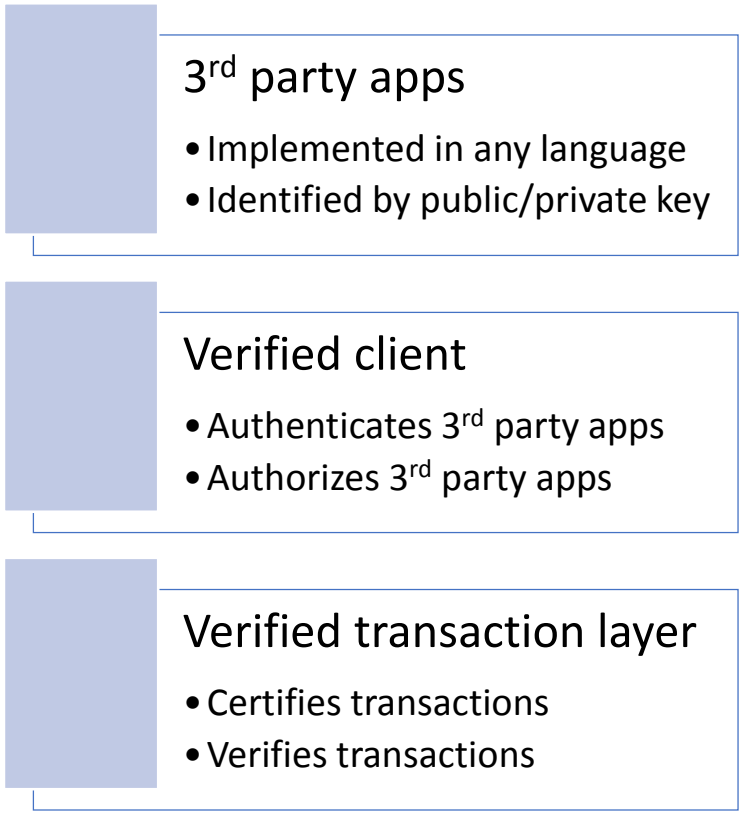
Verified Blockchain Protocol

Design specifications

What is the Verified blockchain used for ?

- Verified blockchain is not an application like Bitcoin
- Verified does not provide a smart contract engine like Ethereum
- Verified does not store a global application state like Ethereum
- Multiple applications can run on the Verified blockchain which provides a thin consensus layer.



