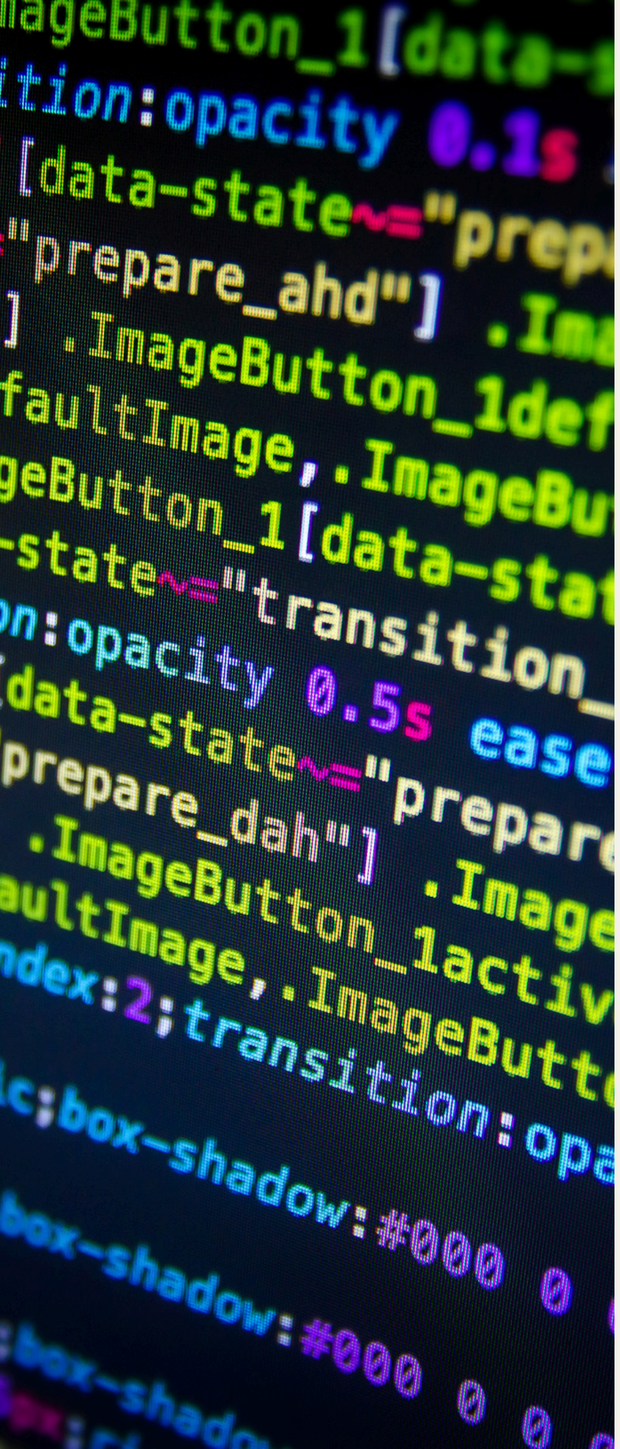




MESSAGE AUTHORIZATION MATRIX

Message Type	Validator	Full Node	Auditor
Propose	✓	✗	✗
Prevote	✓	✗	✗
Precommit	✓	✗	✗
Block Sync	✓	✓	✓
Evidence	✓	✓	✓
RPC Relay	✗	✓	✓

SENDING UNAUTHORIZED MESSAGES IS A PROTOCOL VIOLATION.



