DSR

DOING SOFTWARE RIGHT

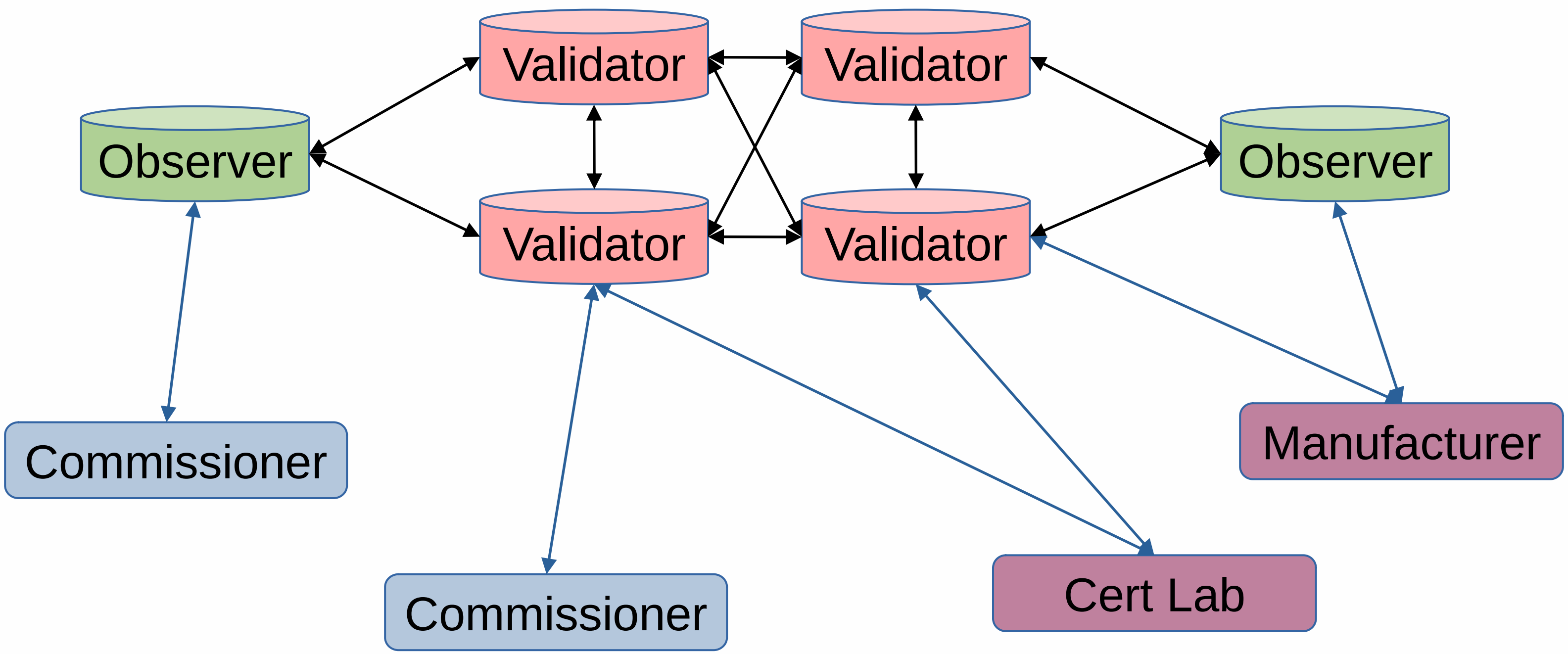
ZB Ledger

Architecture Overview

Agenda

- Arch overview
- Data structures
- Writing to ledger
- Reading from ledger
- Dealing with correlation

