01/01/2022

eVESTX

White Paper v0.1

t.me/eVESTXCOIN

fb.com/eVESTXCOIN

discord.gg/HeZrgc2cJh

VestXHybrid - Blockchain Technology

Table of Contents

Introduction

eVESTX is the official cryptocurrency of the VestXHybrid blockchain ecosystem. It's a revolutionary useful cryptocurrency for social, economic and utility purposes; designed to be fast, powerful, and limitlessly scalable. When it comes to blockchain technology, scalability is one of the biggest challenges. As networks grow, they often face limitations in terms of transaction speed and confirmation time. For this reason, we designed our blockchain to combat these limitations without compromising security and decentralization. Bitcoin, Ethereum, and many other projects suffer from scalability and speed issues. Using a method known as Proof of History (PoH) and Proof of Stake (Pos), our blockchain is capable of handling thousands of transactions per second and rewarding all the members that allowed the mining and validation of that block to be possible, given that from the smallest to the largest member, everyone wins.

Network Design

The eVESTX blockchain ecosystem has two chains, the main chain, and the side chain, both with different purposes, but following the same consensus logic. In the main chain, at any time a system node is designated as a leader to generate a History Proof sequence, providing the global network with reading consistency and a verifiable time passage. The Leader sequences user messages and orders them so they can be efficiently processed by other nodes in the system, maximizing throughput. The Side Chain runs side by side with the Main Chain, which, unlike the Main Chain, has EVM compatibility. EVM compatibility was chosen because the first practical and widely used smart contract platform is Ethereum. The implementation should leave room for the sidechain to keep up with more Ethereum updates. Staking-based consensus is greener and leaves a more flexible option for community governance.

Additional Features

Low fees

One of the most discussed factors these days is the high transaction fees. We've managed to create an environment where transaction fees are super low and with fixed amounts that don't vary.

Scalable

Integrate once and never worry about scaling again. We ensure composition in ecosystem projects, maintaining a single global state as the network grows. Never deal with fragmented chain systems.

High Latency

With a speed never seen before, we promise to bring you the best that a blockchain ecosystem can bring with the speed and latency greater than a centralized payments network can bring.

Enhanced Security

The biggest problem for coins is the likelihood of 51 percent attacks once the master nodes are breached. However, since it uses a POS and POH architecture for consensus, it dissuades validators from affecting the value of the coin adversely

Smart Contract

Smart contracts are a widespread form of transactions. These are programs that run on each node and modify the state. Thus, digital contracts are automatically executed.

Asic Attack Resistance

For ASIC attacks during Partitions, the Rate at which bonds are unstaked is non-linear, and for networks with large partitions the rate is orders of magnitude slower then expected gains from an ASIC attack.

Consensus

Proof of History

Introduction

Proof of History is a sequence of computations that can provide a way to cryptographically verify the passage of time between two events. It uses a cryptographically secure function written so that output cannot be predicted from the input, and must be completely executed to generate the output.

The function is run in a sequence on a single core, its previous output as the current input, periodically recording the current output, and how many times it's been called. The output can then be re-computed and verified by external computers in parallel by checking each sequence segment on a separate core.

Data can be timestamped into this sequence by appending the data (or a the hash of some data) into the state of the function. The recording of the state, index, and data as it was appended into the sequences provides a timestamp that can guarantee that the data was created sometime before the next hash was generated in the sequence. This design also supports horizontal scaling as multiple generators can synchronize amongst each other by mixing their state into each others' sequences. Horizontal scaling is discussed in depth in the section.

Timestamp for Events

This sequence of hashes can also be used to record that some piece of data was created before a particular hash index was generated. Using a 'combine' function to combine the piece of data with the current hash at the current index. The data can simply be a cryptographically unique hash of arbitrary event data. The combine function can be a simple append of data or any operation that is collision-resistant. The next generated hash represents a timestamp of the data because it could have only been generated after that specific piece of data was inserted. The index and the sha256 are recorded as part of the sequence output. So anyone verifying this sequence can then recreate this change to the sequence. The verifying can still be done in parallel and it's discussed in Section. Because the initial process is still sequential, we can then tell that things entered into the sequence must have occurred sometime before the future hashed value was computed. Inserting the data into the sequence of hashes results in a change to all subsequent values in the sequence. As long as the hash function used is collision-resistant, and the data was appended, it should be computationally impossible to pre-compute any future sequences based on prior knowledge of what data will be integrated into the sequence. The data that is mixed into the sequence can be the raw data itself, or just a hash of the data with accompanying metadata. Every node can determine the order in which all events have been inserted and estimate the real-time between the insertions.