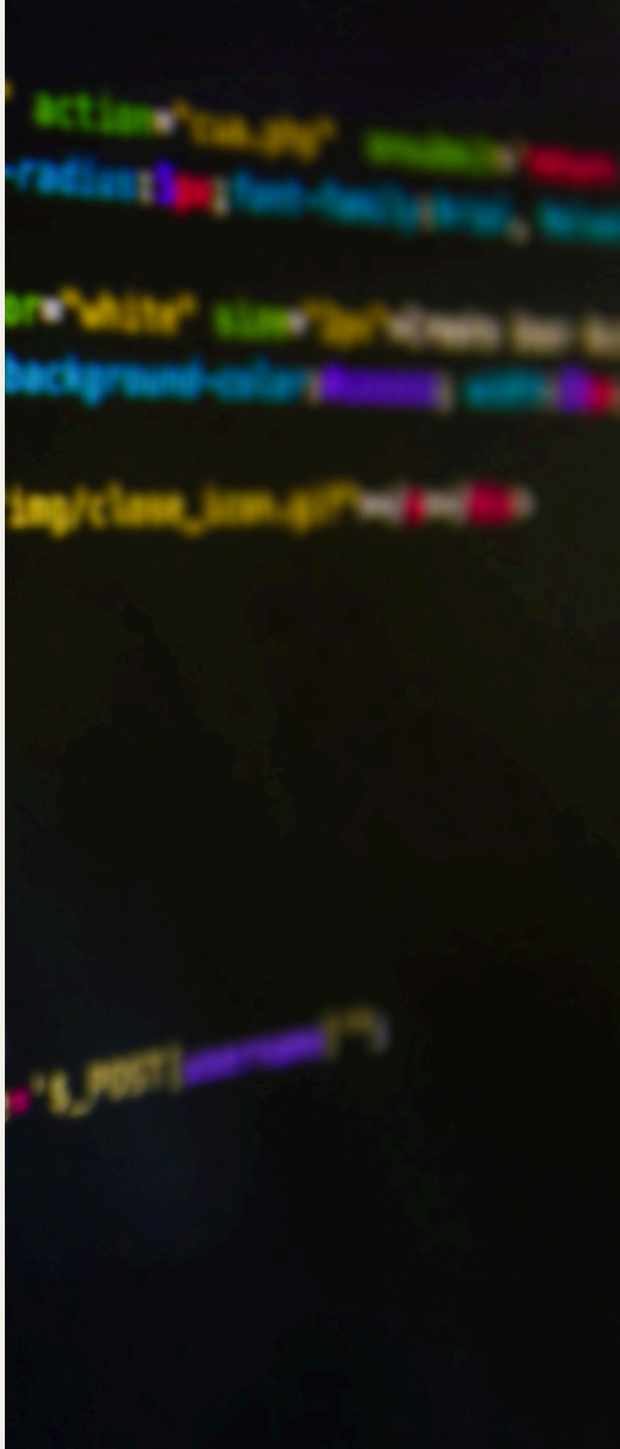
NETWORK SECURITY RULES

All messages signed

All messages validated before processing

Rate limits per peer

Malformed messages → immediate disconnect



RPC Specification (Full, Normative)

THIS SECTION DEFINES ALL RPC SEMANTICS. RPC IS FINALITY-AWARE BY DESIGN.

Global Rules

Latest always means finalized

No RPC may expose non-final state as canonical

Errors are explicit; silence is forbidden

Ethereum-Compatible RPC (eth_*)

Supported subset:

Eth_blockNumber
Eth_getBlockByHash
Eth_getBlockByNumber
Eth_getTransactionByHash
Eth_getBalance
Eth_call

Constraint:
All calls resolve against finalized state only.

SAFO-Specific RPC (safo_*)

Safo_getFinalityProof

Returns raw commit proof.

```
{
 “method”: “safo_getFinalityProof”,
 “params”: [“0xBLOCKHASH”]
}
```

Response:

```
{
 “finalized”: true,
 “commitProof”: “0xRAWBYTES”,
 “validatorSetHash”: “0x...”
}
```

RPC Specification (Full, Normative)

THIS SECTION DEFINES ALL RPC SEMANTICS. RPC IS FINALITY-AWARE BY DESIGN.

Safo_getValidatorSet

```
{
 “method”: “safo_getValidatorSet”,
 “params”: [123456]
}
```

Returns:

Ordered validator list

Public keys

Activation status

Safo_getGovernance Actions

Returns full governance history slice.

Safo_getEvidence

Returns submitted evidence objects including verification result.

RPC Error Semantics

```
{
 “code”: -32001,
 “message”: “BLOCK_NOT_FINAL”
}
```

Errors are machine-processable.

EVM Execution Layer – Decision & Integration

Execution Model

SafeOneChain integrates an EVM execution engine as a pure execution layer.

Properties:

Execution cannot influence consensus

Execution failures cannot affect finality

Gas accounting is deterministic

Integration Boundary

```
Pub trait ExecutionEngine {
 Fn execute_block(
 &self,
 Block: &Block,
 State: &mut State
 ) -> ExecutionResult;
}
```

Consensus:
Validates signatures

Finalizes blocks
Execution:
Processes transactions

Produces state root

Contract Deployment Policy

Deployment may be:

Open

Permissioned

This is a governance parameter, not a protocol constant.

Auditor Read-Only Node (Developer Guide)

Purpose

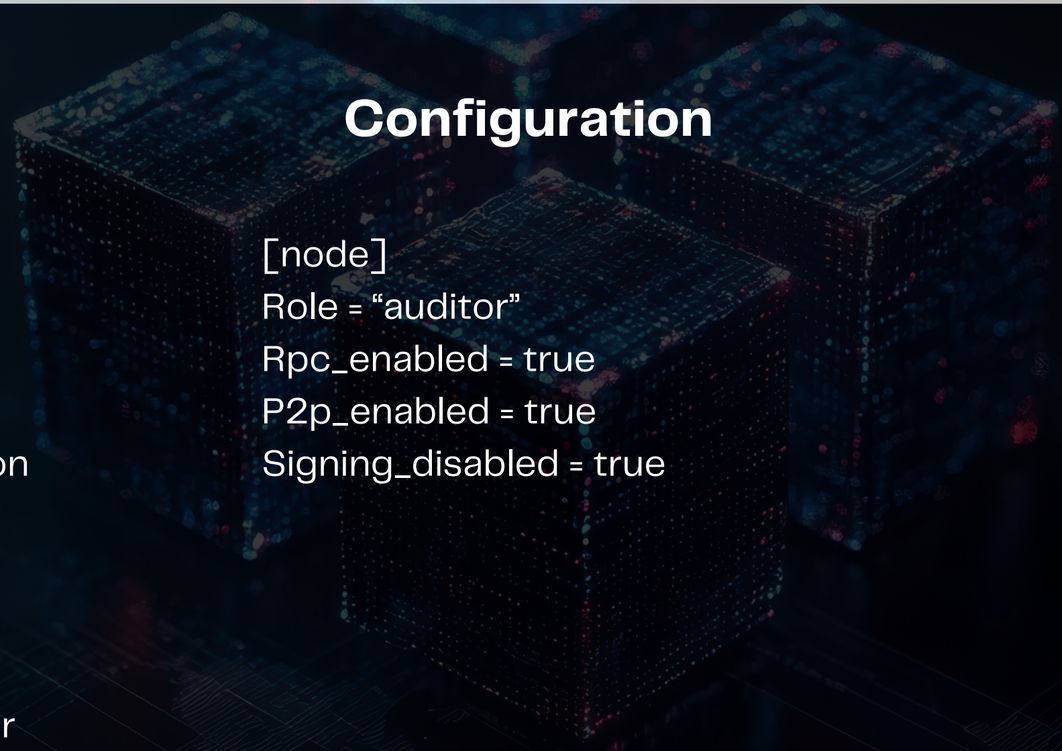
Auditor nodes enable:

Independent finality verification

Governance timeline reconstruction

Evidence validation

Auditor nodes never sign and never broadcast transactions.