Arch overview



Legend: Full node Light client

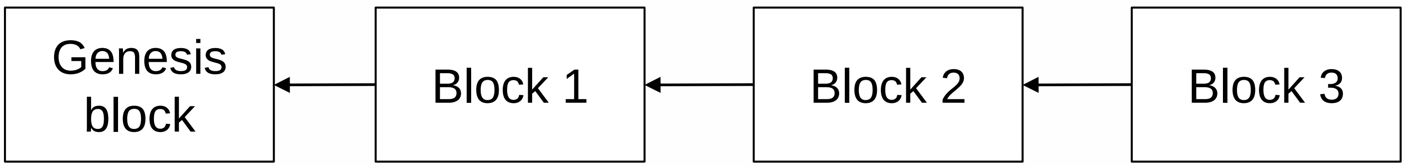
Full nodes

- Full nodes (both observer and validator)
  - Store full transaction log and state
  - Execute transactions from new blocks
  - Have private/public keypair (and announce their public keys to other connected nodes)
  - Forward transactions and consensus messages from other full nodes (gossip)
  - Process light client requests
- Validator nodes
  - Have voting power in block creation process (agreeing upon global order of incoming transactions)
  - Broadcast consensus messages
  - Distinguished from observer nodes by having their public key stored In ledger
  - Public keys of initial validator nodes are stored in genesis block (which is the first block in a blockchain)

Light clients

- Store public keys of current validator set
  - Pre-shared genesis block provide initial keys
  - Actual key set can be safely downloaded from ledger since all updates are signed by previous validator sets
- Can send signed write transactions to full nodes
  - Transaction are then gossiped through network and finally included into signed blocks
  - Authorization is enforced by validator nodes which create and sign blocks
- Can send read requests to full nodes
  - Reply can be trusted because it basically includes signed state proof
  - State proof ensures that given data exists in some version (designated by state hash) of ledger state
  - Signatures of block containing that state hash ensure that this state is really present on the ledger

Ledger data



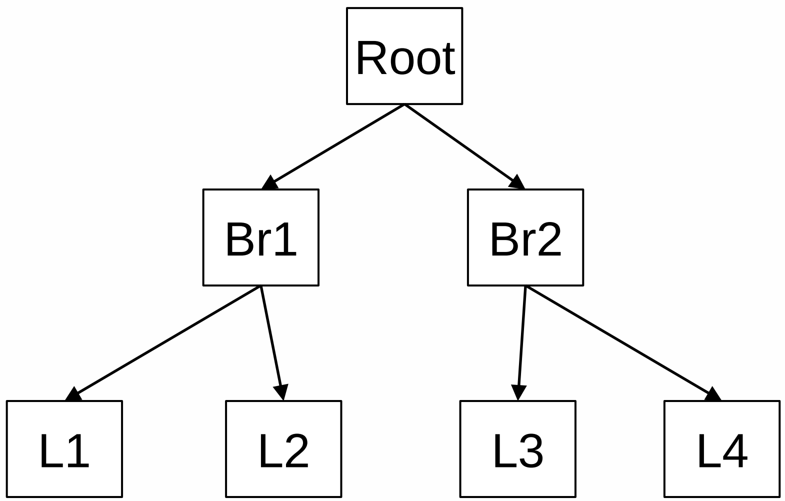
Block contains (simplified):

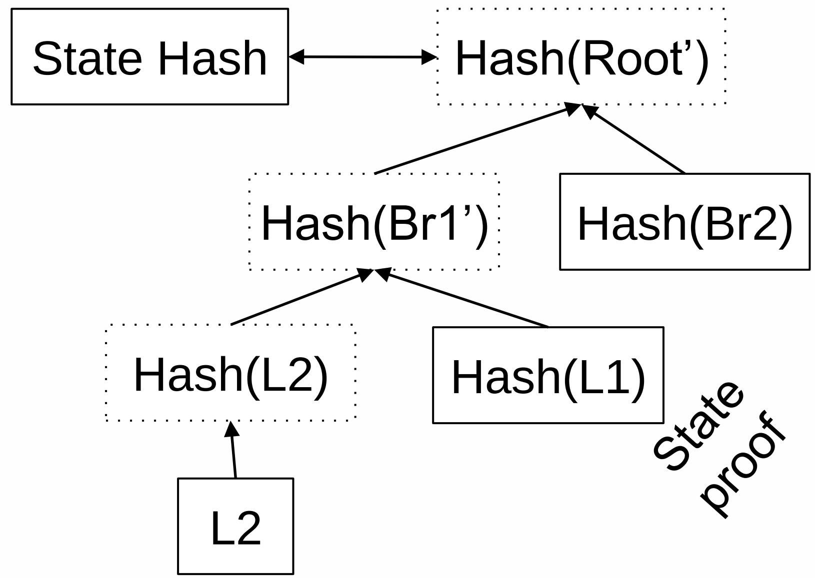
- Block number
- Block time
- Hash of previous block
- Transaction list
- Current and next validator sets
- State hash
- Signatures of above data from at least 2/3 of current validator set

