On Stake and Consensus

Andrew Poelstra
2015-03-22
fe81626

Note. This document was significantly revised in March 2015. The old version can be found at `https://download.wpsoftware.net/bitcoin/old-pos.pdf`.

1 Introduction

In 2009, Satoshi Nakamoto introduced the Bitcoin cryptocurrency[Nak09], an online currency system which allowed peer-to-peer transfer of digital tokens. To ensure a consistent view of token ownership, Nakamoto used a public ledger which can be replicated and validated by all network participants. To avoid a single point of failure, this ledger is authenticated using a *dynamic membership multiparty signature (DMMS)*[BCD+14] consisting of an expensive (but cheaply verifiable) computation done on the entire ledger history every "heartbeat".

Unlike a traditional digital signature, there is no notion of "forgeability" for a DMMS. Instead, every DMMS is costly to produce (in Bitcoin, by requiring a large energy expenditure) and rewarded by introduction of new coins on the ledger. Since these coins are only useful if others recognize them, participants are incentivized to extend one "true ledger" rather than attempting to create their own version of history[1].

Because Bitcoin's DMMS is computationally, and therefore thermodynamically[Poe14a], very expensive, alternatives have been proposed which seek to be economically and environmentally more efficient. One popular alternative, *proof-of-stake*, is frequently proposed as a mechanism for a cheap distributed consensus. As argued by the author[Poe14b] in 2014, this is simply not workable, but nonetheless the idea continues to arise in various forms. Meanwhile, the author's argument is commonly asserted on various forums to be "debunked" or "wrong", despite the author having never been made aware of any workable counterexamples or mistakes. This, combined with (correct) accusations that the paper is obtuse and unreadable, demonstrate that its exposition leaves much to be desired. Although this author is not aware of any inaccuracies in his former work, he has taken the opportunity to continue and elaborate his argument more formally.

Further, there has been significant progress in scientific understanding of Bitcoin's consensus[MLJ14, BMC+15] which was not available when the original paper was written.

This paper aims to be an updated version of the author's original paper, which gives more explication on the problem Bitcoin solves, why it makes the design decisions that it does, and why

[1] To ensure that the "true ledger" is visible to all participants, we require a synchronous network such that all (valid) data reaches all participants in some characteristic time λ, and that the network heartbeat time is much larger than λ. Without a synchronous network, distributed consensus is much harder. (There is an oft-cited impossibility result for distributed consensus using deterministic algorithms [FLP85]; this is easy to evade simply by using probabilistic algorithms, and therefore doesn't say much about the difficulty of designing consensus systems. Thanks to Dominic Williams for pointing this out; an earlier version of this document erroneously quoted this result as blocking any distributed consensus in an asynchronous network.)

proof-of-stake and similar mechanisms are fundamentally unable to produce a distributed consensus within Bitcoin's trust model.

2 Distributed Consensus

Before discussing Bitcoin's solution to the distributed consensus problem, it is good to understand what this problem is. A *distributed consensus*, as the term is used in Bitcoin, is a consensus (i.e. global agreement) between many mutually-distrusting parties who lack identities and were not necessarily present at the time of system set up. As explained in the author's original paper,

> For the purposes of cryptocurrency, it is sufficient to achieve distributed consensus on the time-ordering of transactions (and nothing else). This implies consensus on the "first transaction which moves these particular funds", which assures the funds' new owner that the network recognizes them as such.
>
> The reason that this consensus is needed is called the *double-spending problem*. That is, in any decentralized digital currency scheme there is the possibility that a spender might send the same money to two different people, and both spends would appear to be valid. Recipients therefore need a way to be assured that there are no conflicts, or that if there are conflicts, that the network will recognize their version as the correct one. A distributed consensus on transaction ordering achieves this: in the case of conflict, everyone agrees that the transaction which came first is valid while all others are not.
>
> (The other problems with digital currency, e.g. authentication and prevention of forgery, are comparatively easy and can be handled with traditional cryptography.)

It is important to realize that while *distributed* consensus is a hard problem, ordinary consensus is much easier and better studied, and can be solved some trillions of times more efficiently by use of trusted identifiable signing parties. Therefore, cryptocurrencies which compromise by introducing trusted parties, even under limited circumstances, should consider whether their new trust model is one for which consensus is easily achieved by some other mechanism. Readers interested in more efficient, trusted schemes are encouraged to investigate this.