Configuration

```
[node]
Role = “auditor”
Rpc_enabled = true
P2p_enabled = true
Signing_disabled = true
```

Auditor Guarantees

An auditor node can:

Verify commit proofs offline

Detect equivocation

Validate governance decisions

Without trusting operators.

External Audit Scope – Statement of Work (SOW)

THIS SECTION IS BINDING FOR AUDITORS AND DEVELOPERS.

Audit Targets

Auditors must assess:

PoA-BFT safety invariants

CommitProof correctness

Evidence pipeline correctness

Governance authority boundaries

RPC finality semantics

Must-Fail Tests

Auditors must confirm failure for:

Forged commit proofs

Duplicate validator signatures

Governance actions without quorum

Non-final blocks returned as latest

Passing these tests is mandatory.

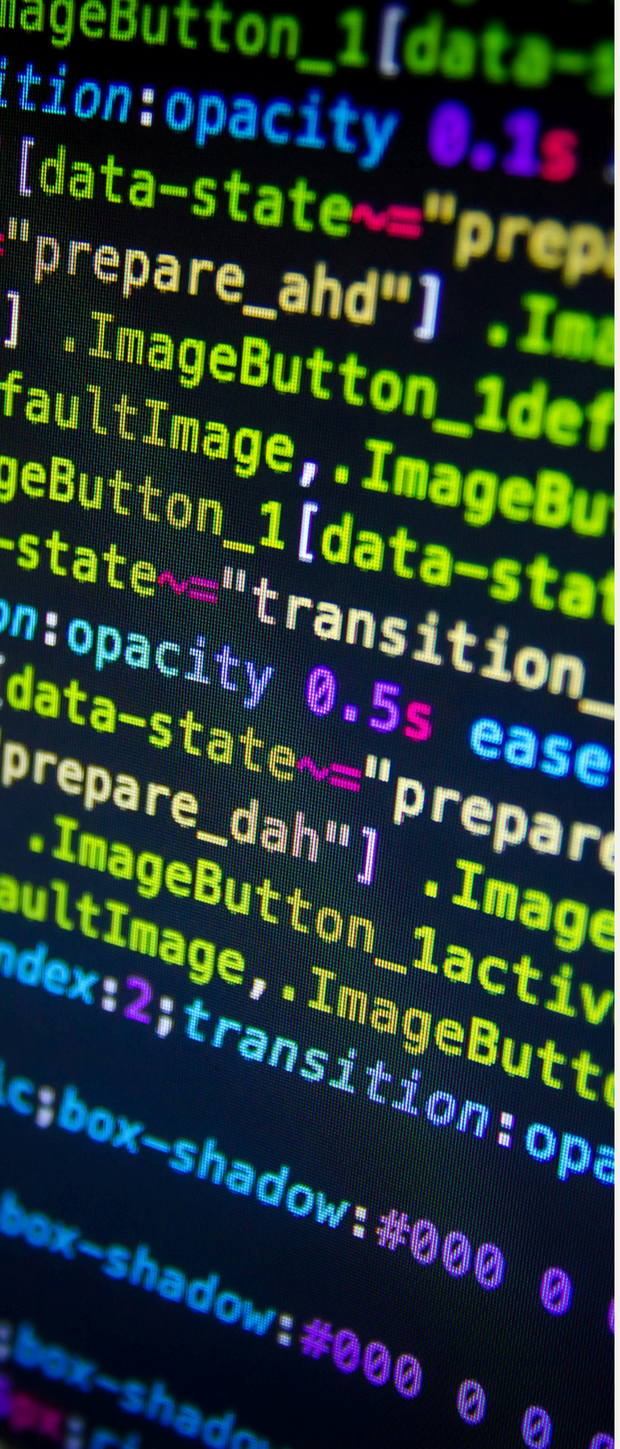
Deliverables

Written report

Reproducible PoCs

Severity classification

Explicit yes/no on finality bypass



IMMEDIATE IMPLEMENTATION – NEXT RUST COMMITS

The following commits are required before audit start:

1. Consensus/ finality enforcement

2. Extradata/ strict decoding + fuzz tests

3. Rpc/ finality-aware guardrails

4. Evidence/ deterministic detection

5. Networking/ handshake enforcement

No feature work beyond this list is permitted pre-audit.

SAFEONECHAIN — EXTERNAL AUDIT READINESS MAP (DEVELOPER)

This map is normative.
Every protocol claim must be traceable to code, tests, and on-chain evidence.

TRACEABILITY MATRIX (CANONICAL)

