ECTO
A Novel Proof-of-Stake Cryptocurrency
6.824 Final Project Writeup

Henry Aspegren, James Lovejoy
{*henrya*, *jlovejoy*} @ mit.edu

May 11, 2018

1 Introduction

Currently the most popular cryptocurrencies use Proof-of-Work. However this consensus mechanism consumes large amounts of energy: Bitcoin alone uses more energy than the entire country of Cuba [1]. Proof-of-Stake has been proposed as an energy-efficient alternative. Much like Proof-of-Work, in Proof-of-Stake, a single participant known as the leader, staker or block proposer, is selected to add a block to the chain. Leaders are selected probabilistically in proportion to how much of the currency the leader has "staked". Ideally this "stake" constitutes a bond that the leader will lose if she "misbehaves." Realizing Proof-of-Stake in practice has been challenging. Indeed at the time of this writing, no Proof-of-Stake currency has ever achieved widespread adoption[1].

One of the main reasons for this is called the *Nothing-at-Stake* problem. In Proof-of-Work, equivocating by proposing multiple blocks to extend competing chains requires the consumption of additional energy and computational resources with real-world cost linear with the number of chains being extended. So to extend 5 competing chains simultaneously requires 5 times the real-world cost. Furthermore, any blocks that do not end up in the canonical chain will not receive the associated rewards and the cost incurred to produce them is permanently lost. This creates a strong incentive for a rational actor to mine on only one chain: the chain that the network will eventually reach consensus on, where their mining rewards will be honored. However, in existing Proof-of-Stake designs, the real-world computational cost of producing a block is deliberately negligible. As a result there is no additional cost associated with producing 5 competing chains. In fact it may be completely rational for a block proposer to extend as many possible system states simultaneously to ensure that they will put a block in whatever chain the network eventually accepts.

Because of the *Nothing-at-Stake* problem, current Proof-of-Stake systems provide more limited guarantees of eventual consistency than Proof-of-Work. Any block proposer can trivially show equivalently valid system states to different clients or in the case of a staker (or group of stakers) at any point having a majority of the stake in the system, arbitrarily re-write long-established history. Existing Proof-of-Stake implementations have solved this problem by using a developer-operated coordinator (sometimes distributed) that endorses a canonical chain for clients to follow. In this work we propose a novel Proof-of-Stake consensus protocol that disincentivizes equivocation and does not require a centralized coordinator to operate.

2 Related Work

2.1 PeerCoin

PeerCoin [2] was the first attempt at a Proof-of-Stake cryptocurrency. ECTO uses a consensus mechanism inspired by PeerCoin's original design. However because of the *Nothing-at-Stake* problem PeerCoin uses a centralized "check-pointing" scheme. This is a major security concern since a corrupted check-pointing

[1] As defined by being among the top 5 largest currencies by market cap

system is a single point of failure for the entire system and allows the developers to censor transactions or rewrite history.

2.2 Algorand

Algorand [3] is another alternative to Proof-of-Work. However Algorand, as described by its authors, is not Proof-of-Stake and has a different security argument entirely. The security of Algorand depends on a Byzantine Agreement protocol that is run among a randomly selected set of participants. In Algorand, "stake" can be used to determine the set of nodes in the system and their relative voting weights in the BA protocol.

3 ECTO Consensus

In any blockchain-based system, the consensus algorithm must determine the following:

1. Who is allowed to propose a given block?
2. Given multiple competing chains, which one should be followed?

3.1 Joining the Stake Pool

To participate in the consensus algorithm, the participant or *staker* must create a special transaction that *stakes* coins. This *stake* constitutes a bond to the network that could be lost if the *staker* equivocates and broadcasts a block that extends multiple competing chains. The staking transaction consists of following:

$$tx = \{value, publicKey, R\_point\}$$

The *value* is the total number of coins staked, the R_Point is a cryptographic commitment and the $publicKey$ is the public key of the potential future spender of the output. Later in the paper we will use the R-point commitment to guard against equivocation. Once this transaction has been included in a block, the output's age will be initialized to 1 increase by 1 for every block after the inclusion block until the output is used to propose a block when its age resets to 1.

This process contrasts to Proof-of-Work because it makes the system *permissioned*. A potential staker needs to first have coins within the system and the ability to get their transactions included in the chain. Proof-of-Work only requires computational resources to participate and is thus *permissionless*.

3.2 Selecting a Block Proposer

In Proof-of-Work, a miner (block proposer) can be anyone who can find a nonce such that

$$SHA256(block\_id||nonce) < target\_difficulty$$

where || is bitwise concatenation. The $target\_difficulty$ is regularly adjusted by the network as miners come and go to keep the rate of block generation stable over time. The target is increased as average block time increases to make the space of possible solutions larger, and decreased in the opposite case.