3 Dynamic Membership Multiparty Signatures

Bitcoin's ledger is publically available and the validity of its transactions can be checked by any participant in the network. However, because the ledger is ultimately a historical claim, and cryptography cannot distinguish a true history from a false one, there must be some party to authenticate the ledger, and this party must be trusted not to sign false histories.

The earliest digital cash systems used a single non-anonymous party to sign all transactions[Cha83]. However, this introduced a single point-of-failure for the system as well as giving the signing party (and everyone with legal or physical power to coerce said party) the ability to censor transactions or enable double-spending. Although censorship can be prevented by use of blind signatures (also

described in [Cha83]), both the single-point-of-failure and double-spending problems cannot be. They may be alleviated by using multiple signing parties, but the requirement that these parties be difficult to simultaneously coerce (e.g. by putting them in different legal jurisdictions) conflicts with the requirement that these parties be trusted by all participants. Their non-anonymity also means that a dedicated attacker will eventually always be able to attack the system.

Bitcoin's solution is to do away with the idea of a fixed, identifiable signer entirely. Instead Bitcoin's ledger is authenticated by a collection of signers called *miners* who do not identify themselves to other participants and may costlessly enter or leave the system. They produce signatures by a process called *mining* wherein they collectively produce *proofs of work*[Bac02] on successive blocks of transactions.

In this section we explain how mining works and how it provides authentication.

3.1 Authentication in an Anonymous World

A cryptographic *digital signature scheme* works as follows. A signing party produces a keypair (s, v) of "signing" and "verification" keys, and publishes v in some public channel alongside her name. Then given a message m, she can produce a *signature* σ such that anyone can check that σ is valid. That is, there is a verification algorithm which takes v, m and σ and will always output 1 if the signature was created honestly.

To be secure, traditional digital signatures must be *unforgeable* by any computationally-bounded adversary except with negligible probability, where *forging* specifically means winning at the following game:

1. The signer gives the verification key v to the adversary.
2. The adversary sends messages m_i and to the challenger and receives valid signatures σ_i on these messages. He may do this as many times as he likes.
3. The adversary produces a message m, which was not queried on earlier, along with a valid signature σ on m.

This notion of security is known as *existential unforgeability under chosen-message attack* and is standard in the cryptographic literature.

(For a multiparty signature, there are multiple verification keys corresponding to multiple signers, and signatures are only valid if they are produced by an "admissible subset" of them. To define security, the above game is modified so that the adversary may also request (and receive) secret keys, as long as no subset of the secret keys that he requests forms an admissible subset.)

We see the verification algorithm uses the verification key v to check signatures, and in this way checks the "identity" of the signer that produced it. Since anyone can produce a keypair, for such signatures to be valuable there must be some public record tying verification keys to real-life identities. Then given dishonest behaviour, i.e. signatures on invalid histories, blame can be assigned.

It seems then that this notion of authentication cannot be adapted for use in a system where the signing parties are anonymous and ephemeral. In fact, it's not clear what "authentication" can even

mean in such a system! After all, if anyone can anonymously produce a signature, there is nothing distinguishing dishonest signatures from honest ones, false histories from the true one. Formally, the above security definition no longer makes sense since the adversary is free to join the set of signers and produce a "forgery" that way[2].

To get around this problem, Bitcoin uses an alternate security model in which all parties are on equal footing, but they are incentivized to agree. We describe this in the next section.

3.2 Defining Security for a DMMS

For a DMMS, all parties are on equal footing; we cannot have a security property with an "adversary" having incomplete knowledge. We therefore define a DMMS in three components, none of which resemble the key generation algorithm of a traditional signature:

- A *cost function c* which takes a trace of an algorithm's execution and outputs a "cost" $t \in \mathscr{T}$, where $\mathscr{T}$ is some "cost domain". It must be linear in the sense that the cost of running two algorithms in series is the sum of their individual costs.
- A randomized algorithm AttemptSign which takes a message m and outputs a signature σ. The cost of this algorithm should be 1 for every message m.
- A deterministic algorithm Verify which takes a message m, signature σ, and target cost T, and outputs either 0 or 1.