Verification

The sequence can be verified correct by a multicore computer in significantly less time than it took to generate it. Given some number of cores, like a modern GPU with 4000 cores, the verifier can split up the sequence of hashes and their indexes into 4000 slices, and in parallel make sure that each slice is correct from the starting hash to the last hash in the slice. Each core can verify each slice of the sequence in

parallel. Since all input strings are recorded into the output, with the counter and state that they are appended to, the verifiers can replicate each slice in parallel.

Scaling

It's possible to synchronize multiple Proof of History generators by mixing the sequence state from each generator to each other generator and thus achieve horizontal scaling of the Proof of History generator. This scaling is done without sharding. The output of both generators is necessary to reconstruct the full order of events in the system. Given generators A and B, A receives a data packet from B (hash1b), which contains the last state from Generator B, and the last state generator B observed from Generator A. The next state hash in Generator A then depends on the state from Generator B, so we can derive that hash1b happened sometime before hash3a. This property can be transitive, so if three generators are synchronized through a single common generator A ↔️ B ↔️ C, we can trace the dependency between A and C even though they were not synchronized directly. By periodically synchronizing the generators, each generator can then handle a portion of external traffic, thus the overall system can handle a larger amount of events to track at the cost of true-time accuracy due to network latencies between the generators. A global order can still be achieved by picking some deterministic function to order any events that are within the synchronization window, such as by the value of the hash itself. The generated hashes are mixed into each stream. The synchronization is transitive. A ↔️ B ↔️ C There is a provable order of events between A and C through B. Scaling in this way comes at the cost of availability. 10 × 1 Gbps connections with availability of 0.999 would have 0.99910 = 0.99 availability.

Consistency

Users are expected to be able to enforce consistency of the generated sequence and make it resistant to attacks by inserting the last observed output of the sequence they consider valid into their input. A malicious PoH generator could produce a second hidden sequence with the events in reverse order, if it has access to all the events at once, or can generate a faster-hidden sequence.

To prevent this attack, each client-generated Event should contain within itself the latest hash that the client observed from what it considers to be a valid sequence. So when a client creates the ”Event1” data, they should append the last hash they have observed. When the sequence is published, Event3 would be referencing hash30a, and if it’s not in the sequence before this Event, the consumers of the sequence know that it’s an invalid sequence.

The partial reordering attack would then be limited to the number of hashes produced while the client has observed an event and when the event was entered. Clients should then be able to write software that does not assume the order is correct for the short period of hashes between the last observed and inserted hash. To prevent a malicious PoH generator from rewriting the client Event hashes, the clients can submit a signature of the event data and the last observed hash instead of just the data.

Overhead

4000 hashes per second would generate an additional 160 kilobytes of data and would require access to a GPU with 4000 cores and roughly 0.25-0.75 milliseconds to verify.

Proof Of Stake

Introduction

This specific instance of Proof of Stake is designed for quick confirmation of the current sequence produced by the Proof of History generator, for voting and selecting the next Proof of History generator, and for punishing any misbehaving validators. This algorithm depends on messages eventually arriving at all participating nodes within a certain timeout.

Terminology

Bonds are equivalent to a capital expense in Proof of Work. A miner buys hardware and electricity and commits it to a single branch in a Proof of Work blockchain. A bond is a coin that the validator commits as collateral while they are validating transactions. The proposed solution to the nothing at stake problem in Proof of Stake systems. When proof of voting for a different branch is published, that branch can destroy the validator’s bond. This is an economic incentive designed to discourage validators from confirming multiple branches. A supermajority is 2/3rds of the validators weighted by their bonds. A supermajority vote indicates that the network has reached consensus, and at least 1/3rd of the network would have had to vote maliciously for this branch to be invalid. This would put the economic cost of an attack at 1/3rd of the market cap of the coin.

Bonding

A bonding transaction takes an amount of currency and moves it to an account under the user's identity. Coins in the securities account cannot be spent and must remain in the account until the user removes them. Users can only remove obsolete coins that have expired.

Voting

It is anticipated that the History Proof generator will be able to publish a State subscription within a pre-defined period. Each linked identity must confirm this signature by publishing its own signed state signature. Voting is a simple yes without a no.