Block is valid if it has at least 2/3 correct signatures of validator set agreed upon in a previous block

Unlike PoW no mining is needed, and no forks are possible (as long as supermajority of validators are honest)

State and state proofs





Simplified idea:

- Just a tree with data in leaf nodes
- Use hashes of nodes contents instead of pointers
- State hash is a hash of root node

State proof of L2 being part of state with given state hash is [Hash(L1), Hash(Br2)]:

- Calculate Hash(L2)
- Br1' = [Hash(L1), Hash(L2)]
- Root' = [Hash(Br1'), Hash(Br2)]
- Compare Hash(Root') with state hash from block

Mapping ledger to X509 mindset

- Preshared genesis block with validator keys
  ~ preshared root certificates
  - Updates can be securely transferred to client
- Accounts public keys on ledger
  ~ intermediate certificates
  - Validity is guaranteed by validator nodes keys
- State proofs with block signatures
  ~ signed messages

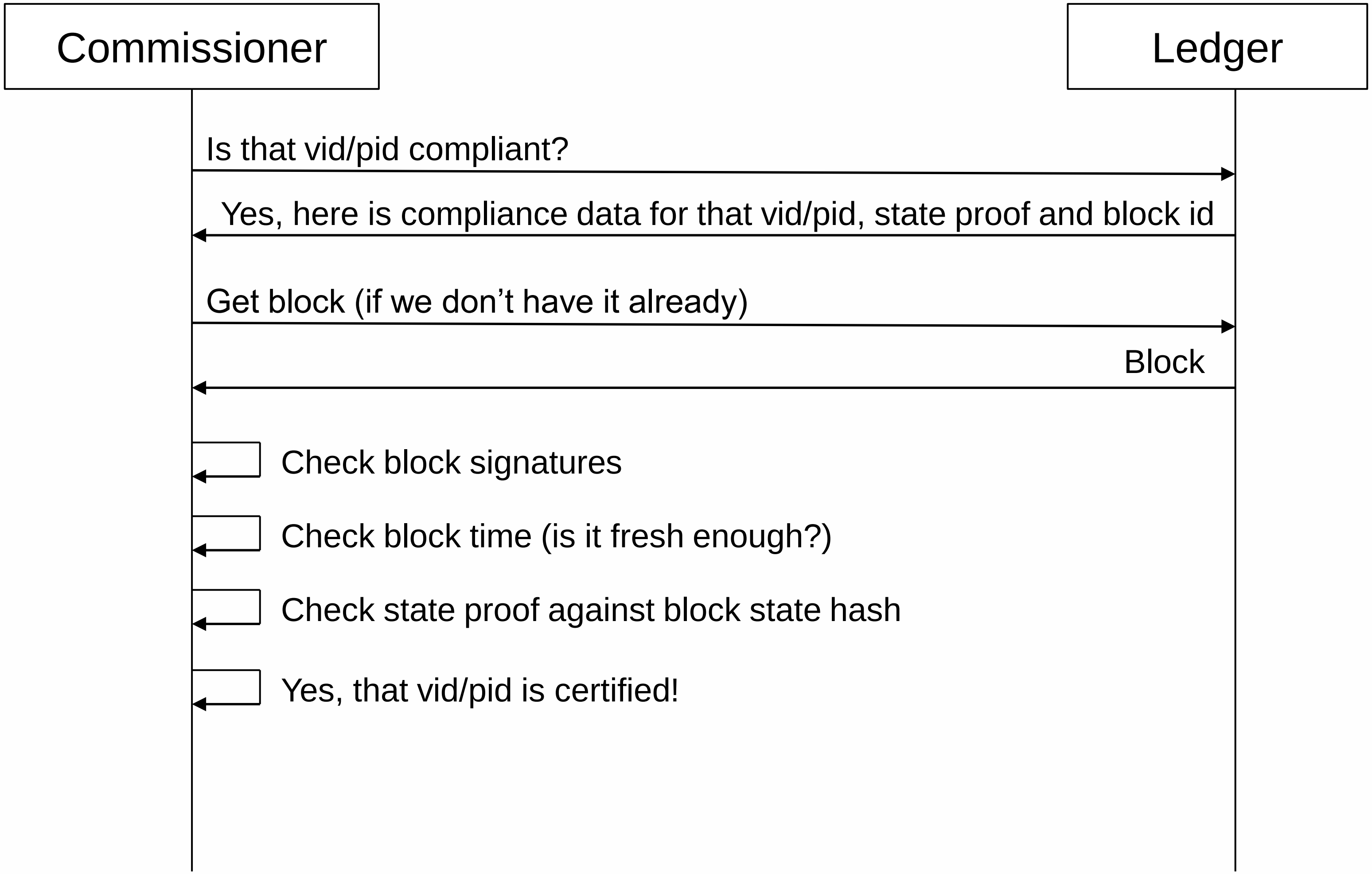
Writing data to ledger

- Signed transaction is sent to any full node
- Transaction is gossiped through network
- If valid it is included into block
  - Validity is checked by all validator nodes
  - If okay transaction is executed, which modifies state and possibly next validator set
  - Block containing transactions, state hash and next validator set is signed by supermajority of validator nodes
- As long as supermajority of nodes are honest it is not possible to get any invalid transaction into correctly signed block

