TL;DR:

A thought experiment on how to make a RANDAO that is less influenceable by single validator by lowering last-contributor influence on the result by adding a “clock” that entropy contributors can’t predict.

The construction presented here lowers the influence of contributors and therefore lowers the incentive to look at the rewards in all shards. However, the current construction has very poor liveness, and I haven’t looked at 51%-attack scenarios.

The main problem with the RANDAO is that there is a “last influencer” of the entropy, who can make a decision to reveal or not reveal. Assume now that we only generate entropy every epoch of K

blocks, XORing all the reveals. Then the revealer of the K

'th block gets this unfair advantage, whereas all the other revealers have no information how they might influence the entropy (assuming no collusion). This can be fixed if we assume that there is an external random clock that could end an epoch at any point in time: You would not be able to know whether your block is the last one or not, thus making the point moot (or at least reducing the value of skipping a block to influence entropy by 1/K

with K

being the average length of an epoch).

Here’s an idea how to generate such an arbitrary “epoch end signal” without the generator of the block knowing whether the epoch will end with his block or not: After each block was generated, each validator XORs the last reveal with their own next reveal*. If it is smaller than a cutoff value p_C

(that can be chosen arbitrarily to get any desired average epoch length K

), then their reveal is a vote to end the epoch. If a certain number, of validators M

vote to end the epoch, they can generate a block together (in practice one of them that has all the votes can generate that block) that ends the epoch immediately and makes the XOR of the last epoch reveals the entropy for the next epoch.

Analysis of this scheme: In this scheme, there are two ways to influence the entropy. When revealing a block of which I don’t like the entropy outcome (if the epoch were to end after it), I can decide not to reveal it. However, there was only a 1/K

chance of that block becoming the last block anyway, so the expected influence on entropy is 1/K

bits.

Secondly, when voting to end an epoch, I can decide not to vote for an entropy outcome I don’t like. This time, I can only influence the vote if there were exactly M

voters to begin with (or rather of M-1

of the other possible voters decided to reveal), in all other cases my missing vote will not have any influence on entropy (assuming no collusion, analogous to [RNG exploitability analysis assuming pure RANDAO-based main chain]).

The probability of getting exactly M

votes can be computed using the binomial distribution:

\displaystyle p_{=M} = \binom{N}{M} p_C^{M} (1-p_C)^{N-M}

where N

is the total number of validators.

The probability of getting more than M

votes can be approximated using the normal approximation to the binomial distribution:

\displaystyle p_{\geq M} = 1 - \Phi \left( \frac{M-p_C N}{\sqrt{N p_C (1-p_C)}} \right)

(\Phi

is the distribution function of the standard normal distribution). Now assuming that I am a rational validator and I see an entropy outcome that is unfavourable to me. Assuming that I have a positive vote and that the overall vote outcome would be positive, then I can only influence the vote if there are exactly M

votes available and not more, i.e.

\displaystyle p_\text{Voter influence} = \frac{p_{=M}}{p_{\geq M}}

.

We can clearly make this smaller than 1, which give each voting validator an expected influence of less than 1 bit on the outcome of the vote.

For tuning the constants, we first need to choose a suitable p_C

by balancing between the following two factors:

1. We want p_C

large, because then the variance of the binomial distribution N p_C (1-p_C)

is large giving less of a chance to influence

1. However, we will need to choose M \text{very}\approx p_C N

(more accurate formula below), so with p_C

too large the number of votes to be aggregated would become unmanageable.

For a given p_C

, and a desired p_{\geq M} = 1/K

can be achieved by setting

\displaystyle M = p_C N + \sqrt{N p_C (1-p_C)} \Phi^{-1} (1-p_C)

Intuitively it makes sense to set p_\text{Voter influence} \approx p_{\geq M}

to give block generators and epoch end voters a similar amount of influence on the entropy outcome. As an example, with N=100000

validators, we can use p_C=0.00325

and C = 348

to get p_\text{Voter influence} \approx p_{\geq M} \approx 0.1

. Then the expected influence of single validators on the entropy is reduced to approximately one bit.

- Maybe you want a separate hash onion for this in order not to interfere with the normal block generation and reveal secrets early.

Current problems with this scheme:

1. Liveness: The above assumes all validators are online all of the time. If validators are randomly offline, and this is evenly distributed, this can be incorporated by simply scaling p_C

. However, of course this is not how reality works. Here it turns out to be a real weakness to use the binomial distribution as a base for voting. It would be nice if we could find a way to compute the votes were the votes are highly correlated (a kind of “landslide mechanism”). This would both reduce the influence of the single voter and could improve the liveness of the mechanism. But how to construct such a mechanism without it being predictable in any way by the contributor of the block?

1. Collusion: I haven’t made a 51% analysis as in [RANDAO beacon exploitability analysis, round 2] yet.