In ECTO , a block proposer is selected according to randomness associated with their *stake* rather than their ability to brute-force a valid nonce. Instead a block proposer must meet the following criteria:

$$SHA256(randomness||output\_id||block\_timestamp) < target\_difficulty * (value \times age)$$

where *value* is the coin value of the *stake* output and *age* is the number of blocks that have passed since the *stake* transaction was created or previously used to propose a block. Note that there is no nonce involved and that the computation required is finite: it takes exactly one $SHA256$ hash per staked output per second to participate in the consensus protocol. By comparison, commercial Bitcoin miners easily churn through over 4 Billion $SHA256$ hashes per second, consuming orders of magnitude more energy. Additionally note that the older and more valuable the *stake* transaction, the easier it is to propose a block. The calculation of $target\_difficulty$ in ECTO is identical to Proof-of-Work.

Selecting the source of *randomness* requires careful thought for the system to be resistant to *stake-grinding*. If the randomness source is too easy to malleate for the staker, they could permute the randomness many times to increase their chance of being selected as a block proposer. In that case the system would descend into Proof-of-Work and become a competition over who can malleate and hash the fastest. In ECTO we use the *block_id* from $max(1, n - 1000)$ blocks previously as the source of randomness with the assumption that with a well-distributed stake pool it will be difficult for a staker to have regular influence over a specific *block_id* from multiple days in the past.

3.3 Preventing Equivocation

Each block is required to be cryptographically signed by the owner of the private key associated with the public key of the output being staked. By using Committed R-point Signatures [6] we are also able to use the signature to disincentivize equivocation. In a regular Schnorr signature we use a common *generator point* G and a private scalar a to form a public/private key pair. The public key A is generated as follows:

$$A = aG$$

Given A and G it is assumed impossible to generate the private key a due to the hardness of the *discrete log problem*. To sign a message m, one first generates another random private scalar r and calculates point R.

$$R = rG$$

followed by:

$$s = r - SHA256(m||R)a$$

The signature then consists of (s, R) and can be verified to prove knowledge of the private key a using the public key as follows:

$$\begin{aligned} sG &= R - SHA256(m||R)A \\ &= rG - SHA256(m||R)aG \end{aligned}$$

Since we assume it is impossible to calculate a from $aG = A$ and r from $rG = R$ then the signing party must know both a and r. It is important that the signer uses a unique and non-deterministic r for every message they sign or given two signatures s_1, s_2 and two messages m_1, m_2 signed with the same r, one can compute both a and r by solving the two equations:

$$\begin{aligned} h_1 &= SHA256(m_1||R) \\ h_2 &= SHA256(m_2||R) \end{aligned}$$

$$\begin{aligned} s_1 &= r - h_1 a \\ s_2 &= r - h_2 a \end{aligned}$$

$$\begin{aligned} a &= (h_1 - h_2)^{-1} * (s_2 - s_1) \\ r &= s_2 + h_2 a \end{aligned}$$

Similarly to in Discreet Log Contracts, we exploit this property of Schnorr signatures to prevent equivocation. If a staker signs and broadcasts two competing blocks using the same R-point they reveal their private key to the network effectively making their coins "spend-to-anyone" outputs. Any peer who notices equivocation would be able to claim the offenders coins as a punishment. Once a block has been broadcast using a pre-committed R-point, the coins cannot be safely staked again until a new R-point has been committed to by either specifying it in the block where the previous R-point was used or by spending the output

Figure 1: Extending Multiple Chains

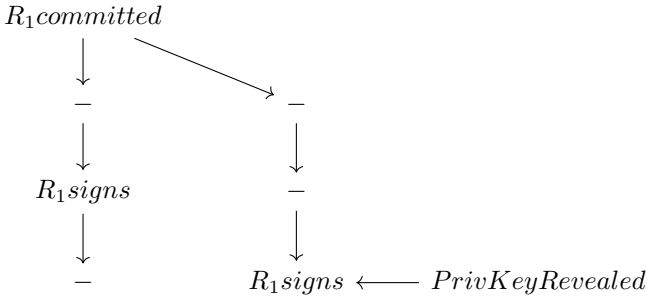
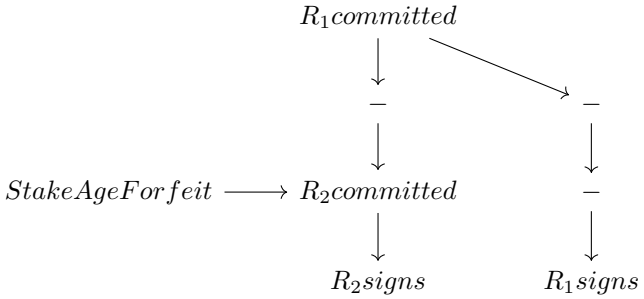


Figure 2: Extending A Non-Canoncial Chain



and specifying a new R-point in a new output as if entering the stake pool for the first time. This scheme requires that stakers pick one branch of a fork to stake on.

In 1 the staker tried to equivocate but their private key is revealed once they bet on the second branch. Anyone who notices this can take their staked coins.