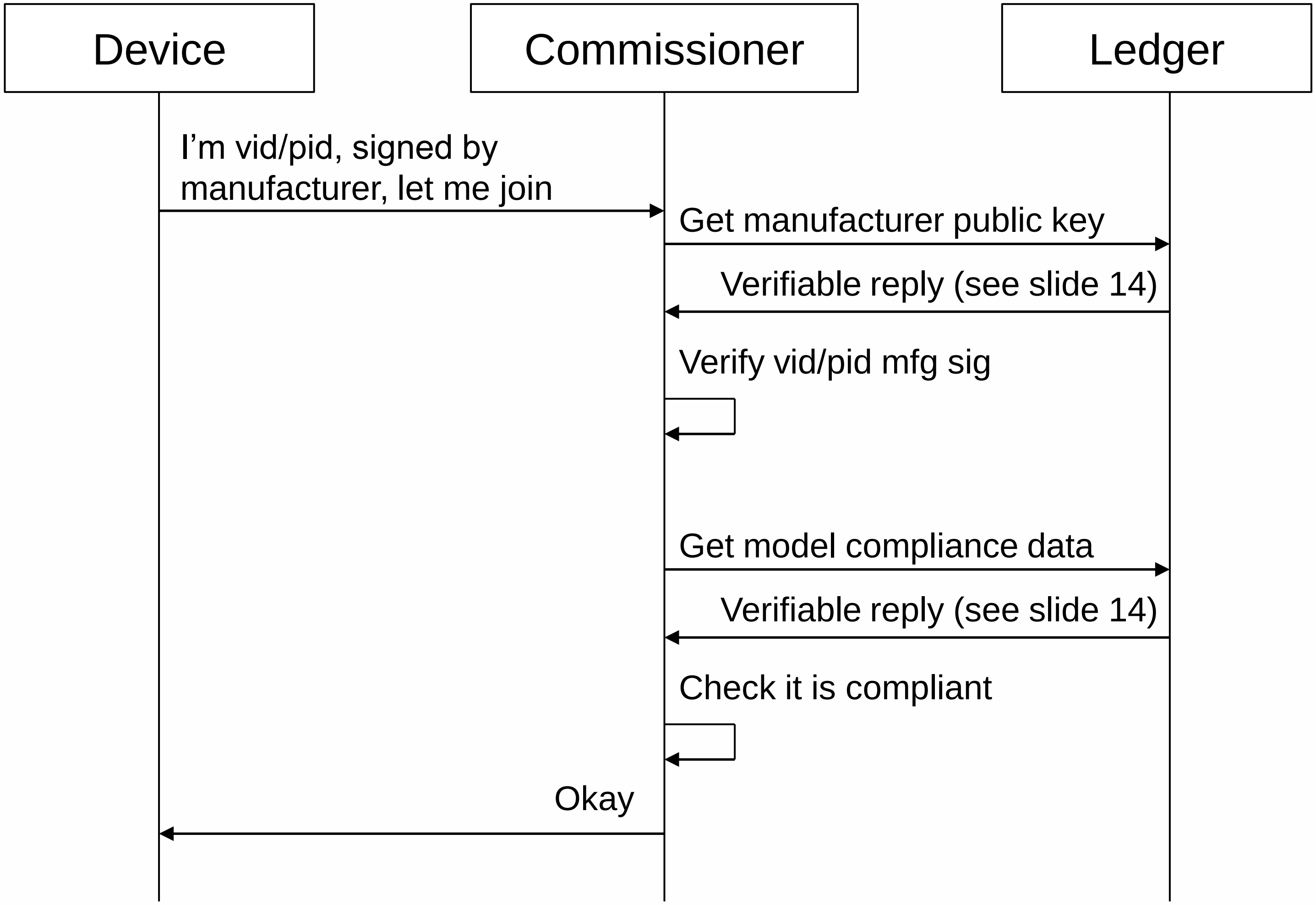
Reading data from ledger

- Read request is sent to any full node
- Node replies with data, state proof and corresponding block number
- If we don't have a corresponding block we request it
- Block contains time, state hash, current validator set and signatures from supermajority of that validator set
  - So we can check state proof, whether reply is stale and validity of block itself, if validator set is same as stored on the client
- If current validator set differs from what is stored on client then it is possible using binary search to quickly request and find all blocks that changed validator set, thus building a chain of trust (each previous validator set signs new one)
- The only preshared/cached data required for validating replies are validator node signatures (Ed25519, 32 bytes per node)

