



Distributed Compliance Ledger (DCL)

Agenda



- About the Project
- Use Cases
- Network Topology: Nodes
- Network Topology: Clients
- Usage Scenarios
- Authentication and authorization
- About Tendermint

About The Project



Two groups of use cases:

- Device Models (created by Manufacturers) and Compliance tests (created by Testers and Alliances)
- Public Key Infrastructure

Makes device attestation and certification process more

- Simple
- Automated
- Secure
- Transparent and trusted

About The Project



- Public Permissioned Distributed Ledger owned and hosted by CHIP and ZB Alliance members
 - Write access to the Ledger is permissioned and restricted
 - Anyone can read from the Ledger

About The Project



- <https://github.com/zigbee-alliance/distributed-compliance-ledger>
- Open source (Apache 2.0)
- Core logic is implemented on top of [Tendermint](#) and Cosmos SDK

Use Cases: Write Data to the Ledger



- **Use Case 1: Writing Device Model Info by Manufacturer**

As a Manufacturer, I need to write information about a Device Model, Firmware and Hardware to the Ledger, so that it can be read by anyone from the ledger for compliance and other purposes

- **Use Case 2: Writing Compliance Tests results by Testing Organization**

As a Compliance Tester, I need to write result of compliance tests for each tested Device Model to the Ledger, so that it can be read by anyone from the ledger for confirming compliance test results and other purposes

- **Use Case 3: Writing Compliance Confirmation by ZB**

As an Alliance (ZB Alliance for example), I need to write confirmation of the Compliance Test Results to the Ledger, so that it can be read by anyone from the ledger for compliance checks (for example, when adding a new device to the Network)

- **Use Case 4: PKI**

As a Certification Authority, I need to be able to publish certificates (intermediate or leaf) to the ledger, so that it can be read by anyone

Use Cases: Read Data from the Ledger



- **Use Case 5: Check for Device Compliance**

As a ZB (or any other) Network, I need to know if the Device is compliant with the ZB (or other) standard by reading information from the Ledger, so that I can add it to the Network

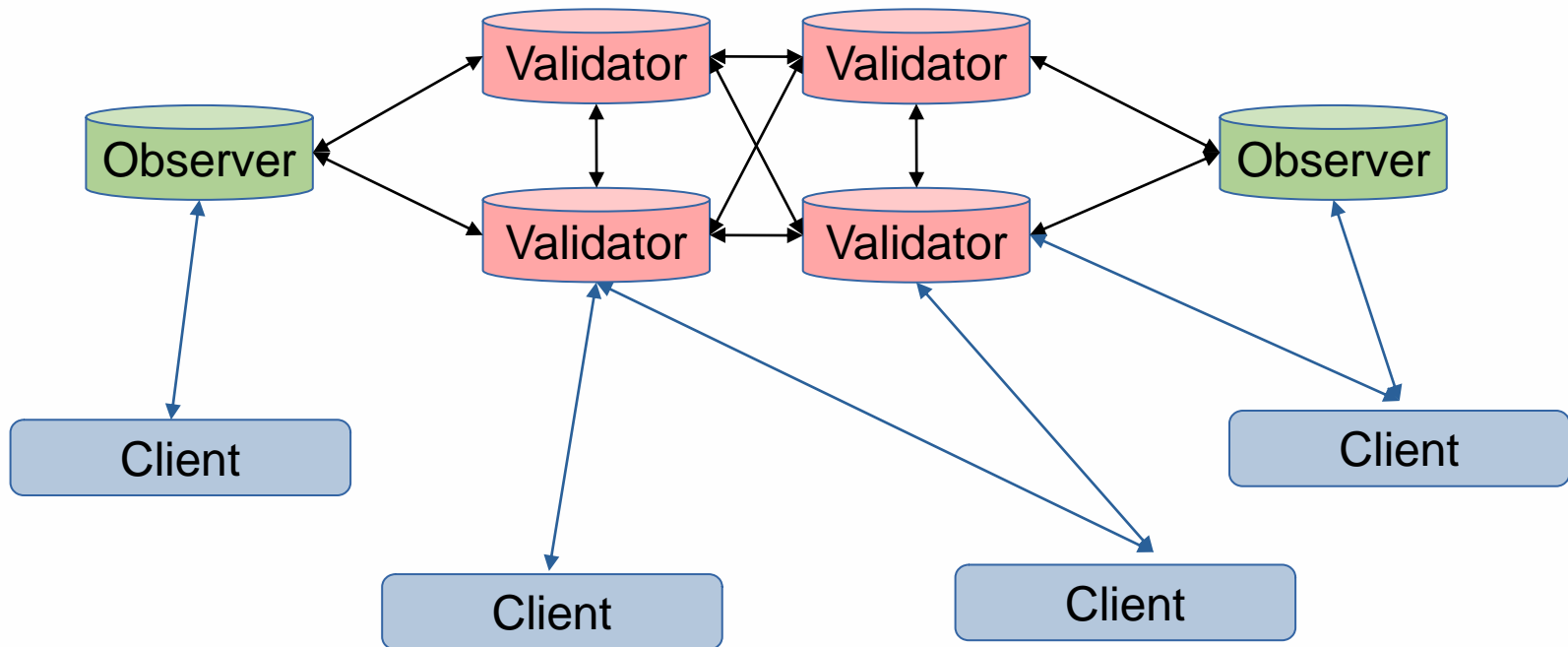
- **Use Case 6: Reading Device Model Info**