Authenticating and Authorizing clients

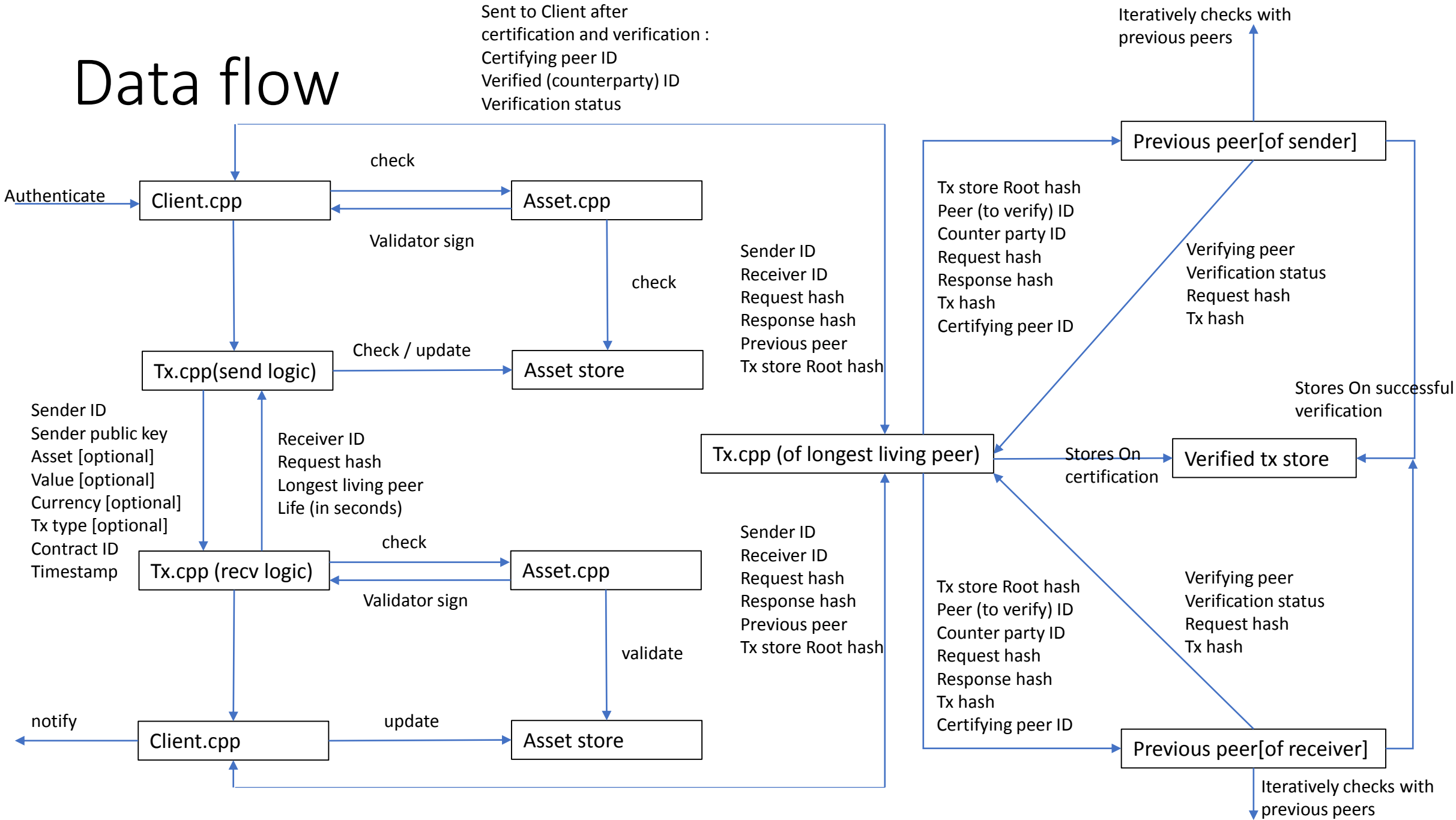
AUTHENTICATION

- Self generated public/private key pair by Verified peer
- Key generated using Verified peer's Infohash (peer key)
- Public key and Private key encrypted with hash function using seed (which is combination of Infohash+password)
- Encrypted Public Key and Private Key (as values) stored on opendht using put operation for key (=seed)
- Public and Private Key are retrieved using seed when app/user re-logs in. If seed is correct, then encrypted private/public key can be correctly retrieved and decrypted.

AUTHORIZATION

- For each transaction, a small amount denominated in Via (stored in the 'Asset' datastructure on the Verified blockchain) is charged.
- Authorization of any transaction requires two checks
  - boolean checkBalance(InfoHash)
  - boolean canDebit(InfoHash)
- On successful check, Via is deducted and held in suspense till transaction is successfully certified and verified.