We say the DMMS is *correct* if $\mathsf{Verify}(m, T, \mathsf{AttemptSign}(m)) = 1$ with probability $1/T$ for all $T \in \mathscr{T}$, where the probability is taken over the coins of AttemptSign. We say it is *secure* if no polynomial-time algorithm $\mathscr{A}$ can acheive

$$\mathsf{Verify}(m, T, \mathscr{A}(m)) = 1$$

with probability greater than $1-(1-1/T)^t$, where t is the cost of $\mathscr{A}$'s execution.

In other words, a secure DMMS is one for which there is no better (in the sense of producing signatures which verify) signing algorithm than to simply execute AttemptSign repeatedly.

We briefly justify our security definition. In order to achieve a dynamic membership set, we cannot have expensive entrance costs, nor can we allow existing signers to exclude new entrants either explicitly or through economic consideration. This implies that signing should be "separable" in the sense of neither requiring nor rewarding any communication between the signers, which in turn means that running a signing algorithm for twice as long should be exactly as likely to succeed as running signing algorithms in parallel across twice as much hardware. In the limit, this means that the optimal signing algorithm should consist of executing a single basic step many times independently, which is the given definition.

[2] As we will see in Section 4, given a cryptocurrency, it is actually possible for anonymous parties to bond value, giving a mechanism to punish dishonest behaviour without identifying anyone. This is essentially what proof-of-stake is. However, a cryptocurrency needs a consensus to work, so the phrase *given a cryptocurrency* prevents us from using proof-of-stake here, on pain of circular reasoning. We address this further in a later section.

3.3 Mining as a DMMS

Bitcoin mining works using a hash-based proof-of-work called *hashcash*[Bac02]. It is a DMMS in the *random oracle model*[BR93]. The random oracle model is a computational model in which a hash function is modelled as a "random oracle" or truly random function[3] whose output is uniformly random and cannot be computed except by evaluating the function itself.

The use of the random oracle model is quite controversial[Gre11] but there is strong empirical evidence justifying its use in security arguments. From here on H will denote a hash function taking arbitrarily many bits to 256 bits, modelled as a random oracle.

Here is Bitcoin's DMMS:

- The cost function gives the number of random oracle calls in the execution.
- AttemptSign takes a message m, and outputs a random $\sigma \in \{0,1\}^{256}$.
- Verify takes a signature σ, message m, and target T. It outputs 1 if and only if $H(m\|\sigma) < 2^{256}/T$.

It easy to see that in the random oracle model, there is no better way to produce valid signatures than to simply query the random oracle repeatedly.

3.4 No Universal Time

Notice that in the previous section we used the number of hash-function calls as our cost function, which is roughly proportional to the number of computations, which in turn is roughly proportional to the amount of heat dissipated, which finally is roughly proportional to the economic and environmental cost of producing these signatures.

An obvious question is whether we can use a cost function which is "cheaper" to satisfy; in particular, why can't we directly use clock time? For that matter, why are we creating chains of DMMS-signed blocks instead of just directly ordering transactions in time, always resolving conflicts in favor of the first?

The answer is that there is no well-defined clock time in a distributed system. Network latency gives a finite speed of information propagation, which we know from special relativity means different observers cannot agree on the time-ordering of events that occur closely in time.

If this were the only problem, requiring transactions to be spaced out by several seconds would be sufficient (if conflicting transactions occur too close together, both are thrown out; but by waiting a few seconds after each transaction all parties can be assured that this won't happen). However, the situation is worse than this for two reasons:

- "Network latency" is not something that can be bounded in an adversarial setting. An attacker may be able to slow systems by arbitrary amounts using denial-of-service measures, and may be able to physically partition the network by other means.

  In relativistic terms, this means that there is no amount of waiting that will assure somebody that they are no longer spacelike separated from other participants in the network.

[3] This model is thoroughly unrealistic, since a truly random function from, say, 512 bits to 256 bits, would require almost $2^{512} \cdot 256$ bits on average to represent. This is more bits than can be represented in the known universe.

- Users who are new to the network or have been offline recently need access to historical data. But there is no way to verify after-the-fact what order transactions occurred in, so they cannot be assured that the transactions they are receiving actually occurred before any conflicting ones.