As a ZB (or any other) Network, I need to know Device's Model Info including Firmware and Hardware versions by reading information from the Ledger

- **Use Case 7: PKI**

As a user, I need to get a chain of X509 certificates to verify the authority of the given X509 certificate.

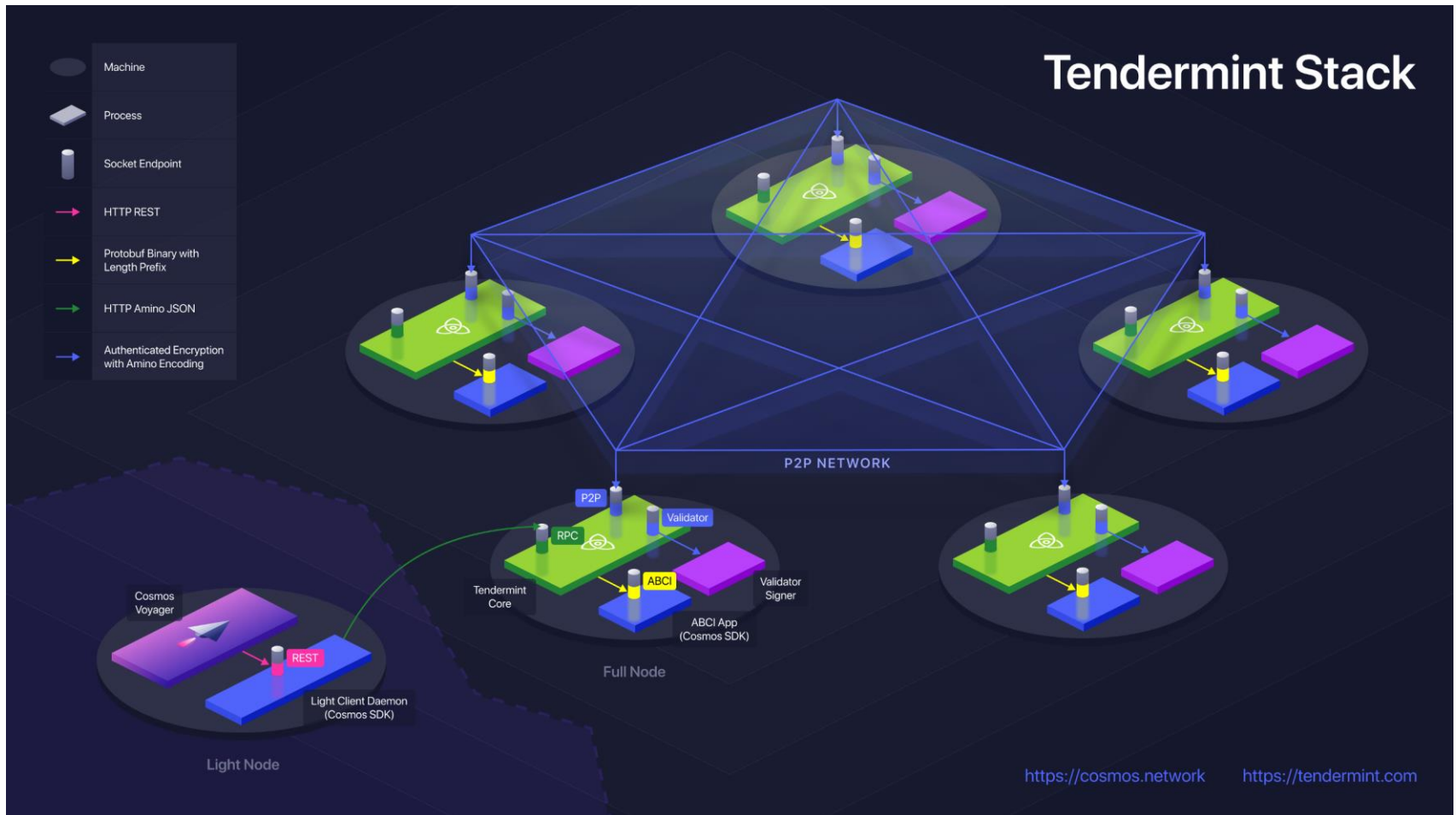
Network topology



Full node

Light client

Network topology (different view)



Network topology: Node (Replica) types



■ Validator Nodes

- Have a full copy (replication) of the ledger
- Permissioned, not anyone can be a Validator Node
- Total number of nodes should be limited
- Initial nodes – genesis; number of nodes can be extended

■ Observer Nodes

- Have a full copy (replication) of the ledger
- Anyone can have an Observer node

Network topology: Clients



- Anyone writing or reading
- Client != Node
- Don't need to maintain or have a full Ledger
- Number of clients is not limited
- Can read from 1 Node only
- Can connect to a node/replica he doesn't fully know and trust, as there are crypto proofs for the replies
- Write access to the Ledger is restricted (write permission needs to be granted)
- Read access to the Ledger is public (anyone can read)

Client Types



- CLI
 - Run on a user's machine
 - Connected to one of the nodes (either a Validator or Observer)
- REST API
 - Backend server: either a local deployment per organization/application/user, or a global
 - The backend server is connected to one of the nodes (either a Validator or Observer).
- Client Libs / API Used by the application itself
 - Core API is there, but language and platform-specific libraries - TBD