Unbonding

The N number of absent votes marks the coins as obsolete and are no longer eligible for voting. The user can issue an unlink transaction to remove them. N is a dynamic value based on the proportion of stale and active votes. N increases as the number of stale votes increases. In a large network partition event, this allows the larger branch to recover faster than the smaller branch.

Elections

The election for a new PoH generator occurs when the failure of the PoH generator is detected. The validator with the highest voting power or highest public key address if there is a tie is chosen as the new PoH generator. If the new leader fails before supermajority commits are available, the next highest validator is selected and a new set of commits is required. To switch votes, a validator needs to vote for a higher PoH sequence counter, and the new vote needs to contain the votes they want to switch. Once a PoH generator is established, a Secondary can be elected to take over transactional processing duties.

Proof Of Replication

Introduction

Filecoin proposed a version of the Proof of Replication. The purpose of this release is to have fast and streaming Proof of Replication checks, which are enabled by keeping track of time in a sequence-generated Proof of History. Replication is not used as a consensus algorithm, but it is a useful tool to account for the cost of storing blockchain history or state on high availability.

Algorithm

CBC encryption encrypts each block of data in sequence, using the previously encrypted block to XOR the input data. Each replication identity generates a key by signing a hash that was generated through a Proof of History string. This links the key to an identity replicator and a specific History Proof string. Only specific hashes can be selected. The dataset is fully encrypted block by block. Then, to generate the proof, the key is used to seed a pseudo-random number generator that selects 32 random bytes from each block. A Merkle hash is calculated with the selected PoH hash attached to each slice. The root is published, along with the key and selected hash that was generated. The replication node is required to publish another proof in N hashes as they are generated by the Proof of History generator, where N is approximately the time it takes to encrypt the data. The Proof of History generator will publish specific hashes for Proof of Replication at predefined periods. The replicator node must select the next published hash to generate the proof. Again, the hash is signed and random slices are selected from the blocks to create the Merkle root. After a period of N proofs, the data is re-encrypted with a new CBC key

Verification

With N cores, each core can stream encryption for each identity. The total space required is 2blocks of Scores since the previous encrypted block is necessary to generate the next one. Each core can then be used to generate all the proofs that were derived from the current encrypted block. Total time to verify proofs is expected to be equal to the time it takes to encrypt. The proofs themselves consume a few random bytes from the block, so the amount of data to hash is significantly lower than the encrypted block size. The number of replication identities that can be verified at the same time is equal to the number of available cores. Modern GPUs have 3500+ cores available to them, albeit at 1/2-1/3 the clock speed of a CPU.

Key Rotation

Without key rotation, the same encrypted replication can generate cheap proofs for multiple Proof of History sequences. Keys are rotated periodically and each replication is re-encrypted with a new key that is tied to a unique Proof of History sequence. Rotation needs to be slow enough that it's practical to verify replication proofs on GPU hardware, which is slower per core than CPUs.

Hash Selection

The Proof of History generator publishes a hash to be used by the entire network to encrypt Proofs of Replication and to use as pseudo-random number generator for byte selection in quick tests. The hash is published to a periodic counter that is approximately equal to 1/2 the time it takes to encrypt the dataset. Each replication identity must use the same hash and use the signed result of the hash as the seed for the byte selection or encryption key. The time each replicator must

provide the proof must be less than the encryption time. Otherwise, the replicator can pass the encryption and delete it for each probe. A malicious generator can inject data into the string before this hash to generate a specific hash.

Proof Validation

The Proof of History node is not expected to validate the submitted Proof of Replication proofs. It is expected to keep track of the number of pending and verified proofs submitted by the replicator's identity. A proof is expected to be verified when the replicator can sign the proof by a supermajority of the validators in the network. The verifications are collected by the replicator via the p2p gossip network and submitted as one packet that contains a supermajority of the validators in the network. This packet verifies all the proofs before a specific hash generated by the Proof of History sequence and can contain multiple replicator identities at once.

Election Triggers

Introduction

PoH generators are designed with an identity that signs the generated sequence. A fork can only occur in case the PoH generator's identity has been compromised. A fork is detected because two different historical records have been published on the same PoH identity.

Runtime Exceptions

A hardware failure or a bug, or an intentional error in the PoH generator could cause it to generate an invalid state and publish a signature of the state that does not match the local validator's result. Validators will

A hardware failure or a bug, or an intentional error in the PoH generator could cause it to generate an invalid state and publish a signature of the state that does not match the local validator's result. Validators will

Network Timeouts

A network timeout would trigger an election.