Protocol Claim / Module / Artifact / Verifier

Deterministic Finality consensus / quorum logic offline verifier

No Reorg of Final Blocks storage / block immutability replay harness

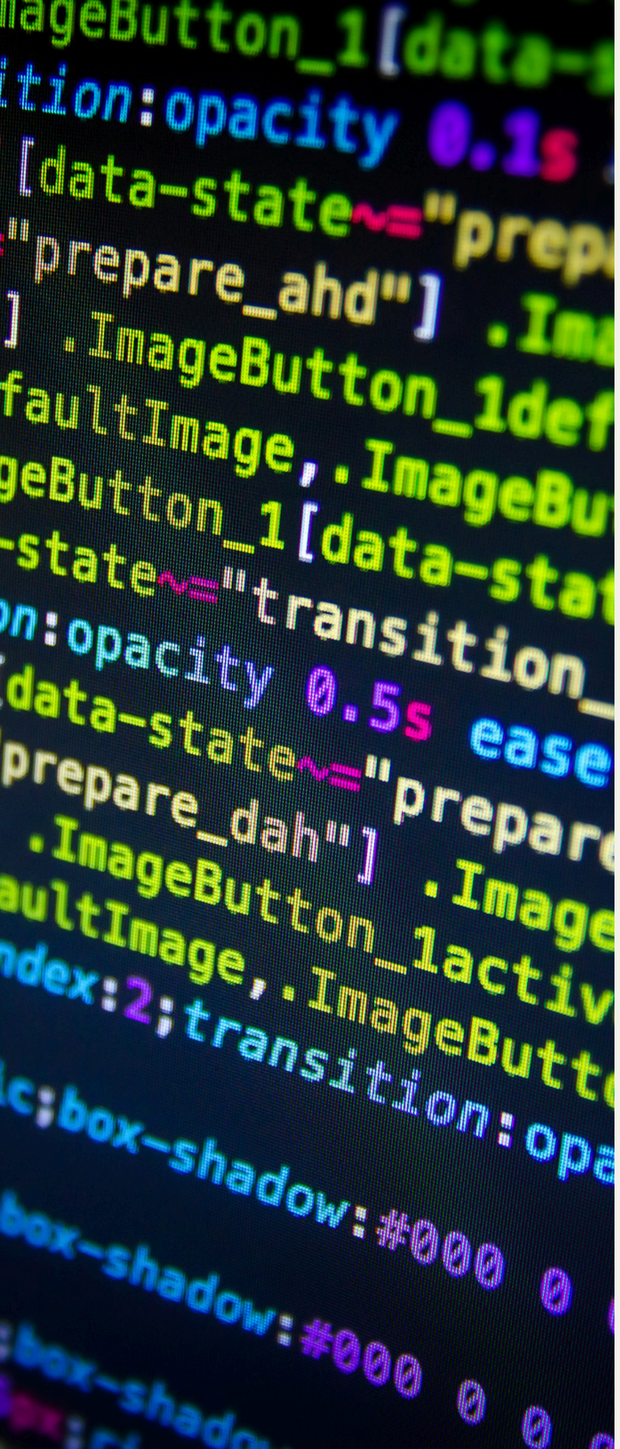
CommitProof Correctness extradata / encode/decode byte-level verifier

Validator Authority governance / system contracts RPC + events

No Auto-Slashing evidence / governance gating governance log

RPC Finality Safety rpc / guards RPC tests

Any change to one row requires re-audit.



LOCAL MULTI-VALIDATOR HARNESS (EXPANDED)

The harness is mandatory for audit and CI.
Canonical docker-compose
Version: “3.9”
Services:

Validator1:
Image: safo/rusk:latest
Environment:
– ROLE=validator
– VALIDATOR_ID=0x01

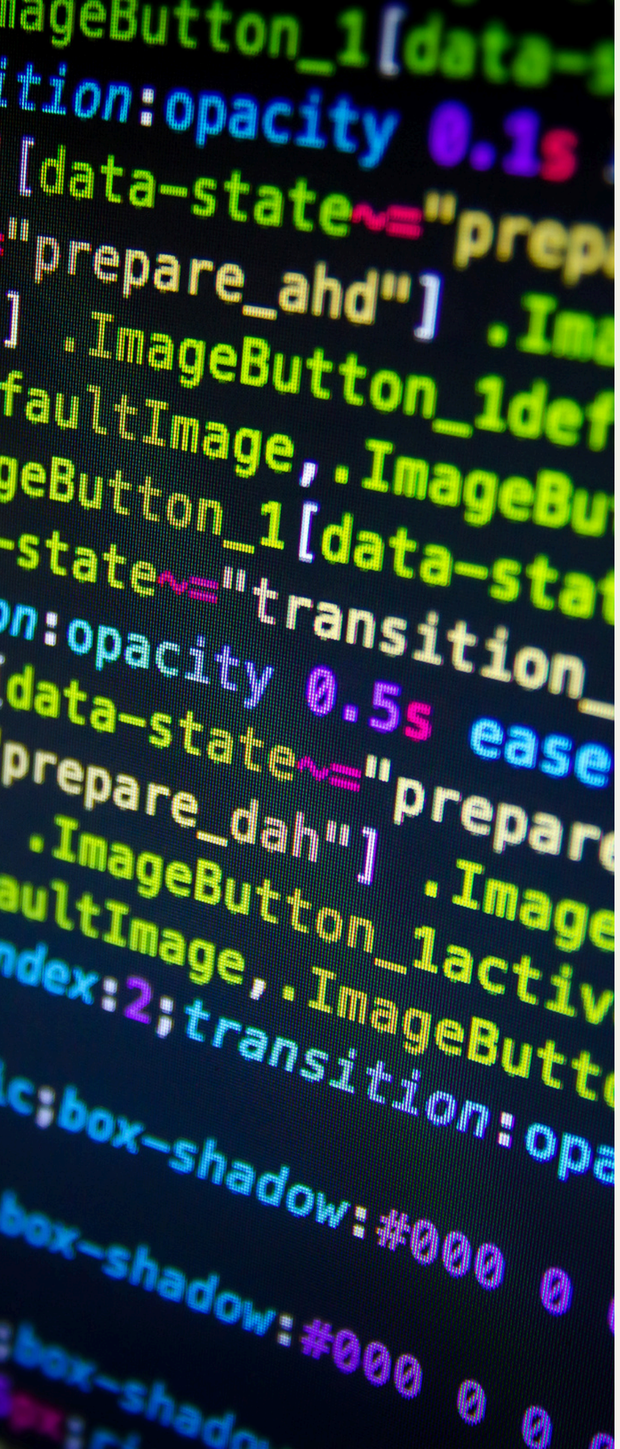
Validator2:
Image: safo/rusk:latest
Environment:
– ROLE=validator
– VALIDATOR_ID=0x02