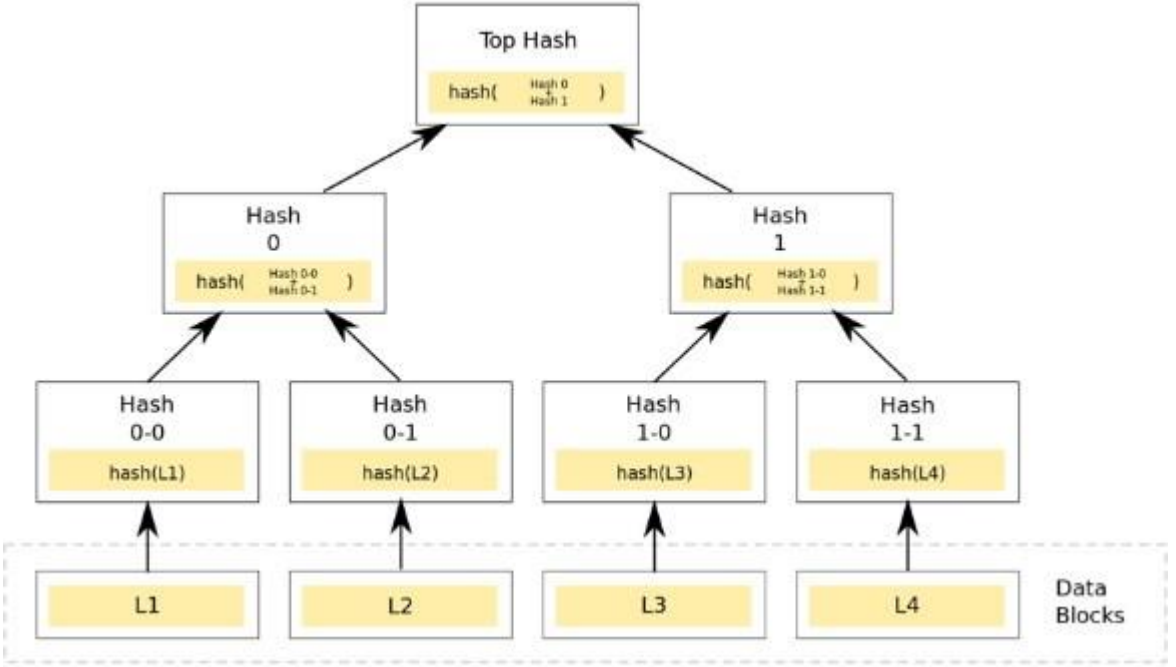
Data flow


Sent to Client after certification and verification :
Certifying peer ID
Verified (counterparty) ID
Verification status
Iteratively checks with previous peers
Previous peer[of sender]
Authenticate
Client.cpp
check
Asset.cpp
Validator sign
check
Tx store Root hash
Peer (to verify) ID
Counter party ID
Request hash
Response hash
Tx hash
Certifying peer ID
Verifying peer
Verification status
Request hash
Tx hash
Sender ID
Receiver ID
Request hash
Response hash
Previous peer
Tx store Root hash
Tx.cpp(send logic)
Check / update
Asset store
Stores On successful verification
Sender ID
Sender public key
Asset [optional]
Value [optional]
Currency [optional]
Tx type [optional]
Contract ID
Timestamp
Receiver ID
Request hash
Longest living peer
Life (in seconds)
Tx.cpp (of longest living peer)
Stores On certification
Verified tx store
Tx.cpp (recv logic)
check
Asset.cpp
Validator sign
validate
Sender ID
Receiver ID
Request hash
Response hash
Previous peer
Tx store Root hash
Tx store Root hash
Peer (to verify) ID
Counter party ID
Request hash
Response hash
Tx hash
Certifying peer ID
Verifying peer
Verification status
Request hash
Tx hash
notify
Client.cpp
update
Asset store
Previous peer[of receiver]
Iteratively checks with previous peers

Asset store and the Patricia Merkle trie

- Asset store attributes
  - Asset ID
  - Asset value
  - Asset balance
  - Currency symbol
  - Transaction type [debit, credit, debit block, etc]
  - Contract ID
  - Request hash
  - Response hash
  - Tx hash (status)

- Asset and Transaction data on the Verified blockchain are stored in Patricia Merkle tries.