Slashing

Slashing occurs when a validator votes two separate sequences. A proof of the malicious vote will remove the bonded coins from circulation and add them to the mining pool. A vote that includes a previous vote on a contending sequence is not eligible as proof of malicious voting. Instead of slashing the bonds, this vote removes the currently cast vote on the contending sequence. Slashing also occurs if a vote is cast for an invalid hash generated by the PoH generator. The generator is expected to randomly generate an invalid state, which would trigger a fallback to Secondary.

Secondary Elections

Secondary and lower-ranked Proof of History generators can be proposed and approved. A proposal is cast on the primary generator's sequence. The proposal contains a timeout, if the motion is approved by a supermajority of the vote before the timeout, the Secondary is considered elected, and will take over duties as scheduled. Primary can do a soft handover to Secondary by inserting a message into the generated sequence indicating that a handover will occur, or inserting an invalid state and forcing the network to fall back to Secondary. If a Secondary is elected, and the primary fails, the Secondary will be considered as the first fallback during an election.

Availability

CAP systems that deal with partitions have to pick Consistency or Availability. Our approach eventually picks Availability, but because we have an objective measure of time, Consistency is picked with reasonable human timeouts. Proof of Stake verifiers locks up some amount of coin in a “stake”, which allows them to vote for a particular set of transactions. Locking up a coin is a transaction that is entered into a PoH stream, just like any other transaction. To vote, a PoS verifier has to sign the hash of the state, as it was computed after processing all the transactions to a specific position in the PoH ledger. This vote is also entered as a transaction into the PoH stream. Looking at the PoH ledger, we can then infer how much time passed between each vote and if a partition occurs, for how long each verifier has been unavailable. To deal with partitions with reasonable human timeframes, we propose a dynamic approach to “unstake” unavailable verifiers. When the number of verifiers is high and above 2/3, the “unstacking” process can be fast. The number of hashes that must be generated into the ledger is low before the unavailable verifier's stake is fully unstacked and they are no longer counted for consensus. When the number of verifiers is below 2/3rds but above 1/2, the unstacking timer is slower, requiring a larger number of hashes to be generated before the missing verifiers are unstacked. In a large partition, like a partition that is missing 1/2 or more of the verifiers, the unstacking process is very very slow. Transactions can still be entered into the stream, and verifiers can still vote, but full 2/3rds consensus will not be achieved until a very large amount of hashes have been generated and the unavailable verifiers have been unstacked. The difference in time for a network to regain liveness allows us as customers of the network human timeframes to pick a partition that we want to continue using.

Recovery

In the system we propose, the ledger can be fully recovered from any failure. That means anyone in the world can pick any random spot in the ledger and create a valid fork by appending newly generated hashes and transactions. If all the verifiers are missing from this fork, it would take a very very long time for any additional bonds to become valid and for this branch to achieve 2/3rds supermajority consensus. So full recovery with zero available validators would require a very large amount of hashes to be appended to the ledger, and only after all the unavailable validators have been unstacked will any new bonds be able to validate the ledger.

Attacks

Reversal

Generating a reverse order would require an attacker to start the malicious sequence after the second event. This delay should allow any non-malicious peer-to-peer nodes to communicate about the original order.

Tragedy of Commons

The PoS verifiers simply confirm the state hash generated by the PoH generator. There is an economic incentive for them to do no work and simply approve every generated state hash. To avoid this condition, the PoH generator should inject an invalid hash at a random interval. Any voters for this hash should be slashed. When the hash is generated, the network should immediately promote the Secondary elected PoH

generator. Each verifier is required to respond within a small timeout - 500ms for example. The timeout should be set low enough that a malicious verifier has a low probability of observing another verifier's vote and getting their votes into the stream fast enough.

Collusion with the PoH generator

A verifier that is colluding with the PoH generator would know in advance when the invalid hash is going to be produced and not vote for it. This scenario is no different than the PoH identity having a larger verifier stake. The PoH generator still has to do all the work to produce the state hash.

Censorship

Censorship or denial of service could occur when 1/3rd of the bond holders refuse to validate any sequences with new bonds. The protocol can defend against this form of attack by dynamically adjusting how fast bonds become stale. In the event of a denial of service, the larger partition will be designed to fork and censor the Byzantine bond holders. The larger network will recover as the Byzantine bonds become stale with time. The smaller Byzantine partition would not be able to move forward for a longer time. The algorithm would work as follows. A majority of the network would elect a new Leader. The Leader would then censor the Byzantine bond holders from participating. Proof of History generator would have to continue generating a sequence, to prove the passage of time, until enough Byzantine bonds have become stale so the bigger network has a supermajority. The rate at which bonds become stale would be dynamically based on what percentage of bonds are active. So the Byzantine minority fork of the network would have to wait much longer than the majority fork to recover a supermajority. Once a supermajority has been established, slashing could be used to permanently punish the Byzantine bondholders.

