

# Sawtooth: An Introduction

Kelly Olson, Mic Bowman, James Mitchell, Shawn Amundson, Dan Middleton, Cian Montgomery

January 2018

## Abstract

*Today's business processes for information sharing are burdened with intermediaries, inefficiencies, and security concerns. Through the use of distributed ledger technology (or blockchains), business processes between companies can be streamlined, and records can be kept in sync without the need for a central authority or manual reconciliation processes. This can help enterprises to reduce their costs and enable them to create entirely new ways of doing business. Sawtooth is a framework for building distributed ledgers for enterprise use, with a focus on modularity and extensibility. Sawtooth builds upon decades of state machine replication research, and is designed to support any consensus mechanism or 'smart contract' language.*

## ABOUT HYPERLEDGER

Hyperledger is an open source collaborative effort created to advance cross-industry blockchain technologies.

It is a global collaboration including leaders in finance, banking, Internet of Things, supply chains, manufacturing and Technology. The Linux Foundation hosts Hyperledger under the foundation.

## Background

### Bitcoin

Blockchain technology was first introduced with the release of the Bitcoin White Paper by Satoshi Nakamoto in 2008. The Bitcoin blockchain was designed to create a peer-to-peer electronic cash system with no central issuer. Approximately every 10 minutes, the nodes on the network come to consensus on a set of unspent coins, and the conditions required to spend them. This data set, known as the 'unspent transaction output', or UTXO, set, can be modified by submitting transactions to the network that replace one or more UTXOs with a new set of unspent transaction outputs. In order to ensure that all nodes on the network come to consensus on this dataset, the Bitcoin protocol leverages a set of transaction validation rules and a novel consensus mechanism known as proof-of-work which allows for permissionless and anonymous participation in the consensus protocol.

### Ethereum

In 2013, Vitalik Buterin published a White Paper that expanded on the blockchain concept by viewing the Bitcoin protocol as a state transition system. State machine replication is a technique used to create fault tolerant services through data replication and consensus, and was first introduced by Leslie Lamport in 1984. The goal of the Ethereum protocol was to create a 'decentralized application platform' that would enable developers to deploy 'smart contracts' that would run on the Ethereum state machine. 'Smart contracts' are state transition functions that take the current state of the ledger and a transaction, and produce a new ledger state. These 'smart contracts' could be used to not only dictate the logic required for the transfer of Ethereum's native currency, Ether, but could also be used to manage arbitrary key-value stores. These key-value stores are most often used to represent the ownership of other assets such as financial securities, land titles, and domain names.

### Distributed Ledgers

Since the release of Ethereum, a variety of other 'distributed ledgers' implementations have been created to meet the needs of the enterprise. These distributed ledger implementations include software that expands upon existing protocols (e.g. Bitcoin - MultiChain, Chain, Blockstream and Ethereum - Quorum) and creates entirely new implementations (Corda and Fabric). These implementations deviate from public blockchains to address limitations in existing protocols such as throughput, security, efficiency, usability, and confidentiality. Some distributed ledgers differ substantially from their blockchain predecessors as a result of changes to the underlying networking, journalling layer, smart contract, and consensus layers; some even going so far as to not include 'blocks' or a 'chain'.

### Related Technologies

- Peer-to-peer networking
- Event driven databases
- Replicated state machines
- Distributed databases
- Replicated logs
- Consensus algorithms

## Sawtooth

Sawtooth is a framework for building enterprise-grade distributed ledgers. It was designed with a focus on security, scalability, and modularity. The Sawtooth architecture has five core components:

1. **A peer-to-peer network** for passing messages and transactions between nodes
2. **A distributed log** which contains an ordered list of transactions
3. **A state machine / smart contract logic layer** for processing the content of those transactions
4. **A distributed state storage** for storing the resulting state after processing transactions
5. **A consensus algorithm** for achieving consensus across the network on the ordering of transactions and the resulting state

Sawtooth is designed with a modular consensus interface and a modular transaction processing platform to accommodate different threat models, deployment scenarios, and smart contract languages. Sawtooth has a 'traditional' blockchain architecture with a gossip network at its foundation, and a cryptographically linked chain of transaction blocks.

