

Quorum Whitepaper

1 Introduction

Smart Contracts on a replicated, shared ledger hold the promise of improving efficiency and lowering costs compared with existing enterprise systems that are based on duplicated business logic and consensus by reconciliation. Existing smart contract systems on replicated shared ledgers are unable to provide data privacy, transactions and smart contract state data are exposed in the clear on the replicated shared ledger. A number of proposals to address the lack of data privacy with Smart Contracts are beginning to emerge involving methodologies such as homomorphic encryption, zero-knowledge proofs, secure multi-party computation, ledger segmentation, cryptographic protocols, etc. This paper outlines a simpler ‘low-tech’ approach to privacy built on the Ethereum platform. Though the design is low-tech, the solution preserves many of the key attributes of Ethereum such as insuring every node on the network participates in and increases the overall security of the entire network while only revealing the details of private transactions to those party to the transactions.

This paper focuses on the modifications to standard Ethereum in order to support the desired ‘Enterprise’ requirements. This paper assumes the reader is familiar with Ethereum and has read the Ethereum whitepaper¹.

2 Quorum

Quorum is a private/permissioned, blockchain based on the official Go implementation of the Ethereum protocol². Quorum uses a voting based consensus algorithm, and achieves Data Privacy through the introduction of a new “private” transaction type. One of the design goals of Quorum is to reuse as much existing technology as possible, minimizing the changes required to go-ethereum in order to reduce the effort required to keep in sync with future versions of the public Ethereum code base.

Much of the logic responsible for the additional privacy functionality resides in a layer that sits atop the standard Ethereum protocol layer. Figure 1 below provides a high-level overview of the Quorum blockchain platform and architectural components.