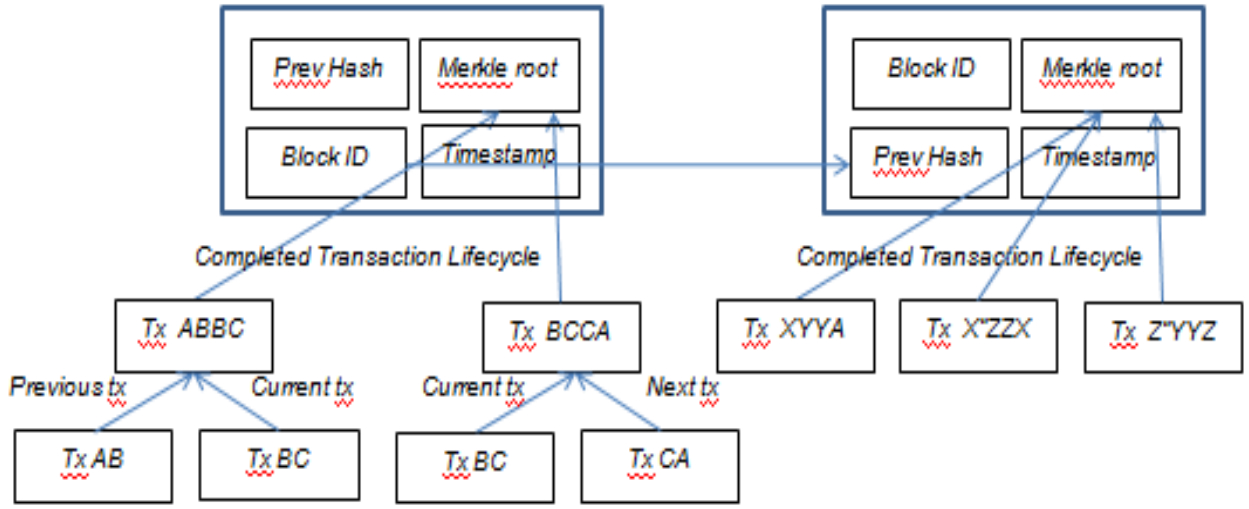


Transaction stores

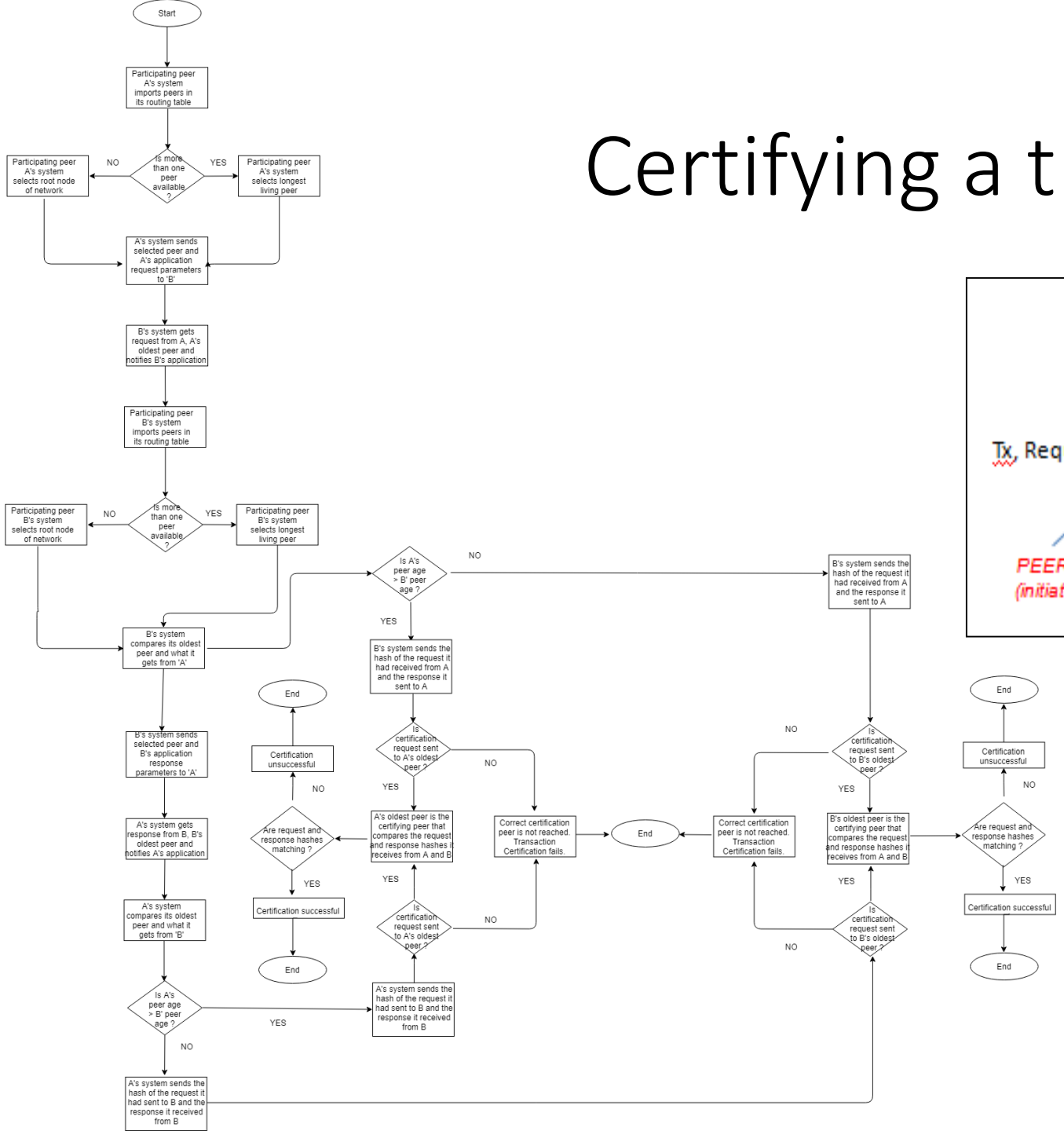
- All transactions are clubbed into blocks that are chained [hence, the term blockchain]