Reading data from ledger



Example: Device authentication



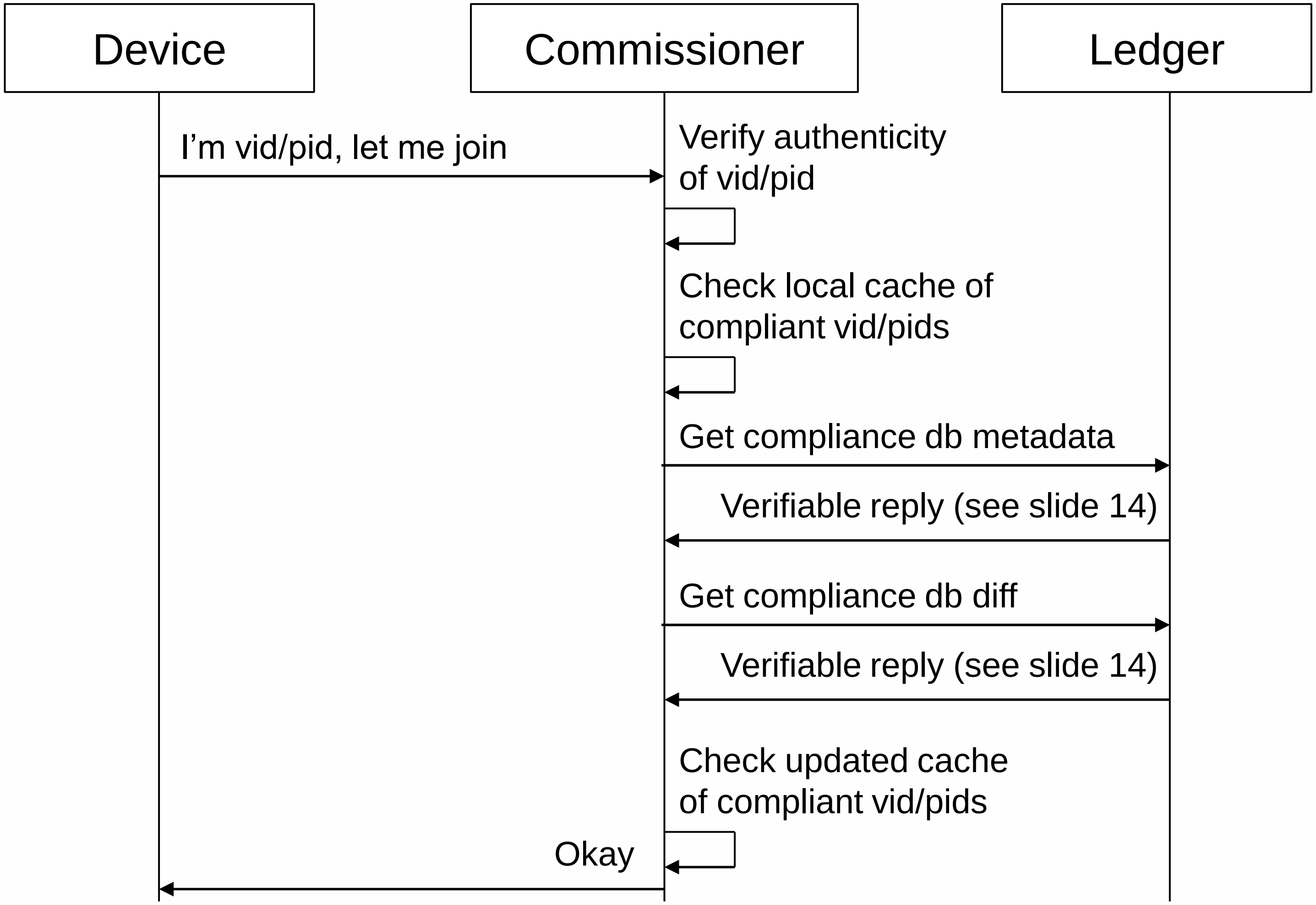
Correlation problem

- When commissioners send requests about certification of individual vid/pid ledger nodes can conclude that somebody behind that IP has device, which is a data leak
- Solution boils down to
  - either requesting much more information than actually needed…
  - ...or moving more information off-ledger

Option 1: Download state

- Idea: request not just one vid/pid, but all avaliable
- Pro: Correlation becomes much harder or even impossible (node doesn't know which vid/pid we are interested in)
- Con: Clients need to store much more data, somewhat undermining light client architecture advantage
- Con: Depending on suboption can require significant development effort
- For 500 vendors each having 500 devices (250000 vid/pid combinations):
  - If only boolean state is needed that should fit into ~ 1-2 Mb (just store list of certified IDs and put a merkle tree on top to allow safe incremental updates)
  - If additional data is needed (like vendor and product names, as well as id of test facility which certified device) this can increase probably to ~ 50 Mb (given that data can fit into 200 bytes per vid/pid pair)
  - There are lots of data layout and compression options for middle ground solutions