Long Range Attacks

PoH provides a natural defense against long-range attacks. Recovering the ledger from any point in the past would require the attacker to overtake the valid ledger in time by outpacing the speed of the PoH generator. The consensus protocol provides a second layer of defense, as any attack would have to take longer than the time it takes to unstake all the valid validators. It also creates an availability “gap” in the history of the ledger. When comparing two ledgers of the same height, the one with the smallest maximum partition can be objectively considered valid.

ASIC Attacks

Two opportunities for ASIC attacks exist in this protocol - during partition and cheating timeouts in Finality. For ASIC attacks during Partitions, the Rate at which bonds are unstacked is non-linear, and for networks, with large partitions, the rate is orders of magnitude slower than expected gains from an ASIC attack. For ASIC attacks during Finality, the vulnerability allows for byzantine validators who have a bonded stake to wait for confirmations from other nodes and inject their votes with a collaborating PoH generator. The PoH generator can then use its faster ASIC to generate 500ms worth of hashes in less time, and allow for network communication between the PoH generator and the collaborating nodes. But, if the PoH generator is also byzantine, there is no reason why the byzantine generator wouldn’t have communicated the exact count when they expect to insert the failure. This scenario is no different than a PoH generator and all the collaborators sharing the same identity vs. having a single combined stake and only using 1 set of hardware.

Partial Erasure

A replicator node could attempt to partially erase some of the data to avoid storing the entire state. The number of proofs and the randomness of the seed should make this attack difficult. For example, a user storing 1 terabyte of data erases a single byte from each 1-megabyte block. A single proof that samples 1 byte out of every megabyte would have a likelihood of collision with any erased byte 1−(1−1/1,000,0000)1,000,000 = 0.63. After 5 proofs the likelihood is 0.99.

Denial of Service

The cost of adding replicator identity is expected to be equal to the cost of storage. The cost of adding extra computational capacity to verify all the replicator identities is expected to be equal to the cost of a CPU or GPU core per replication identity. This creates an opportunity for a denial of service attack on the network by creating a large number of valid replicator identities. To limit this attack, the consensus protocol chosen for the network can select a replication target, and award the replication proofs that meet the desired characteristics, like availability on the network, bandwidth, geolocation, and more.

System Architecture

Components

Leader

The Leader is an elected Proof of History generator. It consumes arbitrary user transactions and outputs a Proof of History sequence of all the transactions that guarantee a unique global order in the system. After each batch of transactions, the Leader outputs a signature of the state that is the result of running the transactions in that order. This signature is signed with the identity of the Leader.

State

The state is a naive hash table indexed by the user's address. Each cell contains the full user's address and the memory required for this computation.

Verifier

The Verifier nodes replicate and provide high availability of the blockchain state. The replication target is selected by the consensus algorithm, and the validators in the consensus algorithm select and vote the Proof of Replication nodes they approve of based on off-chain defined criteria. The network could be configured with a minimum Proof of Stake bond size, and a requirement for a single replicator identity per bond.

Validators

These nodes are consuming bandwidth from Verifiers. They are virtual nodes and can run on the same machines as the Verifiers or the Leader, or on separate machines that are specific to the consensus algorithm configured for this network.

Smart Contract

Smart contracts are a generalized form of transactions. These are programs that run on each node and modify the state. This design leverages extended Berkeley Packet Filter bytecode, which is fast and easy to analyze, and JIT bytecode as the smart contract's language. One of its main advantages is a zero-cost Foreign Function Interface. Intrinsics, or functions that are implemented on the platform directly, are callable by programs. Calling the intrinsics suspends that program and schedules the intrinsic on a high-performance server. Intrinsics are batched together to execute in parallel on the GPU. Each program is suspended until the batch execution of the intrinsics is complete. An example intrinsic is ECDSA verification. Batching these calls to execute on the GPU can increase throughput thousands of times. This trampoline requires no native operating system thread context switches since the BPF bytecode has a well-defined context for all the memory that it is using.