3.5 Consensus From DMMS

Now that we have this notion of DMMS, and have shown that Bitcoin's hashcash is a secure one, let's consider obtaining a distributed consensus from a DMMS.

Our claim is that given a DMMS, it is possible to create a distributed consensus.

We first need our cost function to measure, i.e. be a monotonic function of, (a) some scarce resource which cannot be allocated to producing a signature on more than one message at once, and (b) the amount of time required to produce the signature on average. (It may be required for decentralization that this resource be permanently consumed, as in Bitcoin, so that marginal costs of mining dominate capital costs. This is argued by the author in [Poe14a], and disputed by John Tromp in [Tro14].)

In Bitcoin's case, our cost function is defined as "number of hash function calls". We claim that this is actually a measure of energy consumed in computing the signature, a claim justified by the *Landauer limit*[Lan61]. This is a physical minimum on the number of joules of heat which must be dissipated by any irreversible bit operation. By calculating the number of irreversible bit operations involved in a `sha256d` computation, we can, at least in principle, put a lower bound on the amount of energy that must be expended in creating a Bitcoin DMMS.

It is also therefore a measure of the amount of time taken to produce a signature, since only so much energy can be expended per unit time without producing a black hole (which would not allow a signature, or indeed any information, to be extracted in usable form). Of course, in real life Bitcoin miners operate nowhere near the black hole limit, and this time is determined by the speed of mining hardware. As this hardware improves, and more of it comes online simultaneously, the time required to produce a DMMS decreases. In Bitcoin, the target cost adjusts in response to this to keep the time per signature around 10 minutes[4].

So, with a secure DMMS whose cost function measures a scarce resource, how do we get consensus history? First, we assume a network that is syncronous, so that all valid date is available to (a supermajority of) participants within some time λ. We split our transaction history into a series of *blocks*, which each contain a set of transactions along with a cryptographic commitment[5] to the previous block in the series. Each block, to be valid, must have a DMMS on it, whose target is set so that blocks require significantly more time than λ to be created. (This way all miners working on the consensus history know which block is the most recent, and its original creator does not gain a

[4] The reader may be wondering how this adjustment can be done accurately, given that in Section 3.4 it was argued that there is no universal time. In fact, timestamps are inserted into blocks by miners, and it is true that there is no way to enforce that they do so honestly. Bitcoin is resiliant to small target skew caused by dishonest timestamps, and the target is prevented from changing too quickly, so that large target skews are expensive to create and likely to be stymied by non-cooperating miners. For more discussion see [Poe14c, Section 6.3].

[5] A *commitment* is a cryptographic object which is computed from some secret data, but does not reveal it, such that the data cannot be changed after the fact. An example of a commitment is a *collision-resistant hash function*: given data x, one can publish $H(x)$ where H is a hash function, and only later reveal x. Verifiers can then confirm that the revealed value is the same as the original value by computing $H(x)$ themselves.

large advantage by knowing it earlier than anyone else.) To be clear: the target cost for each DMMS is defined by the rules of the system, and is not at the miners' discretion.

Network participants operate as follows: they consider all series of valid blocks beginning with some *genesis block* hardcoded in the system. (Since each block commits to the previous block, the result is a directed acyclic graph rooted at the genesis block, so these series can be consider as paths in this graph.) They weight each series by summing the target cost of the DMMS's on each block. Whichever series has the highest weight they consider as the "true" history.

Miners create blocks by collecting transactions according to their whim, along with a special "reward transaction" which assigns fees from the other transactions plus some network-defined subsidy to the miner himself, adding a commitment to the most recent block of the true history (in their view), and computing a DMMS on this. If another miner creates and publishes a block, they update their "most recent block of the true history" accordingly and change the commitment in their block if necessary. Once they compute a DMMS, they publish the finished block to the network.

We claim that this forms a consensus history in the sense that the network will gradually come to agreement on which blocks belong to the true history and which don't, with disagreement occuring only near the tip. This is argued concretely by Miller and LaViola Jr. in [MLJ14] (using the observation that our measurably-costly DMMS's are "moderately-hard puzzles" as described in that paper) but an informal argument is as follows.

Because the network is synchronous with characteristic time much smaller than the block rate, all participants are quickly aware of the highest-weighted history.