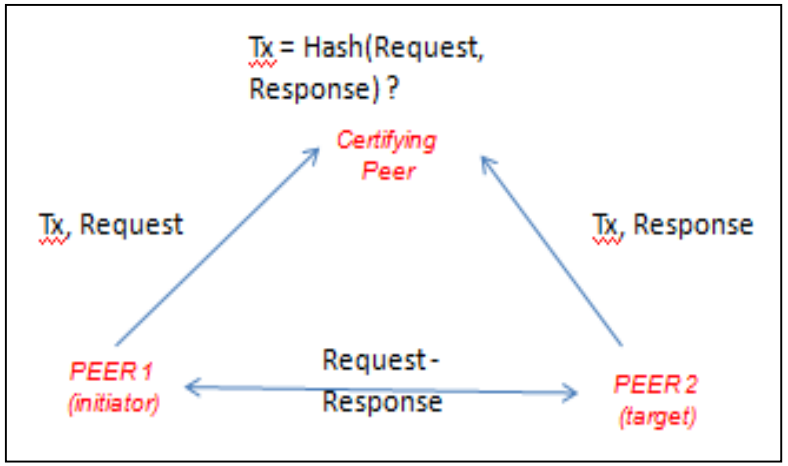
- Verified transaction store [used by certifying peers]
  - Sender ID
  - Counter party (receiver) ID
  - Request hash
  - Response hash
  - Previous peer
  - Next peer
  - Status
  - Tx hash
  - Block ID
  - Timestamp

- Transaction store [used by each peer to store its own transactions]
  - Transaction hash
  - Transaction type
  - Block identifier
  - Certifying peer ID
  - Counter party (receiver) ID
  - Sender ID
  - Timestamp