¹ <https://github.com/ethereum/wiki/wiki/White-Paper>

² <https://github.com/ethereum/go-ethereum>

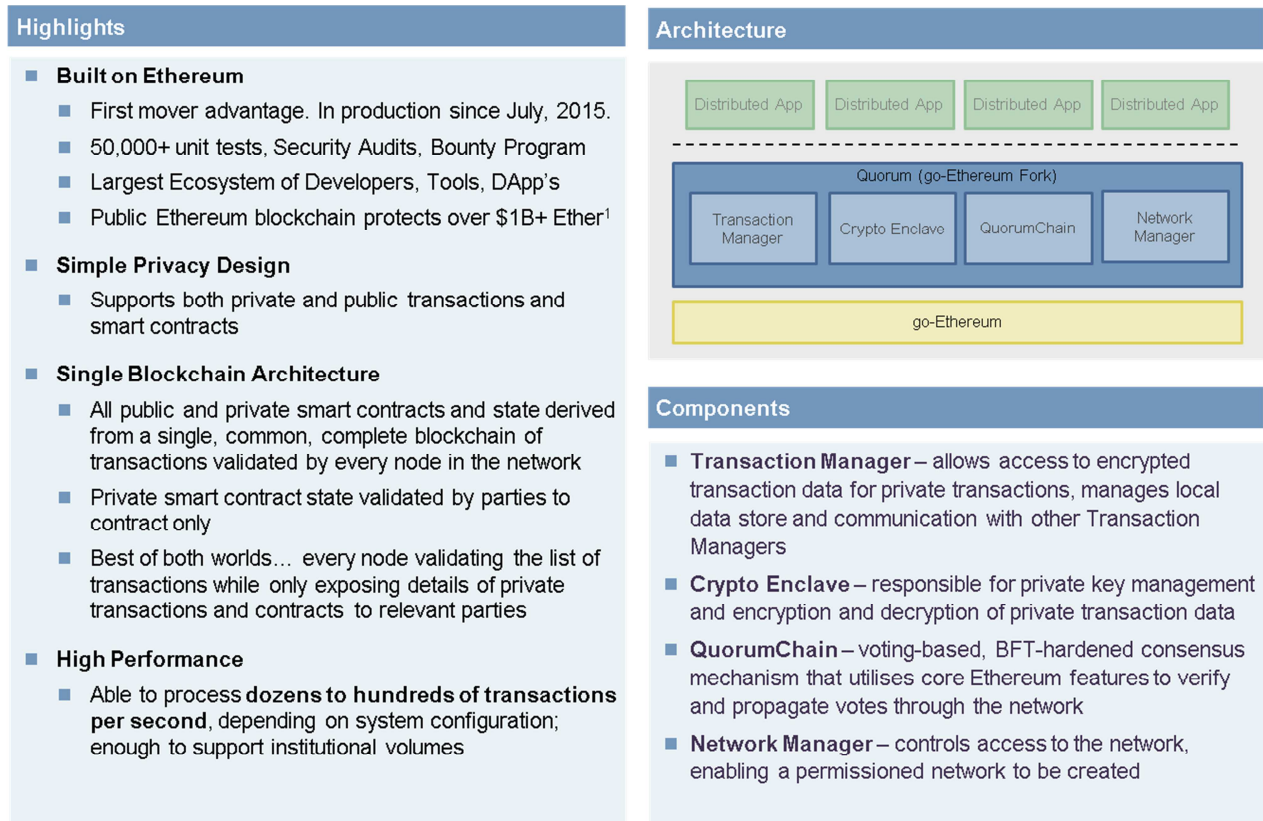


Figure 1 - Quorum Overview

The basic idea behind Quorum is to use cryptography to prevent all except those party to the transaction from seeing sensitive data. The solution involves a single shared blockchain and a combination of smart contract software architecture and modifications to Ethereum. The smart contract architecture provides segmentation of private data. Modifications to the go-ethereum codebase include modifications to the block proposal and validation processes. The block validation process is modified such that all nodes validate public transactions and any private transactions they are party to by executing the contract code associated with the transactions. For other "private transactions", a node will simply skip the contract code execution process.

This will result in a segmentation of the state database, i.e. the state database is split into a private state database and a public state database. All nodes in the network are in perfect state consensus on their public state. The private state databases will differ. Even though the client node state database no longer stores the state of the entire global state database, the actual distributed blockchain and all the transactions therein are fully replicated across all nodes and cryptographically secured for immutability. This is an important distinction relative to other segmentation strategies based on multiple block chains and adds to the security and resiliency of the design (see figure 2 below).



Figure 2 - Quorum Network

3 Quorum Chain Consensus

Quorum is a majority voting protocol where a subset of nodes within the network are given the voting role. The voting role allows a node to vote which block should be the canonical head on a particular height. The block with the most votes will win and is considered the canonical head of the chain.

Block creation is only allowed by nodes with the `block maker` role. A node with this role can create a block, sign the block and put the signature within in `extraData` field of the block. On block import nodes can verify if the block was signed by one of the nodes that have the `block maker` role.

Nodes can be given no role, one of the roles or both roles through command line arguments. The collection of addresses with special roles is tracked within the Quorum smart contract.

Quorum is implemented in a smart contract. Voters and block makers can be added or removed and the minimum number of votes before a block is selected as winner can be configured.

4 Data Privacy

Data privacy in Quorum is achieved through cryptography and segmentation. Cryptography is applied to the data in transactions (which everyone sees on the blockchain). Segmentation is applied to each nodes local state database (which contains the contract storage and is only accessible to the node). Only nodes party to private transactions are able to execute the private contract code associated with the transactions which results in updating the private contract data storage in the local state database. The result is that each node's local state database is only populated with public and private data they are party to. Figure 3 below outlines the high-level logical design of the Quorum privacy solution.