We further claim that a majority of the network is working on producing a DMMS which extends the true history. An elegant reason that this is true is given by Vitalik Buterin in [But15]: since the reward transaction is only recognized if its block occurs in the true history, a Nash equilibrium for each miner is to go along with the majority[6].

For the true history to then change, an attacker must produce an alternate higher-weighted history, which forks from the true one some number N of blocks away from the tip. He has fewer resources than the set of miners who are working on extending the true history, since this is the majority. Then the probability of him gaining on the network is less than 1 (since in any given time period the network makes more attempts to extend the true history than he can to extend his false one), and he must do this more than N times before his history is longer than the true one. If his probability of gaining once was $P < 1$, his probability of winning is P^N, which is negligible in N.

(This is a poor argument since it only considers an attacker who constantly gains on the network, and it doesn't consider difficulty changes which may allow even a very weak attacker can recreate old blocks faster than the network can create new ones. The author claims without proof that these are not serious problems, only tedious ones.)

3.6 Consensus Without DMMS

Is it necessary to use a DMMS to produce a distributed consensus? This is an open question. The author's guess is "no". In particular, simple changes to Bitcoin's protocol, such as rewarding miners

[6] In that same blog post, Buterin says "if you are tired of opponents of proof of stake pointing you to this article[Poe14b] by Andrew Poelstra, feel free to link them here in response". It is not clear what he means by this; he did not, there or anywhere, refute that paper's claim that you cannot produce consensus except by consuming an external resource.

with "coupons" to mine far-future blocks with lower difficulty[BCD+14, Section 6.1] seem unlikely to harm consensus while definitely not satisfying the given definition of DMMS.

4 Proof of Stake

With Bitcoin's consensus system behind us, we now consider the most popular alternate proposal, *proof-of-stake*. Proof of stake is described in [Poe14b] as

> With the advent of modern cryptography, the idea that *information* can be physically real — and valuable — has moved from the dingy halls of philosophy departments to the concrete world of business. We are all familiar with the economic activity enabled by secure communication: negotiations, contracts, transactions, sales and commands can be sent on the public Internet with no fear of forgery or interception. We are also familiar with the financial consequences when secret data is lost or stolen.
>
> Since the advent of cryptographic currency in January 2009 [Nak09] this notion of valuable information has been made concrete. It is possible to hold and exchange a *fungible* store of value, using public communication media, with cryptographic rather than physical security preventing fraud or theft. Rather than saying "this encryption key is worth $10,000 because that's what it will cost us if its encrypted data is exposed" one can say "this key is worth $10,000 but can be broken up, sending only $20 of it to another party while keeping the rest".
>
> With this context, proof-of-stake is a simple idea. A proof of stake is a cryptographic proof of ownership. With cryptocurrencies, it is possible for a proof-of-stake to not only prove ownership of a precise amount of currency, but also prove that this currency satisfies some property (say, it is locked and unspendable until some contract is satisfied).
>
> In particular, proven stake in a scarce and experimental cryptocurrency can be considered a proof of vested interest in the project's success. By proving stake which is time-locked, it can be used to prove interest in the project's continued (and sustainable) existence.

That is, proof-of-stake is the idea that we can use cryptographic proof, rather than physical arguments such as the Landauer limit, to demonstrate that some computation is costly.

4.1 Proof-of-Stake versus DMMS

The idea behind using "pure proof-of-stake" to obtain consensus on a cryptocurrency history is that it is possible to produce a DMMS-like[7] signature whose cost function measures the currency itself. This is in principle easy, and we again quote [Poe14b]: with a sufficiently expressive ledger,

[7] Proof of stake schemes do not generally attempt to be progress-free, so they cannot be correct or secure DMMS's for this reason alone. However, the high-level principle of "making history costly to create and rewarding participants in the true one" remains the same, so we continue to speak in terms of costs.

currency holders are able to lock their currency for some amount of time, renting "stake" which is cryptographically verifiable. Then to extend the consensus history, rather than attaching a proof of work, each stakeholder digitally signs the extension. For reasons of practicality, typically a small random selection of stakeholders is chosen for each extension, and only a majority of the selection are required to give the extension validity. The chosen stakeholders are given a reward and after some time they are able to unlock their stake if they so desire.