## Peer-to-Peer Communications

Nodes on the Sawtooth network communicate by sending messages (ØMQ messages) to each other over TCP. These messages contain information about transactions, blocks, peers, and more. All structured messages are serialized using Google's Protocol Buffers. Sawtooth currently relies on a gossip protocol to ensure that all messages about transactions and blocks are delivered to all nodes on the network. This is consistent with the design of most public blockchains such as Bitcoin and Ethereum.

In order to meet the needs of Enterprise deployments, Sawtooth supports network permissioning at the peer-to-peer layer. Through access control lists, Sawtooth nodes are able to control:


1. **who can connect to the network** and sync with the current ledger state
2. **who can send consensus messages** and participate in the consensus process
3. **who can submit transactions** to the network

Recently, newer implementations have abandoned the global broadcast technique to improve the privacy and confidentiality of transaction data. Two examples of this are R3's Corda and JPMorgan's Quorum, which both rely on point-to-point targeted sharing of transaction details. This is a key requirement for financial institutions as regulations often prevent the sharing of information, even in an encrypted format, with parties who are not involved in the transaction. The Sawtooth development team is currently exploring how the underlying ØMQ networking layer could be extended to enable this capability.

## Distributed Log

One of the foundational components of blockchains and replicated state machines is ensuring that nodes on the network process transactions in a consistent order. On Sawtooth, this ordered list of transactions is assembled into a data structure called a 'blockchain'. The 'blockchain' contains an ordered set of cryptographically linked blocks, beginning with a genesis block. Each block in the linked list contains an ordered set of transactions. Nodes reach consensus on the correct ordering of this log through a consensus algorithm.

Currently the Sawtooth blockchain arrives at a total ordering of every transaction. That is, every transaction is strictly before, or after, any other transaction. One area for exploration



is modifying the blockchain structure to allow for partial ordering between non-dependent transactions. On such data structure is a directed acyclic graph. This would potentially allow for the blockchain network to be partitioned or 'sharded' to improve scalability and reduce storage requirements.

## Smart Contracts

Smart contracts are small programs that take a transaction as an input, process it, and produce an output. This is the layer of the blockchain where business logic lives and where domain specific functions are implemented that are tailored to a specific use case. Smart contracts are best thought of as a state machines, where transactions are ingested from the distributed log along with a snapshot of the current state, and a new state is written to the distributed store after the transaction is processed. With this view, blockchain systems can be viewed through the lense of traditional replicated state machine architecture.

The use of smart contracts on a blockchain began with Bitcoin, which offered a very limited scripting language. This scripting language has been pruned even further after it's introduction as vulnerabilities were discovered when certain operations were performed. In addition, the Bitcoin blockchain has a 'stateless' design. Operations can either succeed, in which inputs are consumed and new unspent outputs are created, or they can fail. There is no ability for these smart contracts to have multiple stages or maintain any internal state. While this model provides security and scalability benefits, it can limit the use of the ledger to simple asset exchanges.


Ethereum expanded on the concept of smart contracts by processing transactions in a specially created virtual machine. Smart contracts are written in a higher level language, most often Solidity, that is then compiled down to Ethereum Virtual Machine bytecode. Smart contract logic can be deployed dynamically to the network by broadcasting contract code. This contract code would be used by the network participants to validate the transactions that are then subsequently sent to the contract address. Once a transaction is processed, the internal state of the smart contract is written to a distributed data store.

The Sawtooth framework expands on the concept of smart contracts by viewing the smart contract as a state machine, or 'transaction processor'. After passing through the distributed log, transactions are routed to the appropriate transaction processor. These transaction processors then ingest the payload of the transaction, as well as any relevant state, before processing the transaction. Sawtooth is capable of supporting both the stateless (UTXO) and stateful (Ethereum-style) models.