Usage Scenarios



1. The user's organization doesn't have any nodes/replicas in the network.
 - 1A: CLI connected to one of the nodes in the network.
 - 1B: REST API against a backend deployed at the user's organization (or an organization the user trusts). The backend is connected to one of the nodes in the network.
 - 1C: The user's application calls the API lib connected to one of the nodes in the network
2. The user's organization runs a validator node.
 - 2A: CLI connected to the validator node.
 - 2B: REST API with a backend server connected to the validator node.
 - 2C: The user's application calls the API lib connected to the validator node.
3. The user's organization runs an observer node(s).
 - 3A: CLI connected to the observer node.
 - 3B: REST API with a backend server connected to the observer node.
 - 3C: The user's application calls the API lib connected to the observer node.

Authentication



- Every transaction (write request) must be signed
- No signatures/authentications for read requests
- Sender must have an Account on the Ledger (as a transaction)
- The public key used for signature verification must be on the Ledger (associated with the Account transaction)

Authorization: User Roles



- Trustee
 - create new user accounts
 - assign roles to the account
 - revoke roles from the account
 - approve X509 root certificates
- Node Admin
 - add a new Validator node

Authorization: User Roles



- Vendor
 - publish device model info
- Test House
 - publish compliance test results
- ZB Certification Center
 - certify or revoke certification of device models

