**Introduction to Cryptography** 4/3/2018

# Lecture 18: Proof of Stake and Byzantine Agreements

*Instructor: Vipul Goyal* *Scribe: Justin Cheng*

## 1 Motivation

Proof of work is very expensive, since it requires the "good guys" to generate more hashes than the "bad guys", which could cost millions of dollars.

Proof of stake was one of the first models that tried to circumvent this. Instead of attributing mining power to computing power, as is done in proof of work, proof of stake seeks to attribute that to the proportion of coins held by an individual. Thus, all individuals who owns coins of the currency and wish to participate as builders of the blockchain are able to do so.

## 2 Basic Procedure

Before a new block is added to a blockchain, each builder makes a block proposal with a score, and the block with the highest score gets selected to be added. The scoring system exists to ensure that the score of a blockchain, which is the sum of the scores of the blocks, is proportional to the number of coins held by the builder community.

Let's say that some adversaries try to create a fork of the blockchain. If we assume that they have fewer coins than honest builders, then their fork will have a lower score, and the fork won't be able to take over the main blockchain.

There are two ways to increase the score of a blockchain: increasing the scores of the individual blocks, or increasing the number of the blocks.

POS implements a fixed schedule at which blocks can be added to the blockchain. This prevents adversaries from generating a lot of blocks in order to take over a blockchain.

## 3 Score Generation

A score generation scheme is as follows:

1. my_rand = HASH(prev_block || my_pub)
2. my_score = $f$(my_rand) $\times$ my_acct_balance

Adversaries can attempt to increase the chances at getting a higher score by splitting their account balance into multiple accounts. However, by tweaking distribution $f$, the probability of getting a good score is the same as if the balance wasn't split. If $f$ is an exponential curve, the expected maximum of the score can be the same.

Score is probabilistic, so a bad fork could overtake the main chain. Thus, we must wait for a number of block confirmations to ensure that the correct, honest blockchain is the main blockchain.

Whenever a builder publishes a block proposal, they sign the block with their public key. Since their public key and account balance are both publicly known, anyone can confirm the score of the block.

However, HASH is not a random number generator, but a deterministic hash function that's used as a random number generator. Thus, there is an attack.

Suppose that block 1 has already been confirmed. An adversary has a great score for block 2 and is convinced that it'll be selected for the next block. Before proposing block 2, they predict the score for block 3, which they can do through the equation above. If the score is lower than desired, they can continue changing block 2 until a satisfactory score for block 3 is achieved. This way, adversaries can take control of the entire blockchain.

**Another score generation scheme**

my_rand = HASH(block_seq_no || my_public_key)

Unlike the previous scheme, the random numbers can't be manipulated by changing block content. However, an adversary can search for public keys that lead to good scores for every future block. Since this can all be precomputed, an adversary can easily take over the blockchain.

**A score generation scheme used by NEM**

my_rand = HASH(prev_rand || my_public_key)

In this case, the random number is a function of the sequence of people who generated the previous blocks. The only way to manipulate this random number is by manipulating the sequence of people, but this implies that you already have control of the blockchain.

The random number can be predicted by predicting the future sequence of people. This isn't very difficult, since scores can be precomputed through public information. However, you can only realistically predict around 15-20 minutes in the future, but not 2 hours in the future.

**A score generation scheme used in Byzantine Agreement Cryptocurrencies**

my_rand = HASH(digital signature of miner(prev_rand))

The digital signature has to be a unique signature scheme, which is a digital signature scheme that has a deterministic generation process. If a digital signature involves generating a random number, it is not unique signature scheme.

In this case, my_rand is again a function of the sequence of people who signed previous blocks. However, to calculate the hash, they must generate the signature, and the only person who can do that is the miner. Thus, this is a random function that is hard to predict.

It's possible to predict one block ahead if you have control of the previous block. If an adversary is in control of block 1, they can look for an account that gets a good score in the next block, and include a transaction in block 1 to send their funds into that account. Then, they will be in control of block 2.