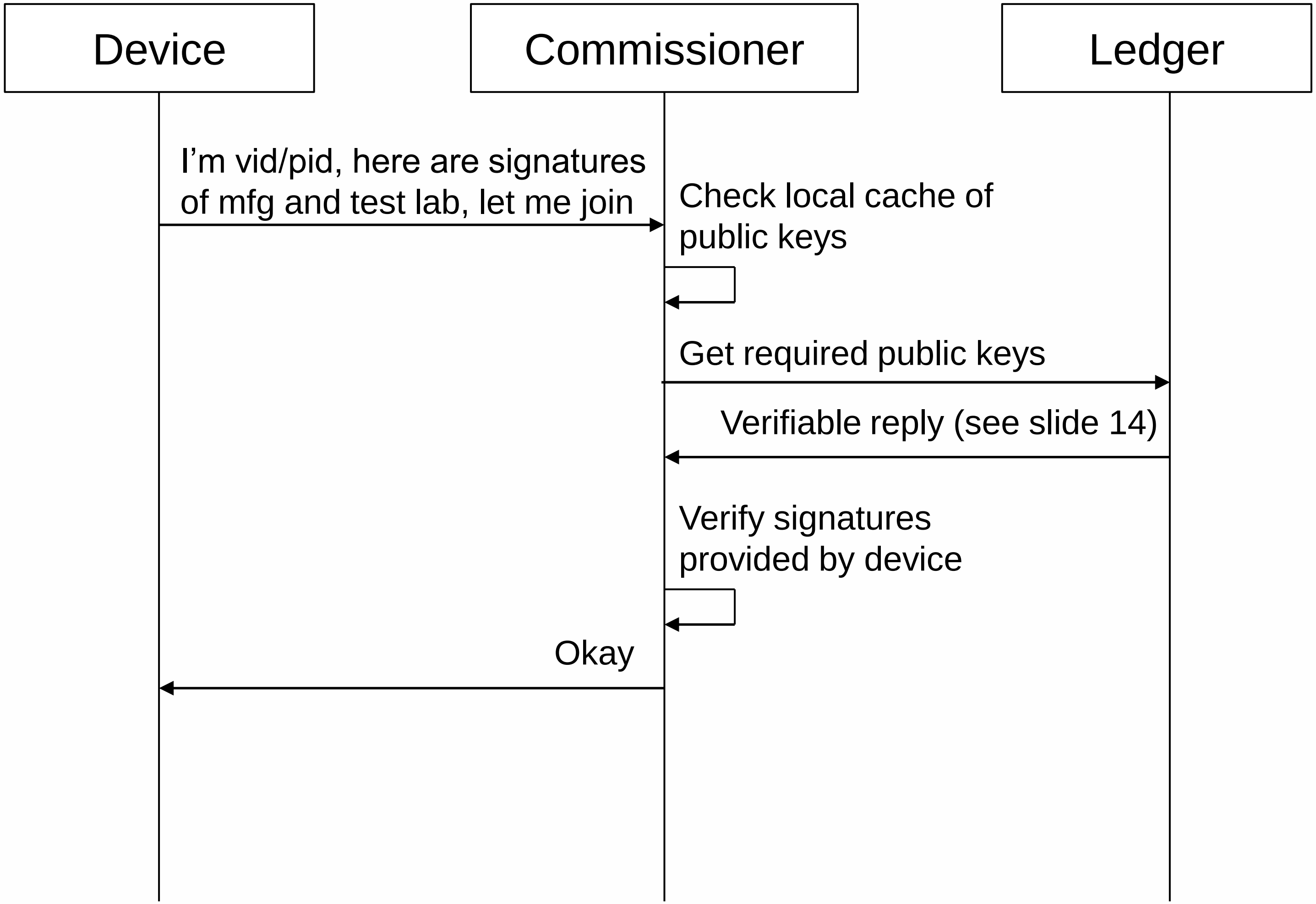
Device verification (option 1)