About Tendermint



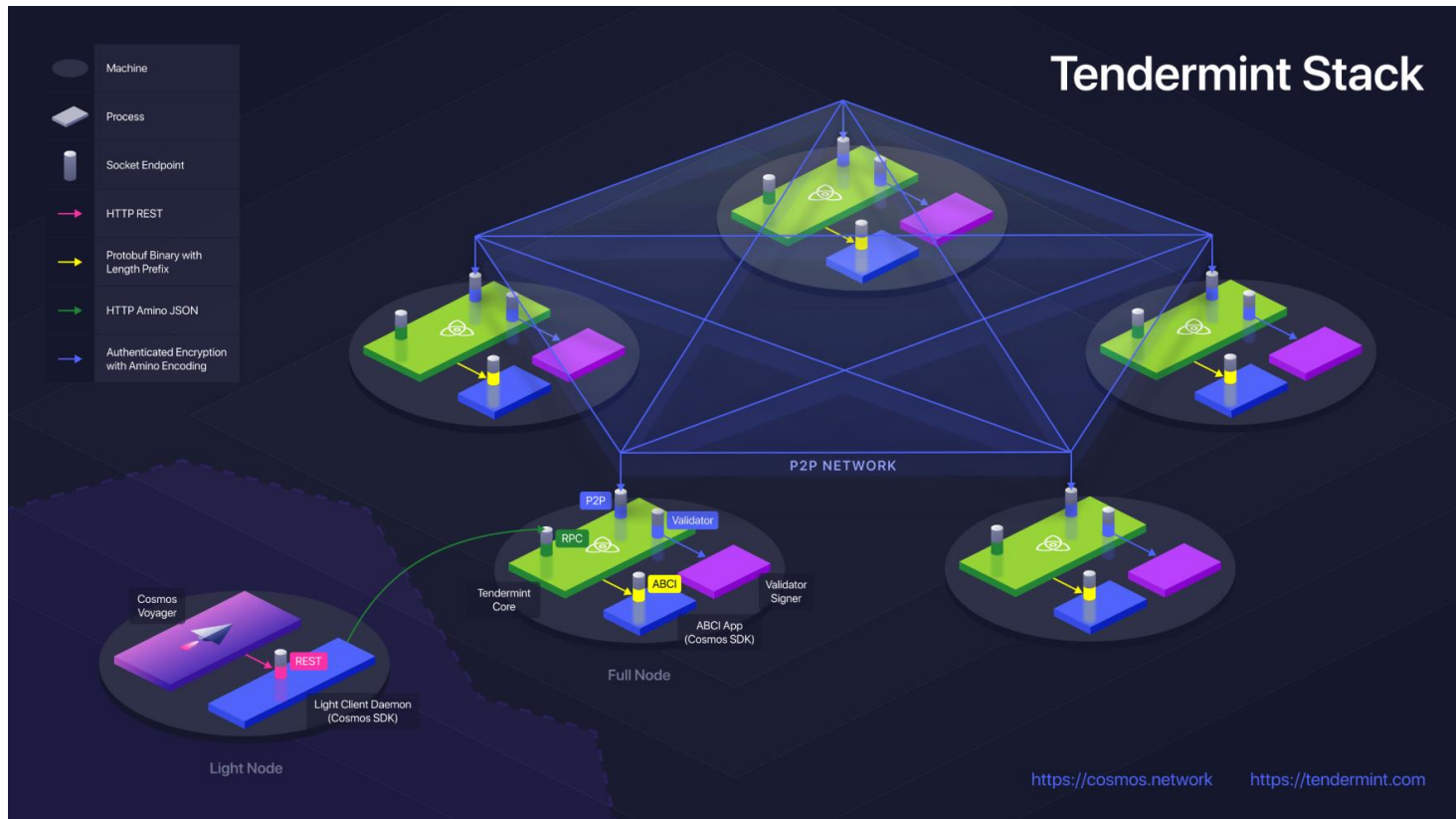
- [Tendermint](#) is a blockchain application platform performing Byzantine Fault Tolerant (BFT) State Machine Replication (SMR) for arbitrary deterministic, finite state machines.
- It provides the equivalent of a web-server, database, and supporting libraries for blockchain applications written in any programming language. Like a web-server serving web applications, Tendermint serves blockchain applications.