The solution to this attack is bonding. Funds must be locked in an account for 10-15 blocks before that account can be a builder. If you no longer wish to be a builder, you can request to unlock your funds, which involves waiting for another 10-15 blocks. Bonding also allows malicious behavior can be detected, if the same digital signature is observed on two different branches.

Evidence suggests that 90% of cryptocurrency owners don't run the software needed to generate blocks. Thus, these people can't contribute to the "good guys" that try to fight the "bad guys". Instead, NEM allows these people to hand their stake to somebody else and have them vote. Although this solves the problem, this is susceptible to social engineering.

## 4 Byzantine Agreements

In Byzantine agreements, a committee decides whether or not to add blocks to the blockchain. Committee members consider proposals and negotiate amongst themselves until they reach a **unanimous** conclusion. Once this occurs, they put the block onto the blockchain, and the whole community signs the block. There are no forks and no need to wait for confirmations, since the decision was unanimous.

Traditionally, the committee is composed of a typed up list of trusted authorities, stored in a text file. Ripple's committee is similar to this.

Initially, people dismissed Byzantine agreements since they thought committees were antithetical to the philosophy of cryptocurrency. However, they eventually realized that just because there's a committee, there's no reason it should be long standing. They could function similar to the US jury system, in which juries are selected, stand in for one trial, and then disbanded. Similarly, stakeholders can be randomly selected, generate one block in the blockchain, and then disbanded.

What if stakeholders were selected based on the number of coins they have?

For example, suppose that 500 committee members are needed to select the next block. A plausible method is to randomly select 500 coins and enlist the owners of those coins as committee members.

Alternatively, users can calculate their own probability to be selected based on the amount of coins they have, which they can publicize to prove that they are a part of the next committee.

Although a completely unanimous decision is ideal, in the real world users crash or go offline, so this system needs tolerance for failure, which is typically 1/3 of the committee. If 2/3 of committee members sign, that's good enough.

One of the simplest algorithms is "Byzantine agreement made trivial" by Micali. The biggest disadvantage about this is that it requires a synchronous network, which means that messages must be delivered in a finite amount of time. Since the internet isn't a synchronous network, this algorithm will only work with complicated modifications.

The second most complicated algorithm is Tendermint. The biggest downside is that this isn't peer reviewed, but it's close. It's also similar to other peer reviewed algorithms.

The third most complicated is PBFT, created by Barbara Liskov.

## 5 Byzantine agreement made trivial

Let's explain Byzantine agreement algorithsm through this algorithm, with three simplifications:

- This algorithm reaches agreement on boolean, not block.
- This algorithm has no termination conditions. It will reach unanimity, but can't detect that it has.
- This algorithm requires existence of "magic coin in the sky". Everyone can see it, but nobody can manipulate it.

Goal: If everybody is in initial agreement on true or false, output true or false, respectively. If initial agreement is divided, output either one. Everyone terminates.

Everyone does at the same time:

1. Everybody initializes bit *b*
2. Loop:
   (a) Transmit *b* to everyone. Everyone knows everyone else's bit.
   (b) Magic coin flips and generate a random bit.
   (c) If 2/3 of committee members say true, set *b* to true.
   elif 2/3 of committee members say false, set *b* to false.
   else set *b* to magic coin.

To see how this works, imagine 4 committee members A, B, C, and D, with D being evil. D's goal is to introduce as much chaos as possible into the system, and only sends message with the intent to disrupt.

**Case 1: A, B, C are initially true**

Once ABC reached agreement, D doesn't have the weight to yank them out of agreement.

**Case 2: A, B true, C false**

If D votes true, D can force the output to be true.

If D votes false, D can force the output to be random (magic coin).

D can send different values to different people. For instance, D can send true to A and false to B. However, no matter what D does, D can't force the output to be false. Either way, some end up with true, and some end up with the magic coin. If the magic coin is true, everyone ends up with true. D can't prevent this!

Only two paths are reachable, either a unanimous consensus, or the magic coin.

There is a threshold on the number of the "bad guys". If they control a large percentage, then this algorithm won't work.