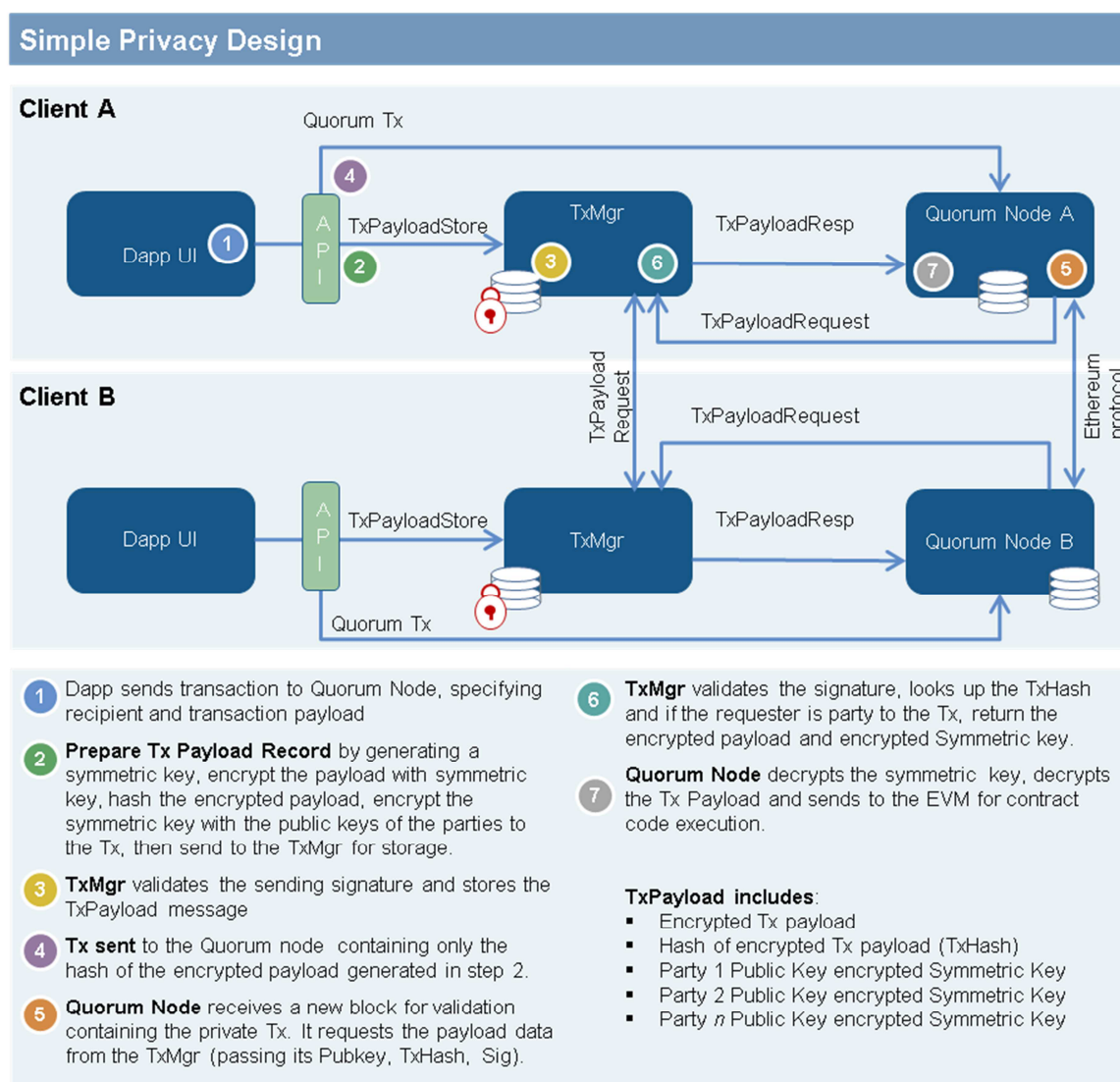


Figure 3 - Simple Privacy Design

4.1 Private Transactions and Private Contracts

Private transactions are facilitated through an API exposed to the Dapp that originates the transaction. A private contract is simply a contract created by a private transaction.

During the block construction and validation process the node party to the private transaction will decrypt the transaction data prior to sending to the EVM. The EVM therefore does not need to support encryption/decryption operations. The private state data of a private contract is stored in the clear in the local state database but this is fine as only nodes party to the transaction will store the data.

4.1.1 Private Transactions

A Private Transaction is a transaction that only carries a 256-bit hash in the data field. It is also identified by the v value of the transaction object as a 27 or 28.

In addition to the basic data a typical Ethereum sendTransaction accepts the new API takes a list of public keys that identify the parties to the transaction (privateFor). With this data a standard Ethereum transaction is generated where the payload is simply the hash of the encrypted private data. This newly formed Ethereum transaction carrying only the cryptographic hash is sent to the Quorum node where it is distributed out to all the nodes in the network as an pending transaction.

A Quorum transaction contains:

- The recipient
- Signature identifying the sender
- The amount of ether
- An optional “privateFor “ (identifies the transaction as a private transaction)
- An optional data field (a 256-bit hash in the case of a private transaction)

The figure below illustrates the basic steps in producing and sending private transactions.

4.1.2 Private Contracts

A private contract is a contract that was created by a private transaction. The state of private contracts are represented as their own separate patricia-merkle trie. One cannot create a private contract with a public transaction because the state of the contract created with a public transaction is recorded in the separate public state patricia-merkle trie.

4.2 Block Validation and State Consensus

Standard Ethereum block validation includes a step to confirm the global state of all contracts match the global state hash included in the block header. This is the cryptographic proof that every node in the network has the exact same state database, a provable replica.