The idea is that rather than depending on the economic inviability of taking control of a history, stakeholders are incentivized to agree on each extension because (a) they are randomly chosen and therefore unlikely to be in collusion, and (b) even if they can collude, they do not want to undermine the system (e.g. by signing many conflicting histories) because they want to recover their stored value when their stake comes unlocked, and (c) they have limited capacity to cause havoc anyway, since for the above reasons the next random selection of stakeholders will probably choose only a single reasonable history to extend.

This exposition slightly overcomplicates things: real proof-of-stake systems use single signers rather than this majority system.

The problem with this is that to well-define a cost function which measures bonded stake in some currency, we need a consensus history for that currency to already exist. If this is defined along the lines of "there exists some history in which these coins are bonded" then while the resulting DMMS may be well-defined, its cost function no longer measures something scarce, and the argument in Section 3.5 falls apart.

This scarcity may be recoverable by punishing stakeholders who sign multiple histories. For example, if they use Schnorr or ECDSA signatures and are constrained to a specific choice of nonce, they must sign two messages with the same (key, nonce) pair in order to sign multiple histories, and this allows anyone to algebraically solve for their private key. However, this does not prevent them from "grinding" through many potential blocks, only publishing one which causes them to be the signer for the next block[8], and taking control of the blockchain by doing this repeatedly. This is an example of a "stake-grinding" attack, which is a special case of a more general problem described in the next section.

In general, the use of one-time signature schemes can prevents the publication of conflicting histories (though only if the attacker values the lost coins more than he values causing mayhem), but they cannot prevent the private creation of conflicting histories.

4.2 Costless Simulation

Ultimately the problem is that the coins bonded against a stake signature only exist within the blockchain to which those coins belong. This means that if the blockchain can be cheaply created

[8]Notice that any randomness must be determined by the blockchain, since this is the only object on which there is consensus. So miners are able to skew any random numbers used by the system in this way. Using multiple blocks or long-past blocks does not solve the problem; ultimately, for every random number there is some party who can manipulate it.

by some party, such a party can create multiple blockchains and select one which favors him.

If cheap histories are available to *everyone*, clearly the system is broken. The point of bonding stake is that "cheap" histories are only available to those willing to those who are holding the currency, and presumably these people value the system too much to attack it. However, because there is no universal time (and to new users, no universal history), there is no way to differentiate users who are "now" holding the currency from users who "were" holding the currency. To the system, for each point in history there is a set of signing keys which allow transfer of coins, bonding and unbonding of stake, and blocks to be signed. Presumably there was some point in real time when the block was created, and at that point these keys corresponded to ownership. However, as real time ticks on, the block remains static, and with it any signing keys associated to it.

To restate: In real time, the blockchain gets longer, stake is unbonded, coins are exchanged for other goods and services, and the keys whose misuse would mean loss of value become meaningless. But to a block in an immutable blockchain, keys will never change their meaning.

Since in a proof-of-stake system some of these signing keys determine the future, this is a problem. Anyone with access to these keys (this may be the original user, who has since sold her coins, or it may be somebody she sold or otherwise lost the keys to; users are not good at long-term storage of secret keys, especially ones they believe they no longer have use for!) has the ability to fork the network, or to create alternate histories which new users cannot distinguish from true ones.

The point is this: even if stakeholders are bonding coins with a large market value, in a way that they will lose the coins if they behave dishonestly, this is only a deterrent to dishonest behaviour *until they move their coins*, which, being an event in the future (both in realtime and blocktime), is necessarily not something that can be detected at the the current point in blocktime. So cheap histories, or "costless simulation", are not something that can be prevented while only bonding value defined inside the system.

4.3 "Long-Range" versus "Short-Range" Attacks

It is possible, by requiring stake to be bonded for many consecutive blocks, and by choosing signers using randomness extracted by long-past (in blocktime) blocks, to force the attacks described above to rewrite long stretches of history. This is often described as "preventing short-range attacks".

It is clear that this does not address the costless simulation issue; after all, if it's easy to change history, it's easy to change long stretches of history. However, proponents argue that since for an honestly-created history, long stretches of blocktime correspond to long stretches of real time, any revision of so much history is sure to contradict the history as remembered by participants in the system. Thus such an attack would be detected, recognized as an attack, and the new history rejected.

