

**Hyperledger Consensus**

**Mechanisms**



# **Consensus in Hyperledger Mechanisms**

## Introduction

A blockchain, is operated by unknown and untrusted parties i.e. we cannot know if it is an individual person, organization, a computer operating automatically, or whatever their decisions and actions. If any entity, individual, or party can submit information to the database of the blockchain.

## Consensus in Blockchain

Consensus is necessary for the distributed operators of the blockchain to evaluate and agree on all additions before they are permanently incorporated into the database & processing peers. It is the process of building agreement among a group of mutually distrusting participants for maintaining a consistently replicated ledger. Algorithms for achieving consensus with arbitrary faults generally require some form of voting among a known set of participants.

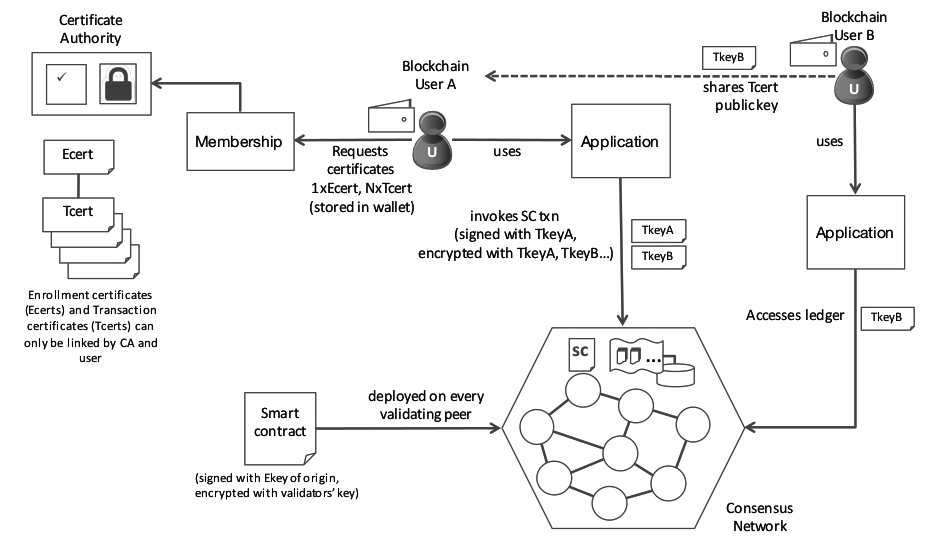
Consensus is the process by which a network of nodes provides a guaranteed ordering of transactions and validates the block of transactions. Consensus must provide the following core functionality:

* Confirms the correctness of all transactions in a proposed block, according to endorsement and consensus policies.
* Agrees on order and correctness and hence on results of execution (implies agreement on global state).
* Interfaces and depends on smart-contract layer to verify correctness of an ordered set of transactions in a block.

Based on consensus block chains can be defined in two types.

* **Permissionless Blockchain**: Every node in the network participate in consensus procedure, e.g. Bitcoin Blockchain (Proof of Work)
* **Permissioned Blockchain**: Only Selected nodes(validators, e.g. Government or trusted nodes) participate in consensus procedure e.g. Hyperledger Blockchain

**Hyperledger Consensus Network**



## Types of consensus

There are four main methods of finding consensus in a blockchain.

1. Practical byzantine fault tolerance algorithm (PBFT),

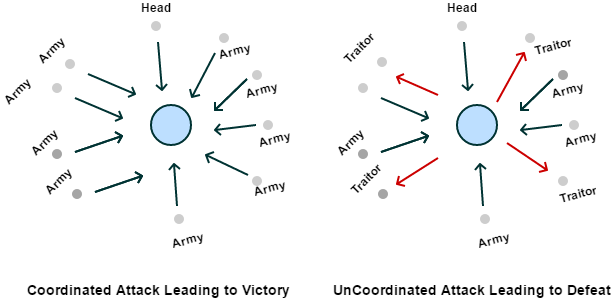
2. The proof-of-work algorithm (PoW),

3. The proof-of-stake algorithm (PoS), and

4. The delegated proof-of-stake algorithm (DPoS).

### 1. Practical byzantine fault tolerance algorithm (PBFT)

The Practical Byzantine Fault Tolerance Algorithm (PBFT) was designed as a solution to a problem presented in the form of example byzantine army attack.