Certifying a transaction

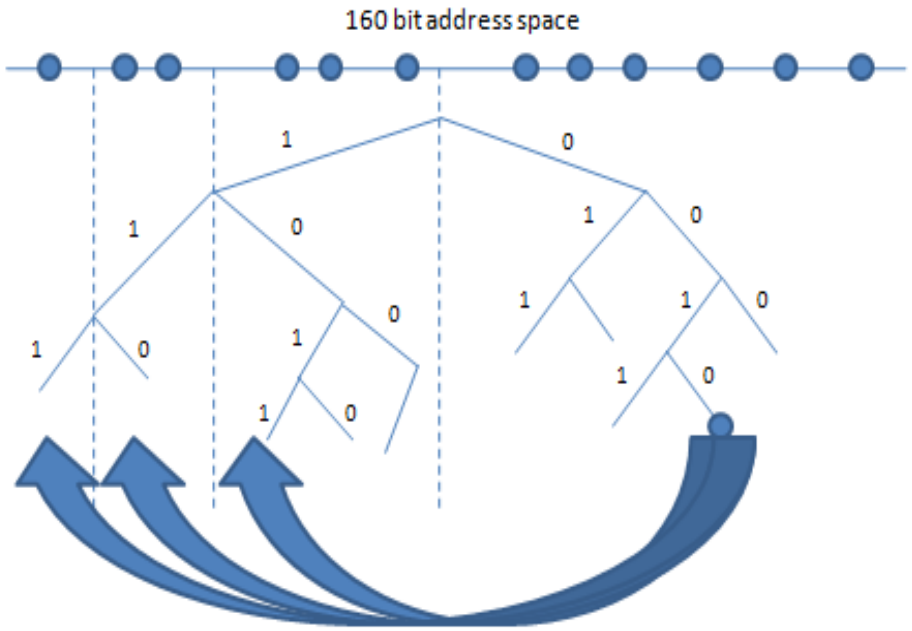




Peer discovery and Communications

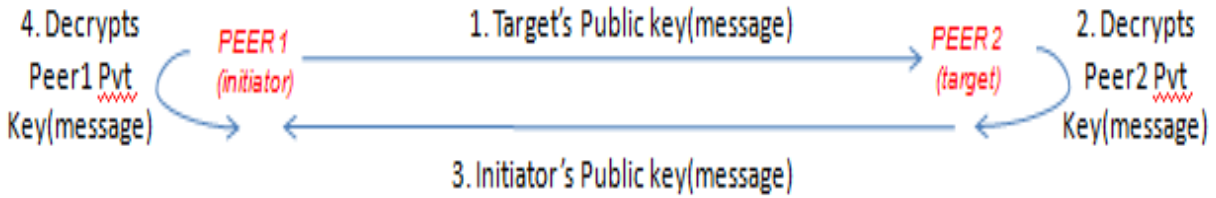
DISCOVERY

- 2^{160} network address space, $O(\log_n)$ hops. Iterative, asynchronous look up.