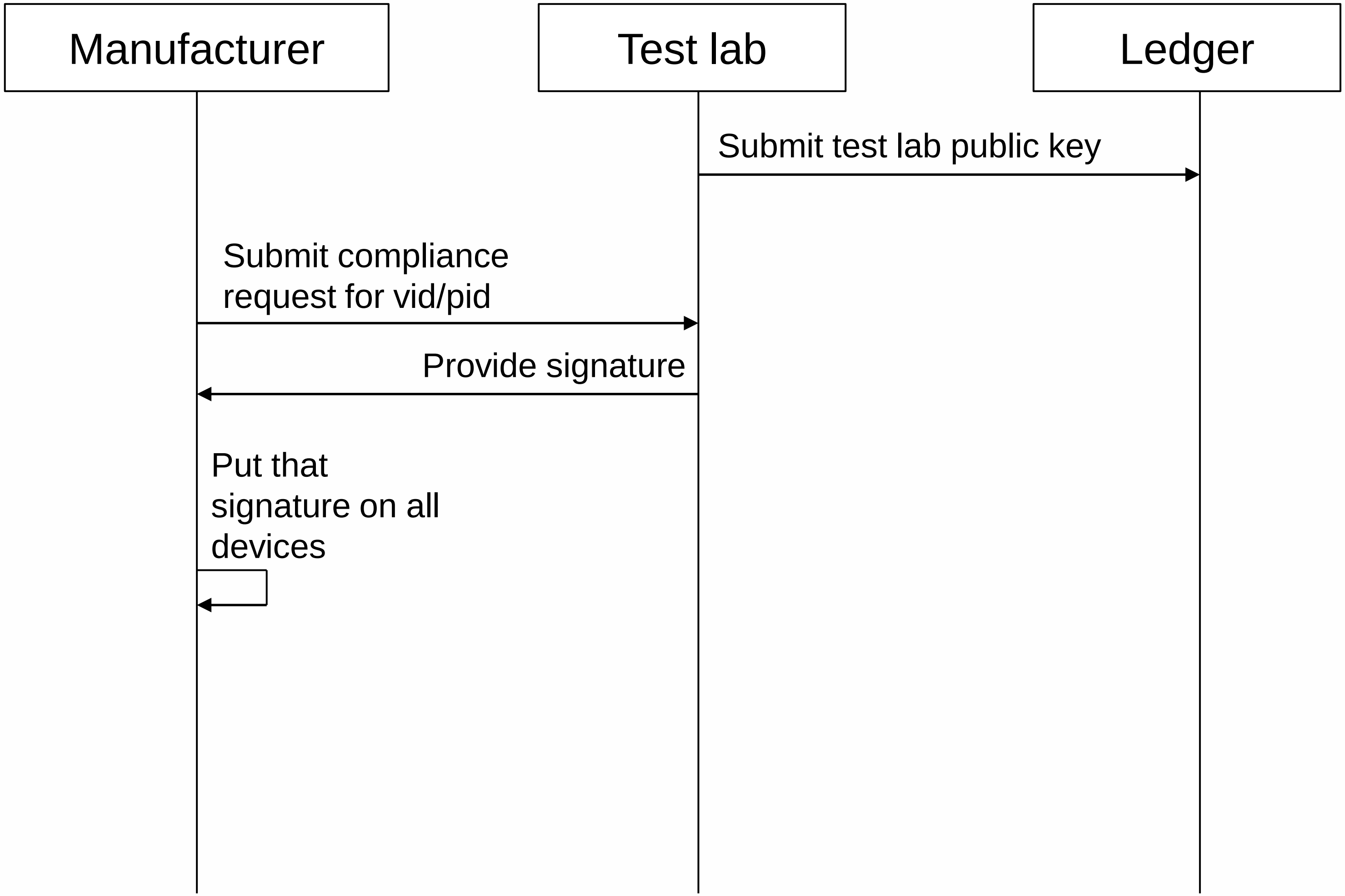
Option 2: Use certificates

- Idea: store only public keys and revocation data of test facilities on ledger, put signatures on devices
- Pro: amount of data to download from ledger becomes much less
- Pro: data about concrete vid/pids doesn't leak
- Con: certification after release require firmware updates to put new signatures
- Con: less flexibility

Device verification (option 2)



Device certification (option 2)



DSR
DOING SOFTWARE RIGHT

Thank you!