About Tendermint



- <https://github.com/tendermint/tendermint/>
- Apache 2.0
- Big community
- 3.5K stars on GitHub
- Projects in Production

About Tendermint



About Tendermint: BFT



- Tendermint works even if up to $1/3$ of machines fail in arbitrary ways (either fail-stop/crash, or behave maliciously)
 - 1 Node in the network of 4 nodes may crash, and the network will still be working
 - 1 Node in the network of 4 nodes may behave maliciously (for example, ignore authentication rules), and the network will still be correctly working (for example, enforce authentication rules)

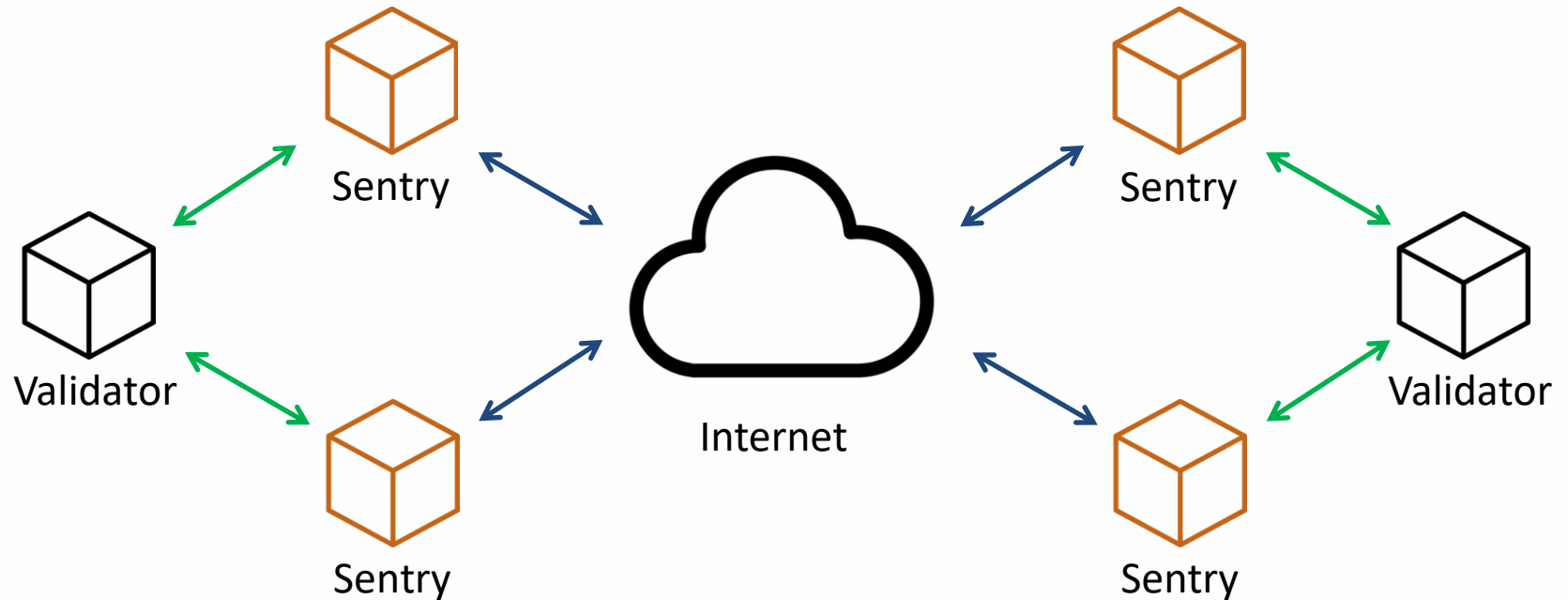
About Tendermint: Light Clients




- Light client can read state from any **single** node without downloading full transaction log and trust the result
- Don't need to have/maintain a full Ledger on the client side
- This is achieved using special data structure for state storage
 - Allows proving that given data is present or absent in the state
 - IAVL in Tendermint
 - Patricia Merkle Tree in Ethereum

About Tendermint: Sentry Nodes

DDoS mitigation strategy



Private connection 
Public connection 