In 2 the staker does equivocate by endorsing two competing chains. However to avoid losing his coins he must first re-stake using R_2 to replace R_1. In doing so he forfeits his stake-age for that output hence decreasing the probability he is selected as the next block proposer. Therefore it is highly unlikely that the output with R_2 committed will be selected before the chain where R_1 is used to sign has overtaken the alternative branch.

3.4 Reaching Consensus

In Proof-of-Work, the network achieves consensus on a canonical chain with the most 'total work'. 'Total work' in this context is the cumulative computational resources that produced the chain calculated as follows:

$$total\_work[n] = \sum_{height=1}^{n} UINT256\_MAX - target\_difficulty[height]$$

In ECTO , the network also follows with the chain with the most 'total work', which is calculated in an identical manner. However in ECTO this implicitly reflects the cumulative stake-age consumed to produce the chain since each target is modulated according to the stake-age being consumed. In other words the consensus algorithm follows the chain that was created using the oldest and most valuable outputs cumulatively.

4 Implementation

To implement ECTO we used CryptoKernel [4], a software package being developed by the Digital Currency Initiative for rapid prototyping cryptocurrencies [2]. CryptoKernel provides out-of-the-box networking, storage and replicated state machine code while allowing developers to implement their own consensus modules and other features. We implemented ECTO by creating a new consensus module for CryptoKernel.

We have created a genesis block for ECTO and are currently running several nodes that are all participating in the consensus process. Anyone can join the network and start using the protocol, using our reference implementation on Github [5]. To join the network simply download the code in the repository and follow the instructions in the README.

Because ECTO selects block proposers based on the amount of stake a user has, we have had to think carefully about how distribute the initial coins. In our prototype there are no block rewards and all of the coins were initially given to the Authors. However we have implemented a faucet where you can post your public key and we will send you coins. Our goal is to distribute the vast majority of the coins to anyone who would like them. The faucet can be accessed by emailing the paper authors with public keys.

In a real system it would be necessary to fairly and widely distribute the coins in the setup phase to ensure that no individual or cabal is able to acquire a majority of the stake. If the developers cannot be trusted to do this manually, having a period of Proof-of-Work to initialize the system and emit all the coins, then switching to Proof-of-Stake has been a popular solution.

5 Analysis

WARNING the authors accept no responsibility for the security of ECTO , and it may be insecure and you may lose all of your money

Analyzing the security of cryptocurrencies is extremely difficult. A full analysis considers both the security of the underlying protocol as well as the game-theoretic implications of rational participants. Actual implementations almost always suffer from unforeseen vulnerabilities. A rigorous analysis is out of the scope of this paper. However we will present a sketch of why we think ECTO might be secure.

5.1 Security Argument (Sketch)

The security of ECTO rests on the assumption that participants will choose to extend the chain with the most total work. Because of our commitment scheme a participant can only safely chose to use a given staked output once. If participants equivocate and try to re-use the staked output by extending multiple chains, then they could lose their stake entirely. If participants chose to extend some other chain instead of extending the longest chain, the implications are more subtle. The participant will not lose his stake, but rather will lose the ability to propose a block on the canonical chain using that stake. This is sub-optimal since the participant will lose out on any rewards associated with producing a block (in ECTO , these are the transaction fees) and lose their accumulated stake-age. Thus participants have a strong incentive to stake on the fork that the network will ultimately accept, since otherwise their opportunity to stake will be wasted. This is conceptually similar to the argument that a miner in Proof-of-Work will only mine on the longest chain.

As with all Proof-of-Stake systems, an individual or cabal at any point gaining a majority of the stake is fatal. From the block where a majority is possible all subsequent history can later be re-written arbitrarily and with negligible cost. Great care must be taken to construct the initial distribution such that this is unlikely to happen.

6 Conclusion

In the cryptocurrency community, Proof-of-Stake is a hotly debated alternative to energy-intensive Proof-of-Work. However, to the best of our knowledge no one has been able to realize Proof-of-Stake in practice. Furthermore, despite the interest there are few documented protocols and, to the best of our knowledge,

[2] James Lovejoy is the lead developer of CryptoKernel

no open source implementations of Proof-of-Stake. This work presents a clear description of a Proof-of-Stake algorithm that solves the *Nothing-at-Stake* problem and provides an open source implementation that anyone can use. Our hope is that this work will help the community continue to debate and experiment with Proof-of-Work alternatives.

References

[1] http://bigthink.com/strange-maps/bitcoin-consumes-more-energy-than-159-individual-countries

[2] https://peercoin.net/

[3] Gilad, Yossi, et al. "Algorand: Scaling byzantine agreements for cryptocurrencies." Proceedings of the 26th Symposium on Operating Systems Principles. ACM, 2017.

[4] https://dci.mit.edu/cryptokernel/

[5] https://github.com/henryaspegren/6.824-Final-Project

[6] Thaddeus Dryja. "Discreet Log Contracts". https://adiabat.github.io/dlc.pdf