Sawtooth currently enables developers to develop smart contracts in a variety of languages such as:

- **Interpreted Languages:** Python and Javascript
- **Compiled Languages:** Rust, C++ and Go
- **Virtual Machines:** Ethereum Virtual Machine and Java

One of the fundamental tradeoffs in smart contract language design is between security and expressibility. By creating a domain specific transaction processor, it is much easier to limit the types of actions that can be performed on a blockchain network, which can improve security and performance. Furthermore, by leveraging more mature programming languages, it is possible to leverage existing tools that can perform static, dynamic, and formal analysis of your smart contract code.



At its heart Sawtooth is agnostic to the smart contract transaction processing language. Sawtooth has proven the ability to run Solidity contracts through its integration with the Hyperledger Burrow EVM. The Sawtooth development team intends to expand the programming languages that it supports, and to integrate with existing and emerging smart contract engines such as Chain's Ivy, and Digital Asset Holding's DAML.

The Sawtooth platform contains a set of smart contracts that are core to the platform's functionality:

1. **Settings** - Provides a reference implementation for storing on-chain configuration settings
2. **Identity** - Handles on-chain permissioning for transactor and validator keys to streamline access management
3. **Validator Registry** - Provides a methodology for registering ledger validators and trusted hardware attestations crucial to the security of PoET

In addition to these smart contracts, Sawtooth also makes available a number of additional transaction families for development, test, and use case prototyping:


1. **IntegerKey** - Simple business logic for testing deployed ledgers.
2. **BlockInfo** - Provides a methodology for storing information about a configurable number of historic blocks.
3. **XO** - Allows users to play a simple board game known variously as tic-tac-toe, noughts and crosses, and XO.
4. **Marketplace** - Allows users to issue and exchange quantities of customized "Assets" with other users on the blockchain
5. **Supply Chain** - Allows users to trace assets through the supply chain and immutably store telemetry data about the goods in transit such as temperature and humidity
6. **Smallbank** - Handles performance analysis for benchmarking and performance testing. This transaction family is based on the H-Store Smallbank benchmark.
7. **Seth** - An implementation of the Ethereum Virtual Machine, enabling EVM contracts to be deployed
8. **HyperDirectory** - A role-based access control system that allows the creation of users and permissions

## Distributed State

Ensuring consistency of state across nodes is an important feature for many blockchain applications, and is the key objective of replicated state machines. In Bitcoin, the state consists of a list of unspent coins, while Ethereum manages the state of its native currency, Ether, as well as the internal state of the smart contracts that run on its network. Sawtooth manages state in a similar way to Ethereum, by storing the serialized state of contracts in a Radix Merkle tree. Each transaction processor is allocated its own namespace to which it can write. The method of serialization is left up to the developer of the transaction processor, and different transaction processors could enforce different methods of serialization.

## Consensus

Consensus is the process by which nodes in the network come to agreement on the order of transactions and the resulting state. There are many consensus protocols and each has differing attributes relating to latency, throughput, population size, finality, censorship-resistance,



threat model, failure model, and more. Sawtooth does not take a fundamental position on which consensus mechanism is best, but rather offers a consensus interface that allows for a wide variety of consensus protocols to be used. There are two general approaches that have been proposed, “lottery” consensus and “voting” consensus. The first, often referred to as “Nakamoto consensus”, elects a leader through some form of “lottery”. The leader then proposes a block that can be added to a chain of previously committed blocks. In Bitcoin, the first participant to successfully solve a cryptographic puzzle wins the leader-election lottery. The second approach is based on traditional Byzantine Fault Tolerance (BFT) algorithms and uses multiple rounds of explicit votes among consensus participants to achieve consensus.