The Quorum state database is split into two, a private state and a public state. These are represented in memory as two separate patricia-merkle trees. Quorum block validation only matches on the public state.

Private state consensus is passed up to the application layer and is supported by a new *storageRoot* RPC. When parties to a private contract require cryptographic state consensus evidence the application can retrieve the private contract state hash for an specified block and share this value with the parties to the contract either off chain or through an on-chain transaction. It should be noted that a specific application design may not require or infact expect parties to the same contract to have different state (see section 5.1 Payments for an example). It should also be noted that if public state consensus

achieved, it should be impossible for parties to a private transactions to be out of consensus. Here's the basic argument:

1. Validating a block includes global transaction hash consensus and public state consensus and a few other checks.
2. The EVM is deterministic... same inputs always generate same outputs.
3. The inputs are the same because we're confirming consensus on the list of transactions.
4. We know we're using the same blockchain of transactions because we are achieving public state consensus.
5. If we have the same blockchain of transactions and the public transactions result in public state consensus and since the EVM is deterministic then it should be impossible to have the EVM produce a different state of a private contract by processing a private transaction.

In public Ethereum, in practice, the only time blocks are invalidated is due to a fundamental configuration problem, for example if your genesis block is misconfigured. Once you are properly connected to the network and syncing blocks with your peers, you never actually fail blocks because of a global consensus break. Occasionally in the public network a malicious node(s) will attempt to send bad blocks. Of course the threat of a malicious node is more real in the public Ethereum network so it's critically important to validate global consensus but in reality, because everyone knows that a malicious node would be detected and their blocks rejected no one bothers with that attack vector.

In permissioned Quorum, the same sort of thing occurs in that if you have a fundamental configuration problem, you will quickly see blocks not validating and syncing with the network. If you are able to get blocks to sync by only looking at public state (and don't forget every block has public transactions for block voting) then there is no fundamental configuration problem and nodes party to the same private transactions and contracts will deterministically arrive at the same private state for those contracts. Throwing in the failsafe detection of comparing private contract state hashes (at the application layer) is a simple and easy way and a good practice to detect malicious nodes if that's what your design calls for.

4.3 Privacy through Sharding?

It's worth mentioning that the Quorum privacy design exhibits several properties of sharding techniques that have been proposed to enhance the performance and throughput of blockchain networks. Sharding basically segments the validation of transactions such that not every node in the network is validating every transaction. In Quorum, a node is only processing transactions that are public or private transactions that they are party to.

5 Performance

Tests of Quorum have demonstrated throughput of dozens to hundreds of transaction per second depending on system configuration. No specific code changes were required, rather simply updating the block gaslimit parameter enabling the node to pack in many more transactions per block. The gating factor on throughput is the rate at which the geth node is able to validate blocks in which the processing of transactions must be serial.

6 Applications

Financial applications come first to mind when considering the requirements for privacy. In a consortium network of multiple financial institutions, it is essential to insure the details of transactions between financial institutions are not revealed to other financial institutions that are not party to the transactions. This is for both client privacy concerns as well as meeting legal and regulatory requirements. Several financial application use cases have been considered during the design of the privacy solution of Quorum. From various (simple, high throughput) payment applications to derivative and loan use cases.

It's important to understand the Quorum privacy design does impose constraints on the application design. For example, updating a public contract state from a private contract is not permitted. Also, a private contract may not create a public contract. In general, private transactions cannot change public state, doing so would break the block validation/consensus algorithm.

Nevertheless, these constraints have not been unsurmountable in finding an an application/smart contract design for the financial use cases considered to date.

6.1 Payments

Various payment type use cases are useful proxies for many other financial applications. The simplest form of payment use case does not even require the use of smart contracts. The consensus critical information can be represented entirely in the data of the transaction itself and the blockchain of transactions.

Other types of payment applications may choose to use smart contracts to keep track of balances in lines of credit. 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. For example, consider an inter-bank payment use case where a payment from an account at Bank A is made to an account at Bank B. Bank C is not party to the payment and should not be able to know any information associated with this payment. It is the goal of the application design to insure that sufficient balance exists in accounts to cover the payment.

One particular design calls for the creation of a 'Bank' contract for each bank as illustrated in figure 4. In this design each Quorum node in the network would have 3 contracts, one for each Bank A, B, and C. In addition, a regulator node would be part of the network and would be party to all payment transactions. It is a further design requirement that payment transfers between accounts and banks be atomic operations, i.e. a single transaction must update both Bank contracts, all or nothing. In this design, the state of contract Bank A would only be in consensus with the Regulator node (and other Bank A nodes). The Bank A contract on Bank A's node would see all debits and credits between itself and all the Banks in the network. From Bank B's perspective, the state of Contract Bank A would only reflect the debits and credits between Bank A and Bank B, it would not know of any payments between Bank A and Bank C. In this design, the application would seek to confirm state consensus of its own Bank contract with only the regulatory node. By design, the state of the same contract on different Bank nodes would not be in state consensus.

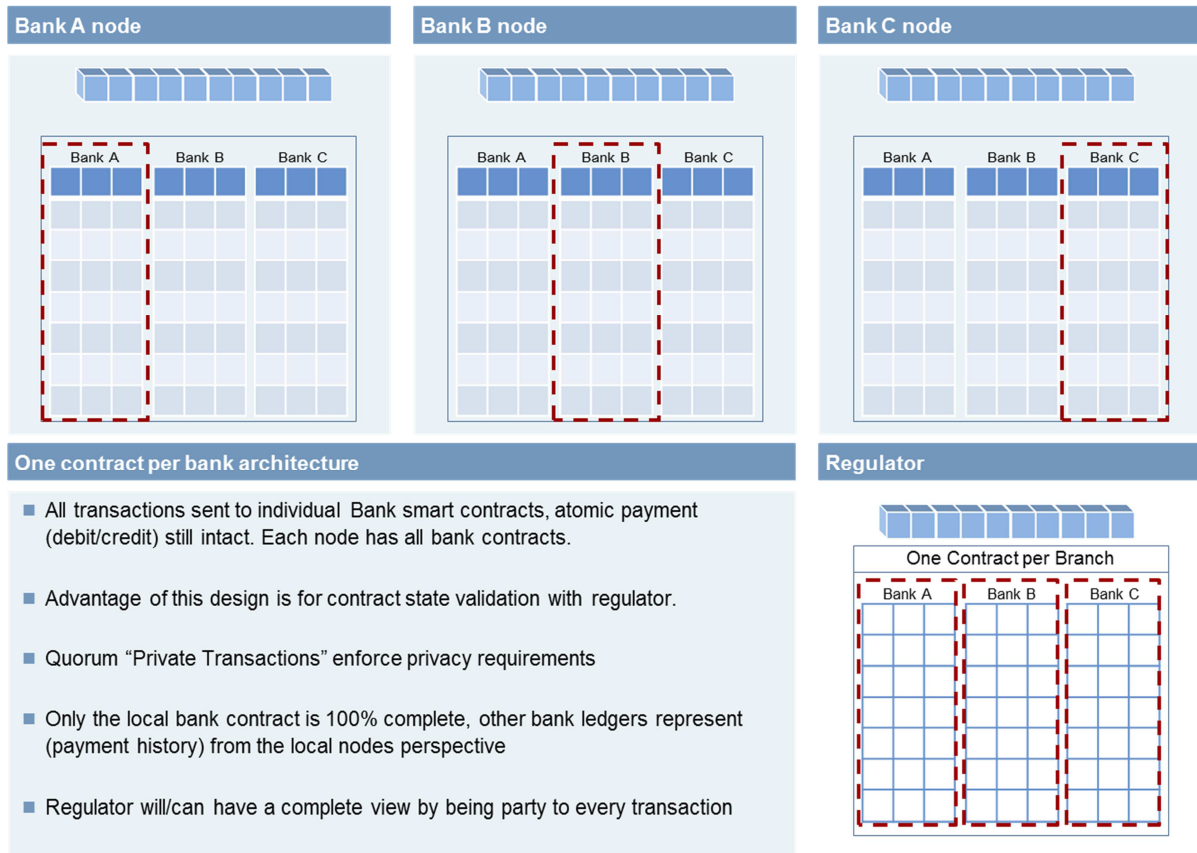


Figure 4 - Sample Payments contract design

7 Conclusion

The Quorum design outlined in this paper demonstrates how the seminal Ethereum platform can be extended to meet the key requirements of enterprise and in particular, financial institutions in preserving privacy in a distributed, decentralized blockchain consortium network. By building on Ethereum, Quorum inherits the maturity of the battle tested go-ethereum code base as well as helps unite the public and enterprise development communities on a common protocol.