In The Name Of ALLAH

Introduction To Quorum

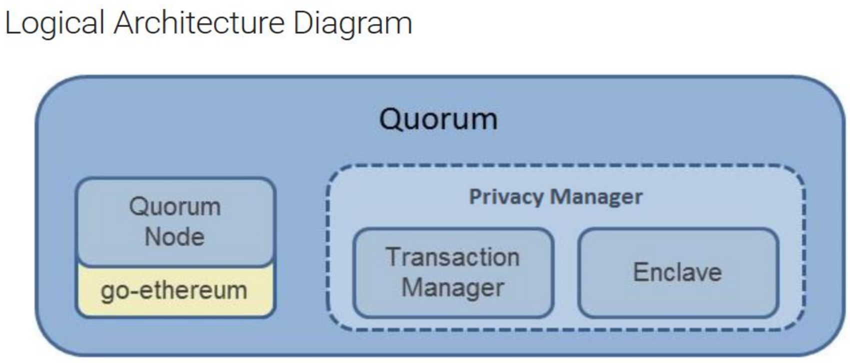
Quorum is an Ethereum-based distributed ledger protocol that has been developed to provide industries such as finance, supply chain, retail, real estate, etc. with a permissioned implementation of Ethereum that supports transaction and contract privacy.

The primary features of Quorum, and therefore extensions over public Ethereum, are:

- Transaction and contract Privacy ---- segmentation/cryptography
- Multiple voting-based consensus mechanisms ---- raft- consensus
- Network/Peer permissions management ---- having TX manager /application layer
- Higher performance ---- by removing gaslimit ,dozen of hundreds TX/s

Quorum currently includes the following components:

- Quorum Node (modified Geth Client)
- Privacy Manager (Constellation/Tessera)
  - Transaction Manager
  - Enclave



General Quorum algorithm for having private transaction is to ***separate the public chain from private chains***.in other word each node is in public chain and each private chain which is pary to , that means ***private transaction are be validating and executing only by nodes that are party to and just their hash will go to public chain***. For this purpose quorum uses different cryptographic methods. Some methods are listed here:

Cryptographic methods:

•Homomorphic encryption

•Zero-knowledge proofs

•Secure multi-party computation

•Ledger segmentation

•Cryptographic protocols

…

Quorum consensuses for including privacy are **raft (default) and Istanbul BFT** now we introduce their features and mechanism for short.

Raft-based consensus :

 faster blocktimes,transaction finality and on-demand block creation

Achive to data privacy by "private transaction"

Minimize the changing on go-ethereum → so syncing with updates ethereum become easier

Additional privacy functionalities are in a layer top of standard ethereum (e.g. for building a private contract you first write a simple one like ethereum then use functions like abi to make it private)

Every node validating the list of transactions while only exposing details of private transactions and contrats to relevant parties

Able to process hundreds of transactions per second depending on system profiling

Nodes:

Simple nodes

Network manager

Transaction manager

Enclave

Details about consensuses;

Raft:
good for closed-membership / consortium

Byzantine fault tolerance is not required

Leader / Follower model

At first all nodes are candidates . when a node elected by majority become leader.

Forking is not allowed

Only leader node mint new block ; He bundles transaction in block without POW . it's new block will be the new head of chain when it has been **verified by majority**. Against ethereum which a block is written and become the head immediately . for more documentation about Raft consensus see here .

Istanbul-BFT : (Byzantine fault Tolerance*)

this is a **three phase consensus** : **PRE PREAPARED , PREAPARED , COMMITED** . and it can tolerate **1/3 of 'validator' nodes being faulty**. the validator nodes pick a proposer node

the process is this: the proposer node , propoe a new block and broadcast it to validators by PRE PREAPARED massage. validators recieve it and come to PRE PREAPARED level.

then send PREAPARED massage.when 2/3 nodes send this massage the network come to PREAPARED level. at last each of them check the block and send COMMITED massage. when 2/3 of them send COMMITED, the block will commit to head of blockchain.

so forking is pretended . there's also an 'extradata' in blockheader to prevent faulty nodes to make a fake chain . for more documentation about Istanbul-BFT consensus see here .