Validator3:
Image: safo/rusk:latest
Environment:
– ROLE=validator
– VALIDATOR_ID=0x03

Validator4:
Image: safo/rusk:latest
Environment:
– ROLE=validator
– VALIDATOR_ID=0x04

Fullnode:
Image: safo/rusk:latest
Environment:
– ROLE=full

Auditor:
Image: safo/rusk:latest
Environment:
– ROLE=auditor



REQUIRED TEST SCENARIOS

The harness must reproduce:

1. Normal finality (3 of 4 validators)

2. Proposer failure

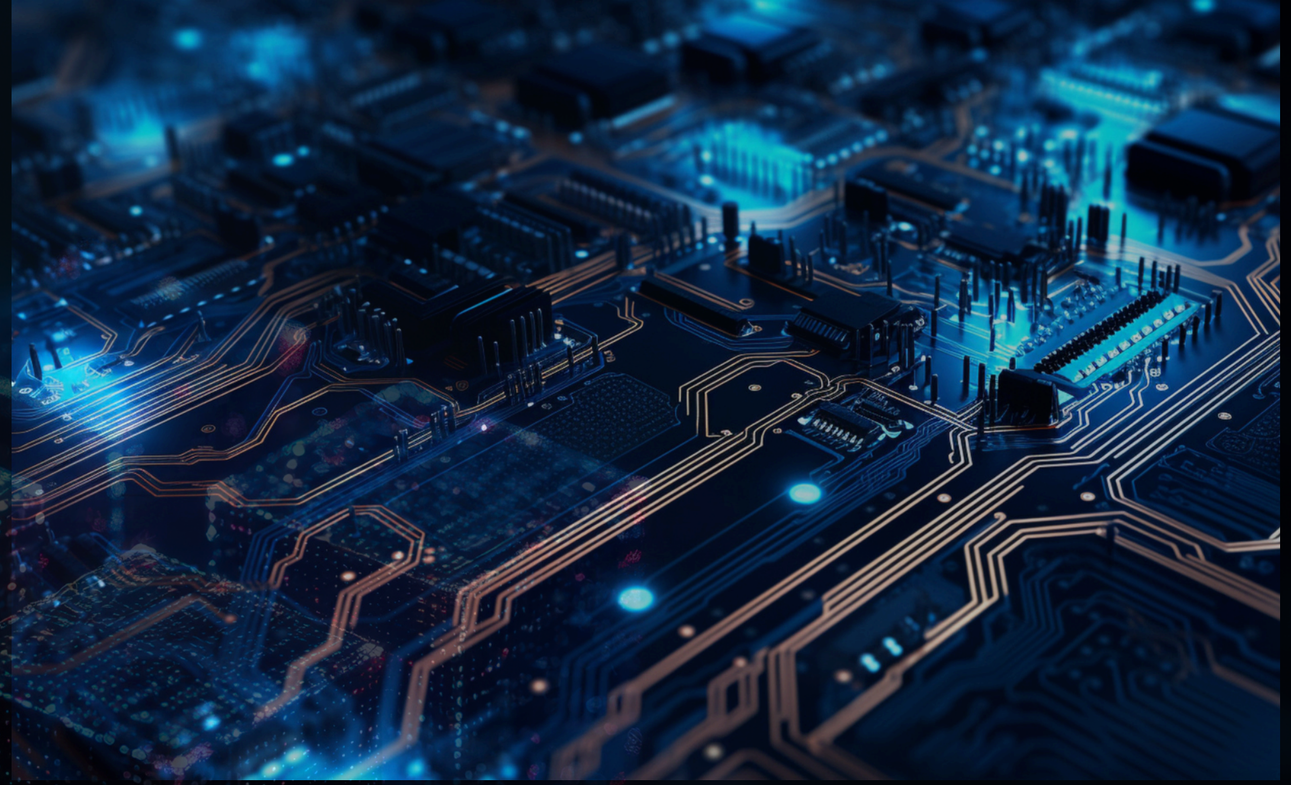
3. Double-sign evidence generation

4. Network partition (2/2 split → no finality)

5. Governance-approved validator pause

Failure to reproduce invalidates compliance.

DETERMINISTIC TEST VECTORS



COMMITPROOF VECTOR

Given:

Header hash H
Round R
Validator set hash V

Expected:

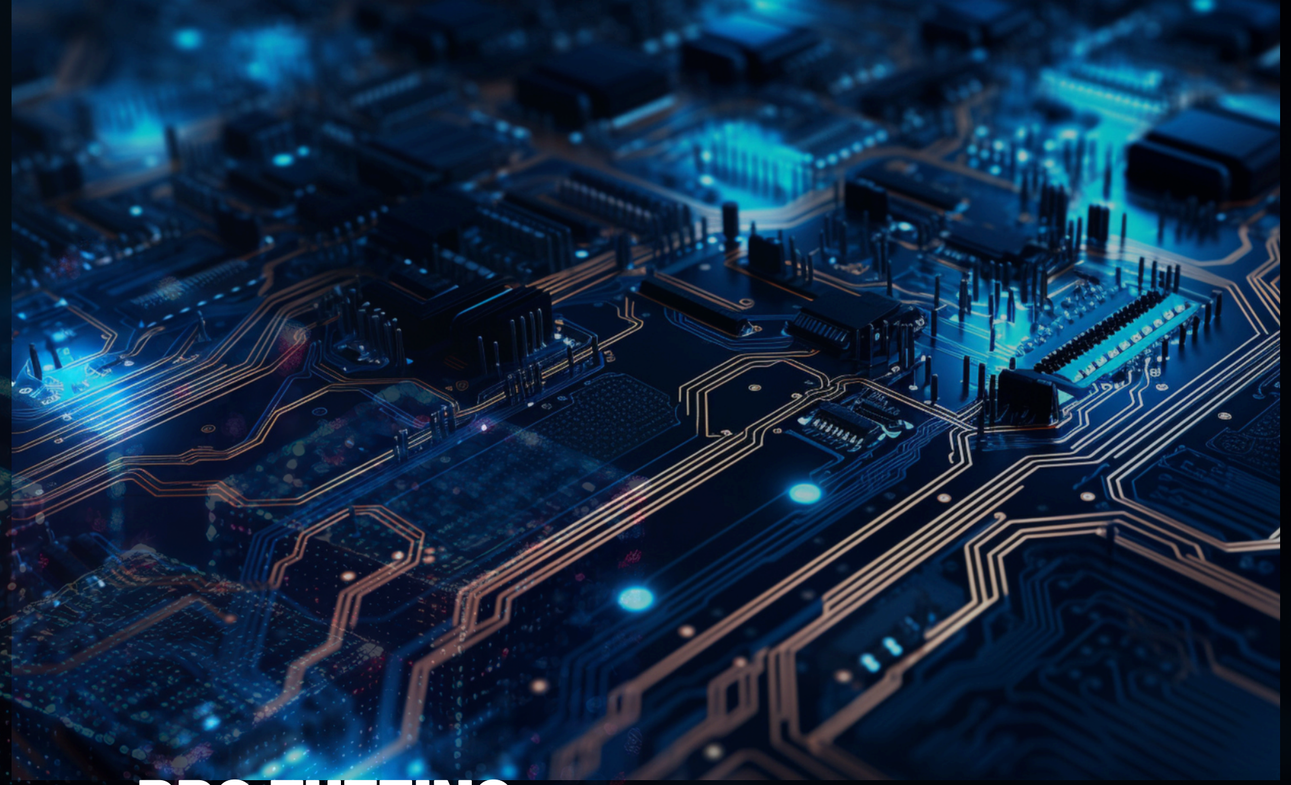
Deterministic CommitProof bytes

Identical across all implementations

EVIDENCE VECTOR

Given:
Two conflicting precommit votes
Same validator
Same height
Expected:
Identical evidence object hash
Governance-processable

FUZZING STRATEGY (MANDATORY)

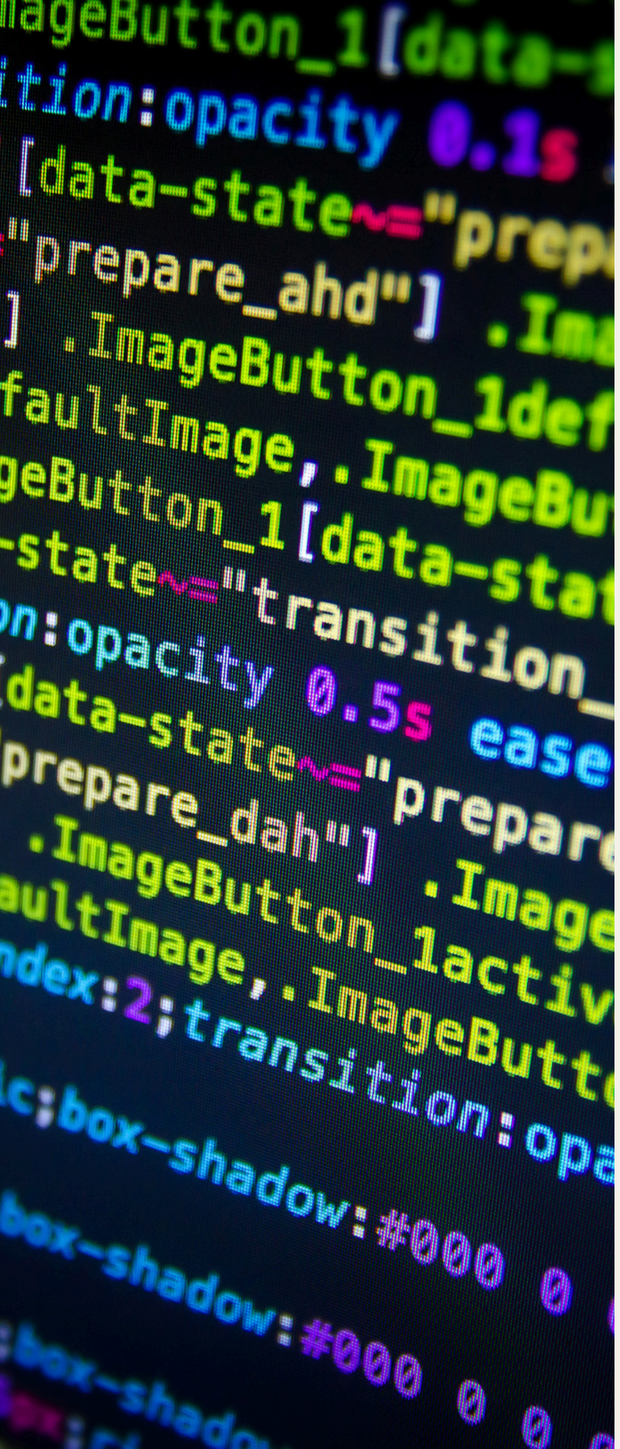


COMMITPROOF FUZZING

Targets:
Malformed MAGIC
Wrong VERSION
Truncated signatures
Duplicate validator IDs
Incorrect SIG_COUNT
All must be rejected before consensus impact.