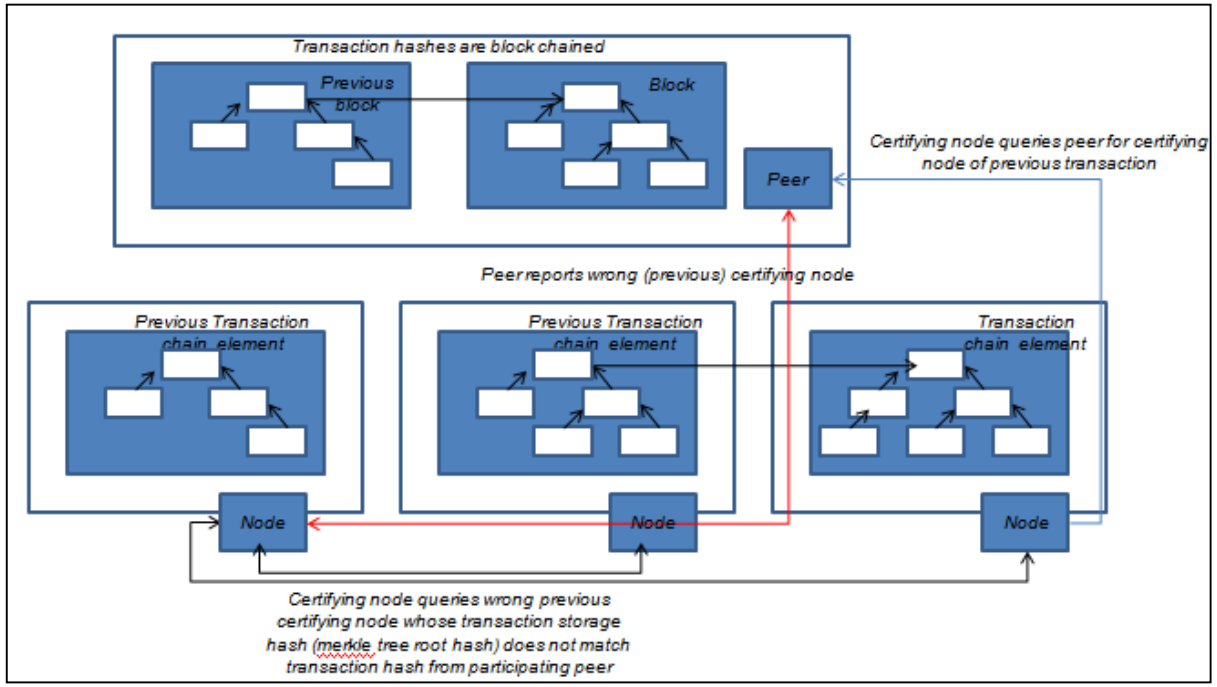


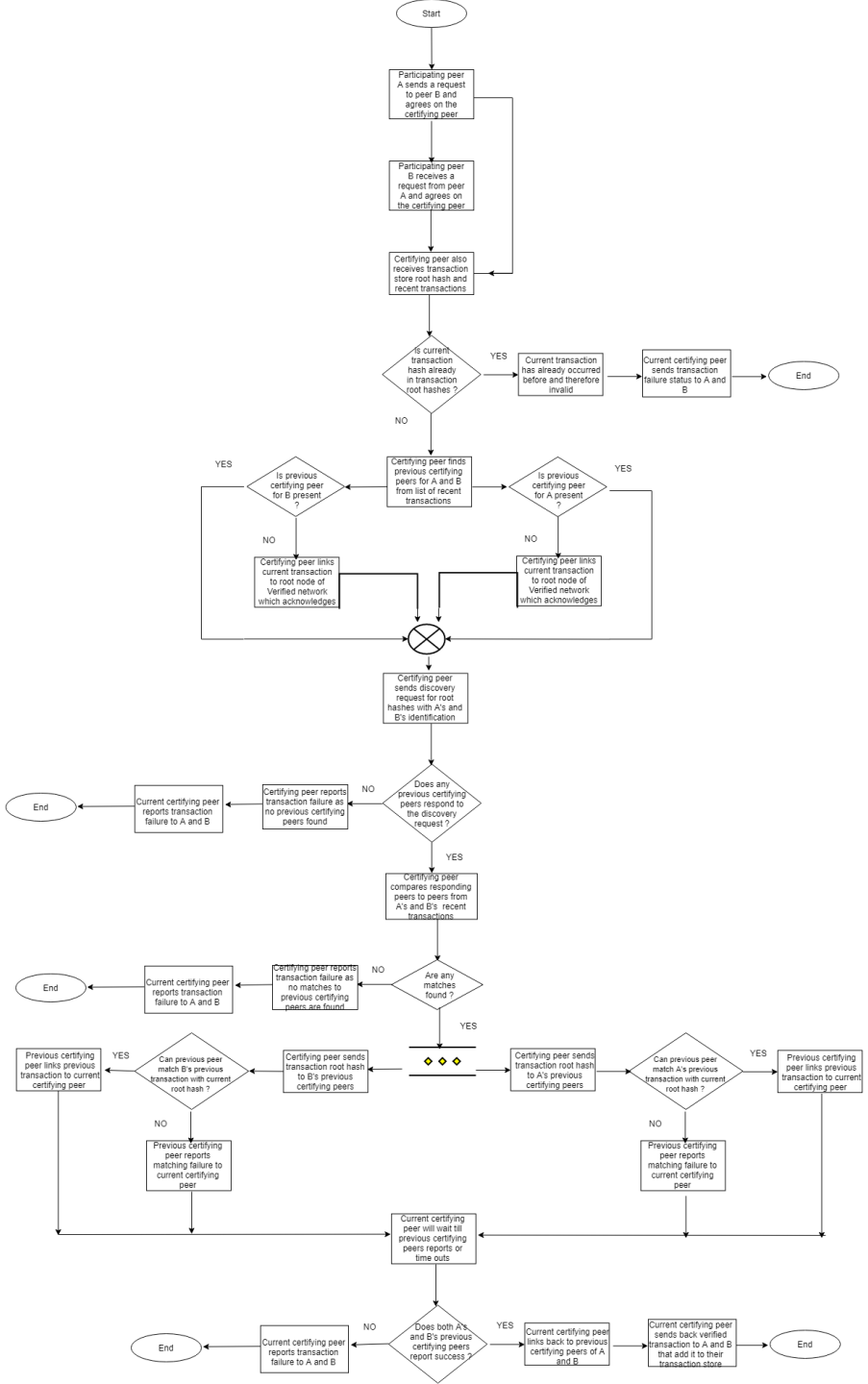
COMMUNICATIONS

- Sender signs message with its private key, encrypts using public key of receiver. Receiver decrypts message using its private key, checks signature using sender's public key.



Verifying a transaction





Provided and Required APIs

- Required APIs
  - Asset APIs
    - requestIssue()
    - requestRedemption()
    - requestExchangeRate()
  - Storage APIs
    - TBD

- Provided APIs
  - Authentication APIs
    - signUp()
    - signIn()
    - signOut()
  - Asset store APIs
    - checkBalance()
    - canDebit()
  - Transaction APIs
    - send()
    - publish()
    - subscribe()
    - verify()
    - store()

Setting up the Verified blockchain

- Decentralized
  - Verified peers run dhtRunners themselves
- Distributed
  - Verified peers connect to dhtRunner nodes running remotely either using RPCs or REST APIs