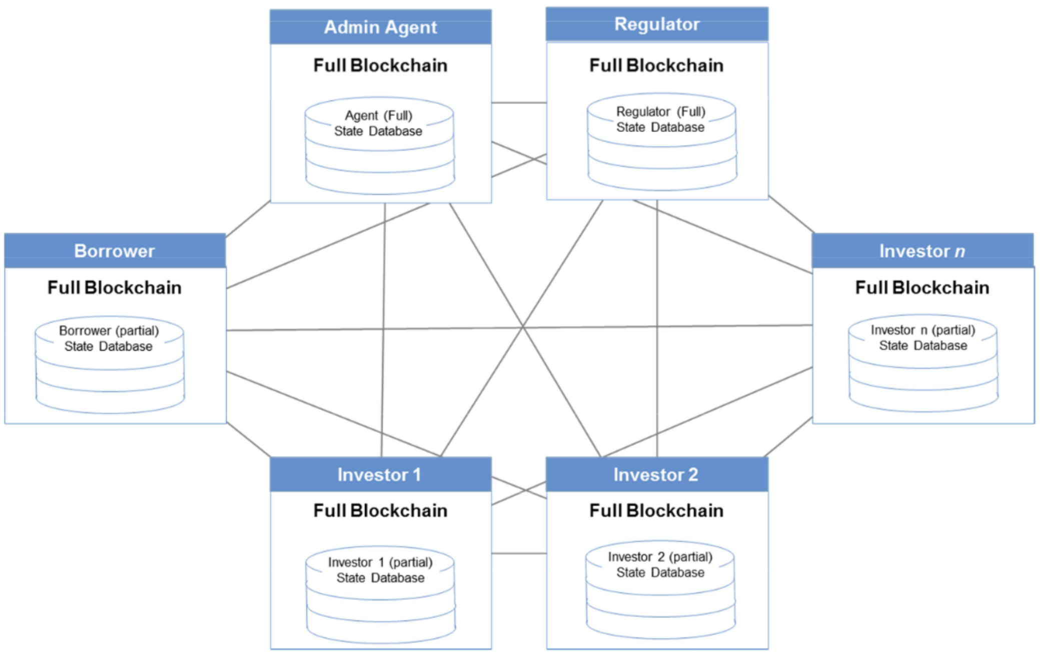
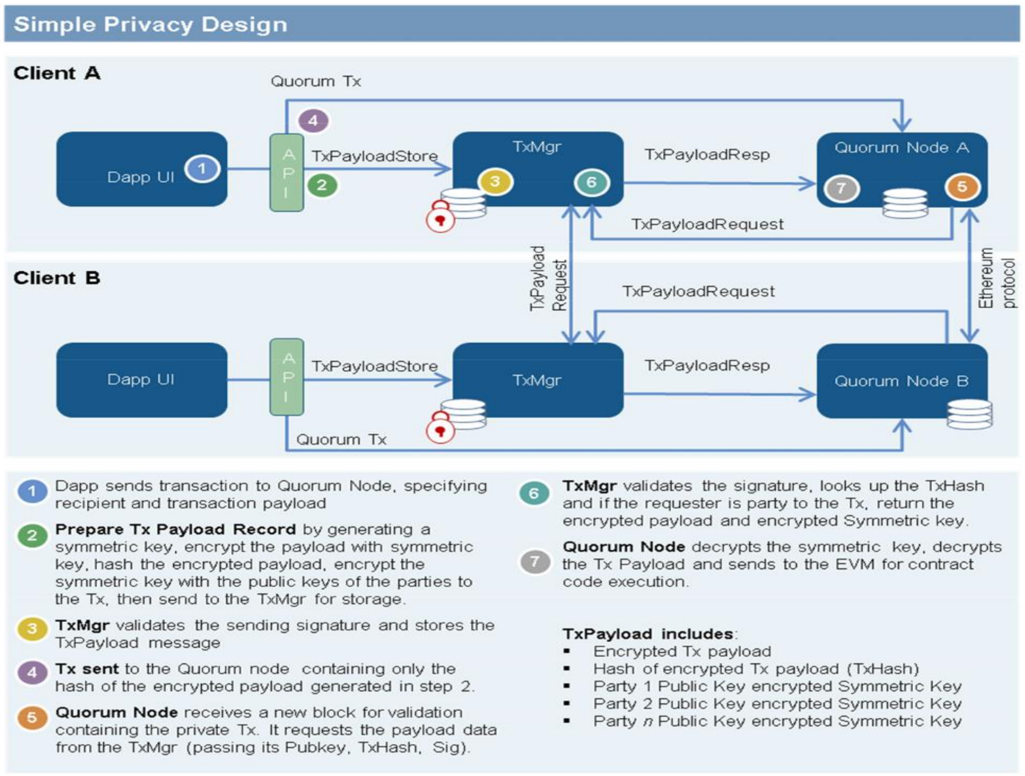

Admin Agent
Full Blockchain
Agent (Full) State Database
Regulator
Full Blockchain
Regulator (Full) State Database
Borrower
Full Blockchain
Borrower (partial) State Database
Investor n
Full Blockchain
Investor n (partial) State Database
Investor 1
Full Blockchain
Investor 1 (partial) State Database
Investor 2
Full Blockchain
Investor 2 (partial) State Database

Figure 2 - Quorum Network



1. Dapp sends transaction to Quorum Node, specifying recipient and transaction payload
2. **Prepare Tx Payload Record** by generating a symmetric key, encrypt the payload with symmetric key, hash the encrypted payload, encrypt the symmetric key with the public keys of the parties to the Tx, then send to the TxMgr for storage.
3. **TxMgr** validates the sending signature and stores the TxPayload message
4. **Tx sent** to the Quorum node containing only the hash of the encrypted payload generated in step 2.
5. **Quorum Node** receives a new block for validation containing the private Tx. It requests the payload data from the TxMgr (passing its Pubkey, TxHash, Sig).
6. **TxMgr** validates the signature, looks up the TxHash and if the requester is party to the Tx, return the encrypted payload and encrypted Symmetric key.
7. **Quorum Node** decrypts the symmetric key, decrypts the Tx Payload and sends to the EVM for contract code execution.

TxPayload includes:
- Encrypted Tx payload
- Hash of encrypted Tx payload (TxHash)
- Party 1 Public Key encrypted Symmetric Key
- Party 2 Public Key encrypted Symmetric Key
- Party *n* Public Key encrypted Symmetric Key

After introducing to basic to general quorum for continue we go step by step by features;

Data Privacy:
There are two approches for this. Segmentation and cryptography
Cryptography is applied on data which exist on transactions and all nodes can see them.
Segmentation is in each local state database nodes. Which result that for a node only public and private chains will update that is party to.

---

* A Byzantine fault (also interactive consistency, source congruency, error avalanche, Byzantine agreement problem, Byzantine generals problem, and Byzantine failure[1]) is a condition of a computer system, particularly distributed computing systems, where components may fail and there is imperfect information on whether a component has failed. The term takes its name from an allegory, the "Byzantine Generals Problem",[2] developed to describe a situation in which, in order to avoid catastrophic failure of the system, the system's actors must agree on a concerted strategy, but some of these actors are unreliable.

In a Byzantine fault, a component such as a server can inconsistently appear both failed and functioning to failure-detection systems, presenting different symptoms to different observers. It is difficult for the other components to declare it failed and shut it out of the network, because they need to first reach a consensus regarding which component has failed in the first place.

Byzantine fault tolerance (BFT) is the dependability of a fault-tolerant computer system to such conditions.

☑

Private transactions & contracts:

Private contract are which is built by private transaction. for vallidating process the data in private transaction will decrypt by node party and send to EVM so the EVM doesn't involve in decrypt/encrypt operations.

Private transactions:

A private transaction just have a hash in chain and it's 'v' value is 37 or 38 against the usual ethereum which is 27 or 28.

so for sending a private transaction the new API just have to set "privat for" parties and then send like a simple ethereum transaction. then it's information's Hash go into public chain and all nodes will see that just.

generally a Tx contents are these:

The recipient
Signature identifying the sender
The amount of ethereum
(Optional) private for
(Optional) data field

Private Contract:

created by privat transactions and updates in it's own seperate partia-merkle tree
so can not creat private contract by a public transaction because it will record on public merkle tree.

Block Validation:

Usual block validation in ethereum , here is just use for public chain validating.
For private chain it goes to application layer and is supported by a new storageRoot RPC API .
When private nodes require cryptographic state consensus evidence the app retrieve the private contract state hash for a specified block and share with parties off-chaine or on a transaction.
At last we have same blockchain of transactions and the public transactions result in public state consensus and since the EVM is determinisic then it should be impossible to have the EVM produce a different state of a private contract by processing a private transaction.
Block time in permissioned quorum actually doesn't have the ethereum problem. because there ,you may have bad nodes who wants to attack and we have to set limits on blocksize.
but in a private one we have more controls on nodes and beside that all TX hash will be in public chain.

Privacy through Sharding:

Again point that each node can process public and party transactions not all..

Performance:
As mentioned **, by removing the gaslimit the performance of quorum wich is tested throughput of dozens of hundreds os transactions per second**

Application:
Privacy is really important for financial institutions which are on consortium network , for example third party shouldn't be aware of transaction of two part.
and **as we mentioned security validation on private network is made by application layer.**

Payments:
There are various types of payment for different purposes. The simplest one doesn't need smart contract and just put transaction in the block.
Or other types payment applications for keeping the balances use smart contracts .**The point is that having the flexibility to manage private state consensus at the application layer is a powerful tool to help find designs to solve for the functionality and privacy requirements of many use cases**. There is a helpful example at the end of Quorum whitepaper about private payments that is described 'bank'

Connection between private and public:
This model imposes a restriction in the ability to modify state in private transactions. Since it's a common use case for a (private) contract to read data from a public contract the virtual machine has the ability to jump into read only mode

☑1. S -> A -> B
☑2. S -> (A) -> (B)
☑3. S -> (A) -> [B -> C]

and the following transaction are unsupported:

☒1. (S) -> A
☒2. (S) -> (A)

Where: - S = sender - (X) = private - X = public - -> = direction - [] = read only mode

The RPC method " eth_storageRoot(address[, blockNumber]) -> hash " can be used. It returns the storage root for the given address at an (optional) block number. If the optional block number is not given the latest block number is used. The storage root hash can be on or off chain compared by the parties involved.
A brief of it's properties are listed here:

•Consensus is achieved with the Raft or Istanbul BFT consensus algorithms instead of using Proof-of-Work.
•**The P2P layer** has been modified to only allow connections to/from permissioned nodes.

•The block generation logic has been modified to **replace the 'global state root' check with a new 'global public state root'**.
•The block validation logic has been modified to replace the 'global state root' in the block header with the 'global public state root'
•The State Patricia trie has been split into two: a public state trie and a private state trie.
•Block validation logic has been modified to handle 'Private Transactions'
•Transaction creation has been modified to **allow for Transaction data to be replaced by encrypted hashes in order to preserve private data where required**
•**The pricing of Gas has been removed, although Gas itself remains**

Transaction Manager

Quorum's Transaction Manager is responsible for Transaction privacy**. It stores and allows access to encrypted transaction data, exchanges encrypted payloads with other participant's Transaction Managers but does not have access to any sensitive private keys**. It utilizes the Enclave for cryptographic functionality (although the Enclave can optionally be hosted by the Transaction Manager itself.)

The Transaction Manager is restful/stateless and can be load balanced easily.

For further details on how the Transaction Manager interacts with the Enclave, please refer here

The Enclave

Distributed Ledger protocols typically leverage cryptographic techniques for transaction authenticity, participant authentication, and historical data preservation (i.e. through a chain of cryptographically hashed data.) In order to achieve a separation of concerns, as well as to provide performance improvements through parallelization of certain crypto-operations, **much of the cryptographic work including symmetric key generation and data encryption/decryption is delegated to the Enclave.**

The Enclave works hand in hand with the Transaction Manager to strengthen privacy by managing the encryption/decryption in an isolated way. It holds private keys and is essentially a "virtual HSM" isolated from other components.

For further details on the Enclave, please refer here.
That was an introduction of Quorum theory. Practical and building Quorum network Dock is in the next chapter.

Thanks for your attention
Pouria Dadkhah