RPC FUZZING

Targets:
Non-final block queries
Invalid parameter types
Large payloads
Replayed requests
RPC must:
Reject explicitly
Never return non-final state



SECURITY INVARIANTS (DEVELOPER-ENFORCED)

The following invariants are checked in code:

1. A finalized block cannot be overwritten

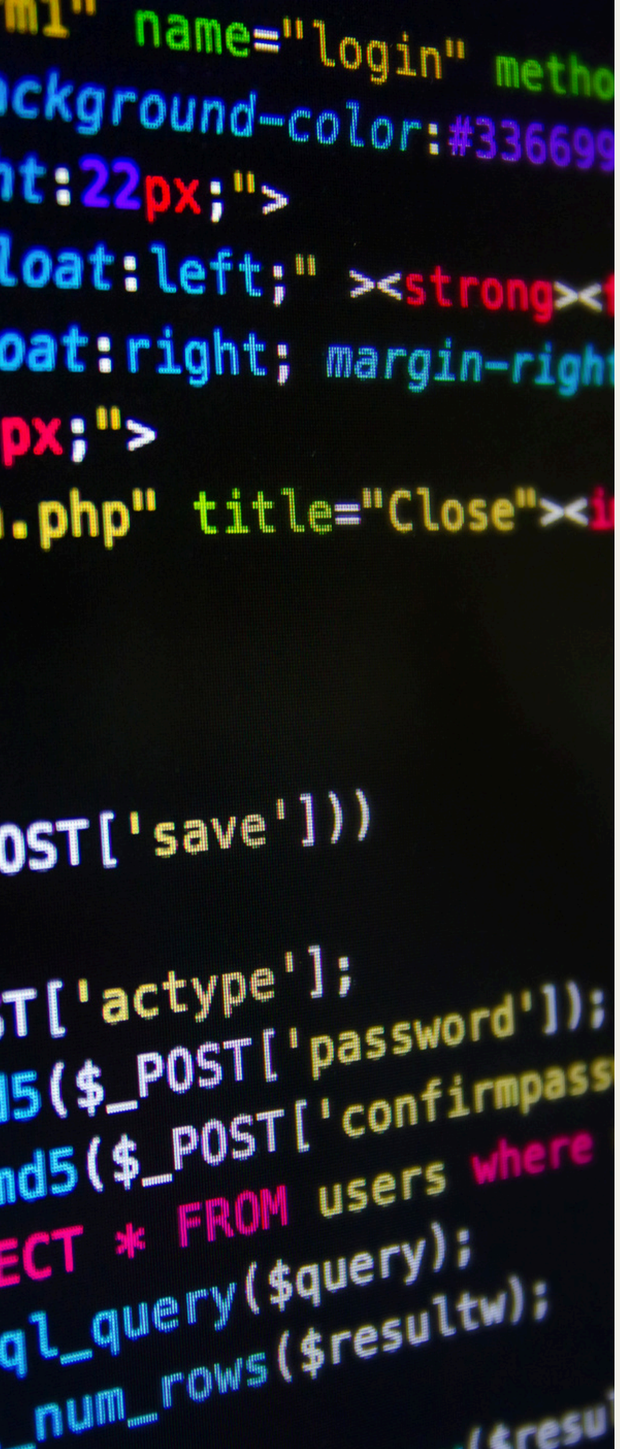
2. A validator cannot precommit twice at the same height

3. Governance cannot modify finalized state

4. Emergency actions cannot bypass consensus

5. RPC cannot expose unfinalized truth

Violating any invariant is a consensus bug.



FINAL CODE FREEZE V1.0 (DEVELOPER)

At this point:

Consensus rules are frozen

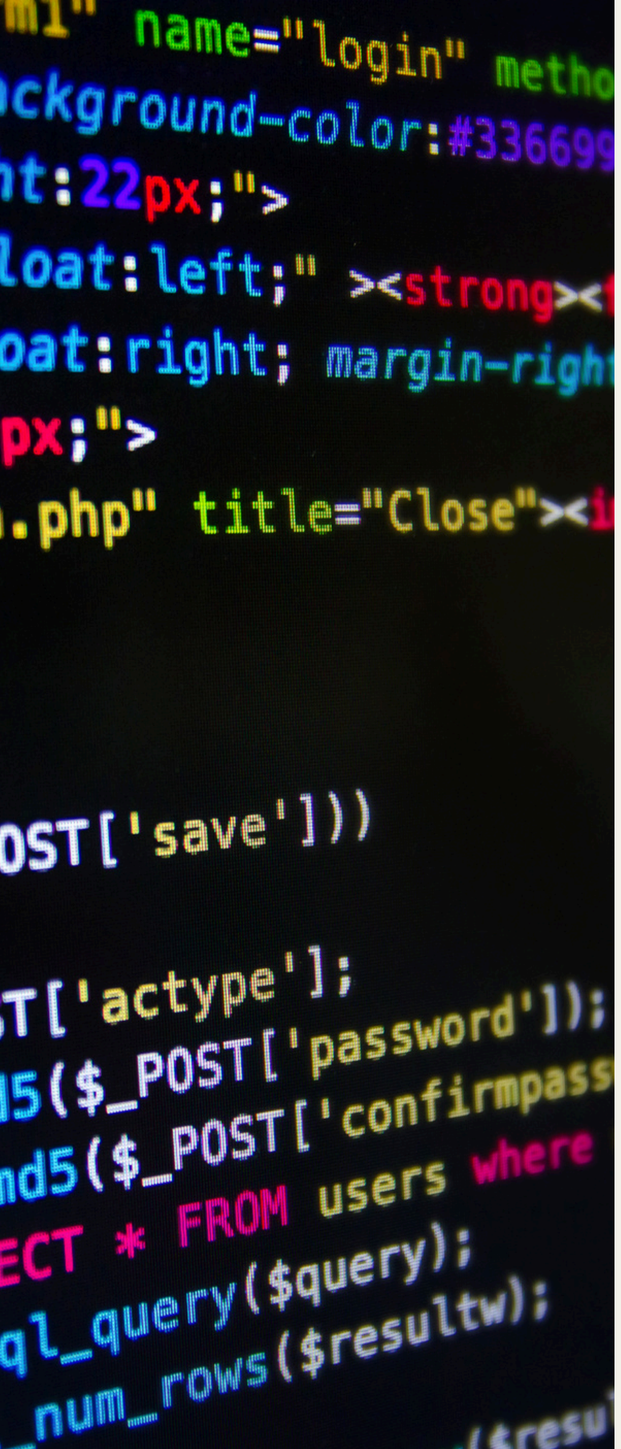
CommitProof v1 encoding is frozen

System contracts v1 are frozen

RPC semantics are frozen

Harness scenarios are frozen

No breaking change is allowed without:
Protocol version bump
Full re-audit



AUDIT KICKOFF MARKER (DEVELOPER)

This document marks the official audit baseline.

Auditors receive:

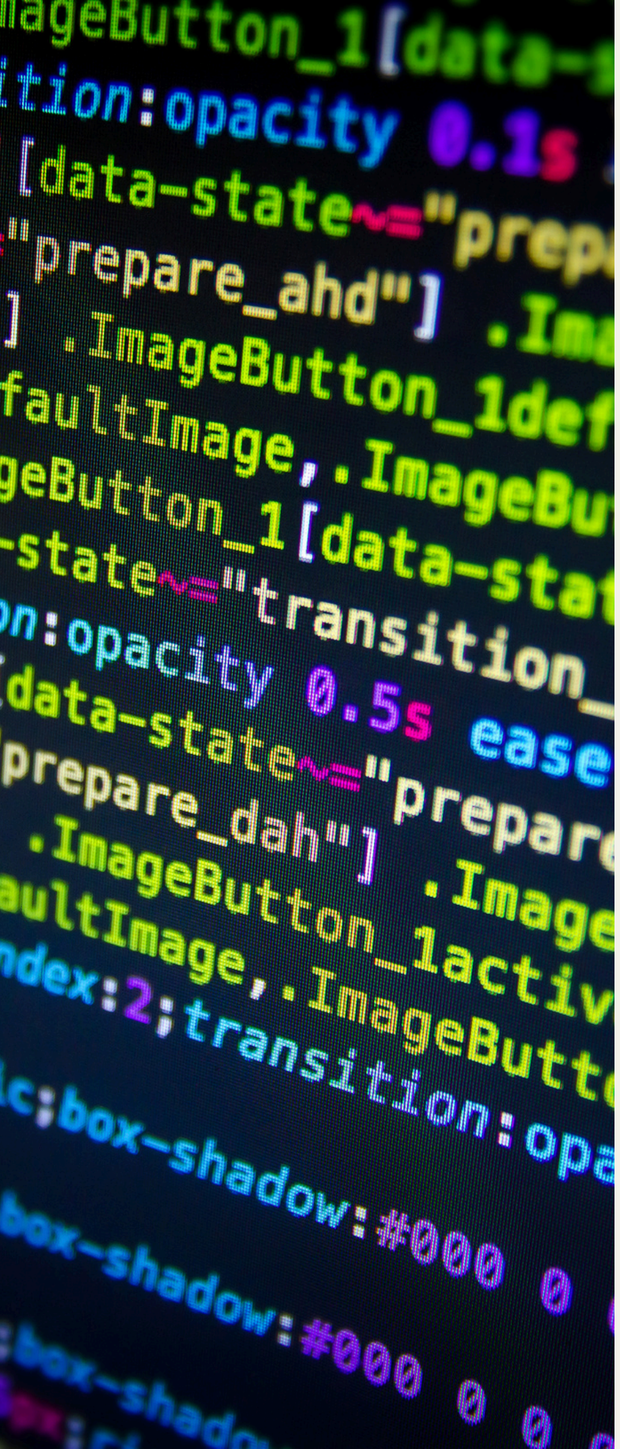
This DPS document

The MPF document

Reference client commit hash

Harness configuration

Anything outside these inputs is out of scope.



FINAL DEVELOPER STATEMENT

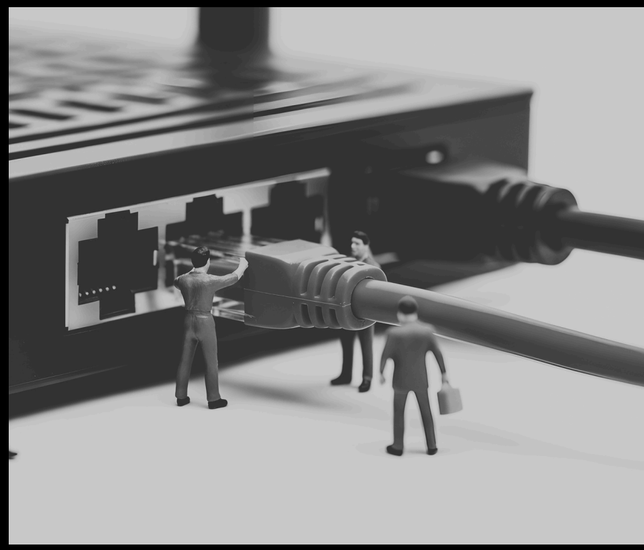
➢ Any client, node, or service claiming SafeOneChain compatibility must conform exactly to this specification. Deviations are detectable, auditable, and non-compliant by definition.

END OF DEVELOPER PROTOCOL SPECIFICATION (DPS-V1.0)

X

Medium

CONTACT US



Telegram

safeonechain.com