If this is implemented correctly, there is no problem with this, except that it *changes the trust model* from that of Bitcoin. New users who encounter multiple histories are no longer able to distinguish them on their own; they need to ask existing participants in the network (which may include friends and family, large corporate entities with reputations to maintain, public websites, etc.) which history they know to be the true one. This is not a distributed consensus! It is a different sort of consensus, which may be formed amongst always-online peers in a decentralized way, but

depends on trust for new users and temporarily offline ones. It is correspondingly vulnurable to legal pressure, attacks on "trusted" entities, and network attacks.

4.4 Other Considerations

Again, we quote [Poe14b]:

> Further, this ability to control the future selection of stakeholders (and even the *set* of stakeholders, by controlling which transactions appear in blocks) has serious consequences. This is because even without a deliberate attacker, the signers who extend the history at every point have an incentive to direct the history toward one in which they have more stake (and therefore more reward), which causes the system to trend toward centralization. They may do this by skewing the stake selection of future blocks, or more insidiously by censoring transactions which (may eventually) increase the set of stakeholders.

5 Conclusions and Further Research

We have described a mechanism, DMMS, for obtaining a distributed consensus. While DMMS, in conjunction with some economic requirements, is sufficient to form consensus, it is probably not necessary. Open problems include reducing these economic assumptions (or showing they cannot be removed), and determining necessary conditions under which distributed consensus can be obtained.

We also explored an alternative to DMMS, *proof of stake*. We showed that by depending only on resources within the system, proof of stake cannot be used to form a distributed consensus, since it depends on the very history it is trying to form to enforce loss of value.

References

[Bac02] A. Back, *Hashcash — a denial of service counter-measure*, 2002, `http://hashcash.org/papers/hashcash.pdf`.

[BCD+14] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, *Enabling blockchain innovations with pegged sidechains*, 2014, `https://www.blockstream.com/sidechains.pdf`.

[BMC+15] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, *Research perspectives and challenges for bitcoin and cryptocurrencies*, To appear, IEEE Security and Privacy 2015, 2015.

[BR93] M. Bellare and P. Rogaway, *Random oracles are practical: A paradigm for designing efficient protocols*, ACM Conference on Computer and Communications Security, 1993, pp. 62–73.

[But15] V. Buterin, *The P + epsilon attack*, 2015, https://blog.ethereum.org/2015/01/28/p-epsilon-attack/.

[Cha83] D. Chaum, *Blind signatures for untraceable payments*, Advances in Cryptology Proceedings of Crypto **82** (1983), no. 3, 199–203.

[FLP85] M.A. Fischer, N.A. Lynch, and M.S. Paterson, *Impossibility of distributed consensus with one faulty process*, J. Ass. for Comp. Mach. **32** (1985), no. 2, 374–382.

[Gre11] M. Green, *What is the random oracle model and why should you care? (part 1)*, 2011, blog.cryptographyengineering.com/2011/09/what-is-random-oracle-model-and-why.html.

[Lan61] R. Landauer, *Irreversibility and heat generation in the computing process*, IBM Journal of Research and Development **5** (1961), no. 3, 183–191, doi:10.1147/rd.53.0183. http://worrydream.com/refs/Landauer%20-%20Irreversibility%20and%20Heat%20Generation%20in%20the%20Computing%20Process.pdf.

[MLJ14] A. Miller and J. J. LaViola Jr, *Anonymous Byzantine consensus from moderately-hard puzzles: A model for Bitcoin*, Tech. Report CS-TR-14-01, UCF, April 2014, https://socrates1024.s3.amazonaws.com/consensus.pdf.

[Nak09] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, 2009, https://www.bitcoin.org/bitcoin.pdf.

[Poe14a] A. Poelstra, *ASICs and decentralization FAQ*, 2014, https://download.wpsoftware.net/bitcoin/asic-faq.pdf.

[Poe14b] ———, *Distributed consensus from proof of stake is impossible*, 2014, https://download.wpsoftware.net/bitcoin/old-pos.pdf.

[Poe14c] ———, *A treatise on altcoins*, 2014, Unfinished, https://download.wpsoftware.net/bitcoin/alts.pdf.

[Tro14] J. Tromp, *Cuckoo cycle: a memory-hard proof-of-work system*, https://github.com/tromp/cuckoo, 2014.