Limits

Generator

The leader is expected to be able to take incoming user packets, order them in the most efficient way possible, and sequence them into a

Proof of History sequence that is published to downstream Verifiers. Efficiency is based on memory access patterns of the transactions, so the transactions are ordered to minimize faults and maximize prefetching. The minimal payload that can be supported would be 1 destination account with a minimum size of 176 bytes. The Proof of History sequence packet contains the current hash, counter, and the hash of all the new messages added to the PoH sequence and the state signature after processing all the messages. This packet is sent once every N message is broadcast. The minimum size of the output packet is 132 bytes. On a 1gbps network connection the maximum number of transactions possible is 1 gigabit per second / 176 bytes = 710k tps max. Some loss (1 – 4%) is expected due to Ethernet framing. The spare capacity over the target amount for the network can be used to increase availability by coding the output with Reed-Solomon codes and striping it to the available downstream Verifiers.

Computational

Each transaction requires a digest verification. This operation does not use any memory outside of the transaction message itself and can be parallelized independently. Thus throughput is expected to be limited by the number of cores available on the system.

Memory

A naive implementation of the state as a 50% full hashtable with 32-byte entries for each account, would theoretically fit 10 billion accounts into 640GB. Steady-state random access to this table is measured at 1.1 107 writes or reads per second. Based on 2 reads and two writes per transaction, memory throughput can handle 2.75m transactions per second.

Main Chain

In the main chain, at any given time a system node is designated as Leader to generate a Proof of History sequence, providing the network global read consistency and a verifiable passage of time. The Leader sequences user messages and orders them such that they can be efficiently processed by other nodes in the system, maximizing throughput. It executes the transactions on the current state that is stored in RAM and publishes the transactions and a signature of the final state to the replications nodes called Verifiers. Verifiers execute the same transactions on their copies of the state and publish their computed signatures of the state as confirmations. The published confirmations serve as votes for the consensus algorithm.

Side Chain

The Side Chain runs side by side with the Main Chain, which, unlike the Main Chain, has EVM compatibility. EVM compatibility was chosen because the first practical and widely used smart contract platform is Ethereum. To take advantage of the relatively mature apps and community, we decided to opt to be compatible with the existing Ethereum mainnet. This means that most dApps, ecosystem components, and tools will work in the sidechain and require zero or minimal changes. The Side Chain node will require similar (or slightly higher) hardware specs and skills to run and operate. The implementation should leave room for the side chain to keep up with more Ethereum updates. Staking-based consensus is greener and leaves a more flexible option for community governance. This consensus, along with the proof-of-history consensus, is expected to enable better network performance blockchain system, i.e. faster lock time and greater velocity and capacity in the transaction, as well as better duration between events and message ordering. With native

support for cross-chain communication between the two blockchains, the communication protocol must be bi-directional, decentralized, and trustless. It will focus on moving digital assets between the two blockchains. The protocol should take care of the bare minimum of other items stored in the state of blockchains, with just a few exceptions.

Coin Name	eVESTX
Ticker:	eVESTX
Network:	Main
Consensus:	PoH/PoS
Block Reward:	100 eVESTX
Block Time:	0.350 Seconds
Stake Maturation:	200 Blocks

References

[1] Liskov, Practical use of Clocks

https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.1089.5025&rep= rep1&type=pdf

[2] Google Spanner TrueTime consistency

https://cloud.google.com/spanner/docs/true-time-external-consistency

[3] Solving Agreement with Ordering Oracles

http://www.inf.usi.ch/faculty/pedone/Paper/2002/2002EDCCb.pdf

[4] Tendermint: Consensus without Mining

http://tendermint.com/static/docs/tendermint.pdf

[5] Dr. Gavin Wood, Ethereum Virtual Machine

https://ethereum.github.io/yellowpaper/paper.pdf

[6] Hedera: A Governing Council & Public Hashgraph Network

https://s3.amazonaws.com/hedera-hashgraph/hh-whitepaper-v1.0-180313.pdf

[7] Filecoin, proof of replication

https://filecoin.io/proof-of-replication.pdf

[8] Slasher, A punitive Proof of Stake algorithm

https://blog.ethereum.org/2014/01/15/slasher-a-punitive-proof-of-stake-algorithm/

[9] BitShares Delegated Proof of Stake

https://github.com/BitShares/bitshares/wiki/Delegated-Proof-of-Stake

[10] An Efficient Elliptic Curve Cryptography Signature Server With GPU Acceleration

https://ieeexplore.ieee.org/document/7555336/

[11] Casper the Friendly Finality Gadget

https://arxiv.org/pdf/1710.09437.pdf