Here is a group of Byzantine generals and they want to attack a city. They are facing two very distinct problems:

* The generals and their armies are very far apart so centralized authority is impossible, which makes coordinated attack very tough.
* The city has a huge army and the only way that they can win is if they all attack at once.

In order to make successful coordination the armies on the left of the castle send a messenger to the armies on the right of the castle with a message that says “ATTACK WEDNESDAY.” However, suppose the armies on the right are not prepared for the attack and say, “NO. ATTACK FRIDAY” and send back the messenger through the city back to the armies on the left. Also they face problems from poor messenger & message change/block. What these generals need, is a consensus mechanism which can make sure that their army can actually attack as a unit despite all the setback.

Each ‘general’ maintains an internal state (ongoing specific information or status). When a ‘general’ receives a message, they use the message in conjunction with their internal state to run a computation or operation. This computation in turn tells that individual ‘general’ what to think about the message in question. Then, after reaching his individual decision about the new message, that ‘general’ shares that decision with all the other ‘generals’ in the system. A consensus decision is determined based on the total decisions submitted by all generals.

### 2. Proof of work

PoW is a system which uses a ‘hash function’ to create conditions under which a single participant is permitted to announce their conclusions about the submitted information, and those conclusions can then be independently verified by all other system participants. False conclusions are prevented by the parameters of the hash function, which ensure that false information will fail to compute in an acceptable way. It requires expensive computer calculations. Therefore this process of searching for valid ‘hashes’ (solutions to the ‘hash function’ created by the message input), is known as ‘mining. A reward is given to the first miner who solves each block problem.

### 3. Proof of stake

In this algorithms replace the hash function calculation with a simple digital signature which proves ownership of the stake. The network selects an individual to approve new messages (that is to say, confirm the validity of new information submitted to the database) based on their proportional stake in the network. Proof of stake inputs can be several times more cost effective in terms of less computing calculations. Here the system is more centralized & there is no block reward.

### 4. Proof-of-stake (DPoS) system

It works along the same lines as the PoS system, except that individuals choose an overarching entity to represent their portion of stake in the system. So imagine, each individual decides if entity 1, 2, or 3 (these could be, for example, computer servers, and are called ‘delegate nodes’ within a DPoS system) will ‘represent’ his or her individual stake in the system. This allows individuals with smaller stakes to team up to magnify their representation, thereby creating a greater network centralization to help balance out the power of large stake holders.

## **Consensus in Hyperledger frameworks**

General approaches include Nakamoto-style consensus, which elects a leader through some form of lottery, and variants of the traditional Byzantine Fault Tolerance (BFT) algorithms, which use multiple rounds of explicit votes to achieve consensus.

### Consensus in Hyperledger Sawtooth Framework

Sawtooth abstracts the core concepts of consensus and isolates consensus from transaction semantics. The interface supports plugging in various consensus implementations. More importantly, Sawtooth allows different types of consensus on the same blockchain. The consensus is selected during the initial network setup and can be changed on a running blockchain with a transaction.

Sawtooth currently supports these consensus implementations:

* Proof of Elapsed Time (PoET), a Nakamoto-style consensus algorithm that is designed to be a production-grade protocol capable of supporting large network populations. PoET relies on secure instruction execution to achieve the without the power consumption drawbacks of the Proof of Work algorithm. PoET uses a lottery for leader election based on a guaranteed wait time provided through a Trusted Execution Environment (TEE). It randomly distributes leadership election across the entire population of validators with a distribution that is similar to what is provided by other lottery algorithms.
* PoET simulator, which provides PoET-style consensus on any type of hardware, including a virtualized cloud environment.
* Dev mode, a simplified random-leader algorithm that is useful for development and testing.

### 2. Consensus in Hyperledger Iroha

Iroha introduces a Byzantine Fault Tolerant consensus algorithm called Sumeragi. It is heavily inspired by the B-Chain algorithm.

Consensus in Sumeragi is performed on individual transactions and on the global state resulting from the application of the transaction. When a validating peer receives a transaction over the network, it performs the following steps in order:

* validate the signature (or signatures, in the case of multisignature transactions) of the transaction
* validate the contents of the transaction, where applicable (e.g., for transfer transactions, is the balance non-negative)
* temporarily apply the transaction to the ledger; this involves updating the Merkle root of the global state
* sign the updated Merkle root and the hash of the transaction contents
* broadcast the tuple (signed Merkle root, tx hash)

When syncing nodes with each other, valid parts of the Merkel tree are shared until the roots match.