Sawtooth provides three consensus implementations, with a forth under active development:


- **Dev\_mode:** a simplified random leader algorithm that is useful for developers and test networks that require only crash fault tolerance.
- **PoET:** short for “Proof of Elapsed Time”, PoET is a Nakamoto-style consensus algorithm. It is designed to be a production-grade protocol capable of supporting large network populations that include byzantine actors. More information can be found about the PoET algorithm here - [PoET 1.0 Specification](#).
- **PoET-Simulator:** Sawtooth includes an implementation of PoET that simulates the secure instructions. This mechanism enables large-scale consensus populations, but forgoes Byzantine fault tolerance.
- **RAFT** (Under Development): The RAFT consensus protocol is a crash-fault tolerant ‘voting’-style consensus algorithm. It is designed for high throughput, low latency transactions.

## Privacy and Confidentiality

Blockchain privacy and confidentiality is one of the key challenges to blockchain adoption in the enterprise. To-date there have been two general means of providing privacy on a blockchain:

1. **Separation of Concerns** - Data is only sent to the relevant parties of a transaction, with an optional hash of that data broadcast to all blockchain nodes. This means of privacy has been implemented by R3’s Corda, JPMorgan’s Quorum, and Hyperledger Fabric through it’s channels model. While this method achieves the privacy and data locality concerns, it has a few drawbacks. The key disadvantage of this model is that it creates a number of ‘sub-chains’ and transferring assets between these chains adds significant complexity or reduces confidentiality.
2. **Computation on Encrypted Data** - In this approach, encrypted data is broadcast across the entire network in a way that the data can be computed on. This may entail advanced cryptographic techniques such as homomorphic encryption or zero-knowledge proofs, or leverage a trusted execution environment such as Intel® Software Guard Extensions (Intel® SGX). This method too comes with several drawbacks. Such drawbacks may include the need for trusted hardware or performance and usability limitations with advanced cryptographic techniques.

Currently Sawtooth does not dictate a privacy and confidentiality approach. The Sawtooth team has experimented with several approaches, including trusted computation of transaction off-chain, which can be found here - [https://sawtooth.hyperledger.org/docs/core/releases/0.8.8/examples/private\\_utxo/overview.html](https://sawtooth.hyperledger.org/docs/core/releases/0.8.8/examples/private_utxo/overview.html).



This area is an active area of research and a key priority for future Sawtooth releases. The Sawtooth team believes that different privacy models are appropriate for different deployments and intends to leverage data isolation, cryptographic, and trusted hardware techniques in their approach.

## Decentralized Governance

One of the unique features of Sawtooth is the on-chain governance mechanism enabled by the ‘settings’ transaction family. The settings transaction family provides a methodology for storing on-chain configuration settings such as which consensus mechanism is used and which transaction processors are enabled on the network

This design of the settings transaction family supports two authorization options, either a single authorized key which can make changes or multiple authorized keys. In the case of multiple keys, a percentage of votes signed by the keys is required before a change is made.

## Sawtooth and Ethereum Compatibility

The primary goal of the Sawtooth-Ethereum integration project, affectionately dubbed “Seth”, is to add support for running Ethereum Virtual Machine smart contracts to the Hyperledger Sawtooth platform. In order to make this possible, the Hyperledger Sawtooth project worked with the Hyperledger Burrow project to integrate their EVM implementation, the Burrow EVM, into the Hyperledger Sawtooth platform.

The secondary goal of Seth is to make it easy to port existing EVM smart contracts, and the DApps that depend on them, to Sawtooth. This has been largely accomplished by replicating the Ethereum JSON RPC API. This also enables developers to leverage existing Ethereum contracts for deployment on Sawtooth. It is important to note that Seth is not an Ethereum node. It is designed to interpret EVM byte code especially in the context of an enterprise deployment. There are some design decisions that differ between Sawtooth and Ethereum. Where possible we made the couplings transparent to the end user and in a few cases we gracefully limit conflicting APIs.

## Conclusion

Sawtooth is a blockchain platform designed for enterprise usage. Sawtooth builds upon the learnings of platforms before it such as Bitcoin and Ethereum and is designed with a focus on modularity and extensibility. Sawtooth is unique from existing blockchain platforms in that it allows developers to program their smart contracts in a variety of traditional programming languages. Future Sawtooth developments will focus on increasing the throughput and privacy of transactions.