### 3. Consensus in Hyperledger Fabric

In distributed ledger technology, consensus is synonymous with a specific algorithm, within a single function. However, consensus encompasses more than simply agreeing upon the order of transactions, and this differentiation is highlighted in Hyperledger Fabric through its fundamental role in the entire transaction flow, from proposal and endorsement, to ordering, validation and commitment.

Consensus is ultimately achieved when the order and results of a block’s transactions have met the explicit policy criteria checks. These checks and balances take place during the lifecycle of a transaction, and include the usage of endorsement policies to dictate which specific members must endorse a certain transaction class, as well as system chaincodes to ensure that these policies are enforced and upheld. Prior to commitment, the peers will employ these system chaincodes to make sure that enough endorsements are present, and that they were derived from the appropriate entities. Moreover, a versioning check will take place during which the current state of the ledger is agreed or consented upon, before any blocks containing transactions are appended to the ledger. This final check provides protection against double spend operations and other threats that might compromise data integrity, and allows for functions to be executed against non-static variables.

### 4. Consensus in Hyperledger Indy

Consensus in Hyperledger Indy is based on Redundant Byzantine Fault Tolerance (RBFT), which is a protocol inspired by Plenum Byzantine Fault Tolerance (Plenum). Think of RBFT as running several instances of Plenum in parallel. Ordered requests from a single instance called the master is used to update the ledger, but the master’s performance in terms of throughput and latency is periodically compared to the average performance of other instances. If the master is found to be degraded, a view change occurs that appoints a different instance to the role of master.

Hyperledger Indy uses RBFT to handle ordering and validation, which results in a single ledger containing both ordered and validated transactions. This is unlike many blockchain networks that use a Byzantine Fault Tolerance (BFT) protocol only for ordering. These networks leave domain-specific validation to happen after requests are ordered.

Plenum, and therefore RBFT, maintains a projection of the ledger called state. All valid, accepted operations performed may change the state, which is stored in a database as a collection of variables and their values. The state is kept in a cryptographically authenticated data structure called a Merkle Patricia tree, which is specified by Ethereum. Hyperledger Indy stores Decentralized Identifiers (DIDs) as state variables with values including the current verification key and a few other things. The in memory copy of the ledger transactions and resulting state are optimistically updated during the proposal phase. The primary is updated while sending the proposal and non-primaries are updated while accepting a valid proposal. The proposal gets ordered, then the ledger and the resulting state are committed. If the proposal gets rejected for some reason, then the changes made to the ledger and the resulting state are reverted.

### 5. Consensus in Hyperledger Burrow

Hyperledger Burrow is a permissioned blockchain node that executes smart contract code. Burrow is built for a multi-chain universe with application specific optimization in mind. Burrow as a node is constructed out of three main components; the consensus engine, the permissioned Ethereum virtual machine and the rpc gateway.

Here for consensus transactions are ordered and finalized with the Byzantine fault-tolerant Tendermint protocol. The Tendermint protocol provides high transaction throughput over a set of known validators and prevents the blockchain from forking.

## **Reference**

### Sample Hyperledger fabric:

### <http://hyperledger-fabric.readthedocs.io/en/release/capabilities.html>

### <http://hyperledger-fabric.readthedocs.io/en/release/samples.html>

### Github file Hyperledger fabric <https://github.com/hyperledger/fabric/blob/release/docs/source/samples.rst>

### Hyperledger Projects

### <https://www.hyperledger.org/projects>

### <https://www.ibm.com/blockchain/hyperledger.html>

### <https://www.sdxcentral.com/articles/news/whats-the-difference-between-the-5-hyperledger-blockchain-projects/2017/09/>

### Introduction to Blockchain and the Hyperledger Project

### <https://pt.slideshare.net/ManuelGarcia122/introduction-to-blockchain-and-the-hyperledger-project/30?smtNoRedir=1>

### <http://www.blogsaays.com/learn-chaincode-tutorial-ibm-bluemix/>

### Refer the Pdf file for your reference:

### <https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf>