Algorand's self selection (https://github.com/ethresearch/p2p/issues/5#issuecomment-441457619) has been considered as a protection against DoS attacks against block proposers (BPs). The goal is to choose a BP from a pool of eligible validators in such a way that only the current BP knows it has been selected. When it publishes a block, it simultaneously publishes a proof showing that it is indeed the current BP.

Algorand's protocol cannot be used as a means to reliably select a single BP. For example, with a single BP to choose among 300 validators, Algorand would fail to select anyone with a probability ~36% and would select two or more BPs ~26% of the time.

We found a possible solution to secret self-selection using ZK-SNARK (ZK-STARK would work as well). Obvious caveat: it's expensive.

So the goal is to select exactly one participant from a pool of many (possibly thousands) of eligible participants. We require the following setup:

- The list of all currently eligible participants is known in advance
- These participants are identified by their public keys.
- A random beacon is available.

We want to select one of these public keys in a way that only the owner of the public key knows it has been selected. For any round the random beacon emits a random number m which participants will need to sign.

A simple solution is having all participants:

- Sign the message m: signed(m)
- Send the hash of this message: hash(signed(m))

At this point we can sort the participants by their hash(signed(m)). With a random beacon r we can select one of the participants, by picking the one at position r modulo number of participants. While this position is known by all, only the owner of the corresponding public key can publish signed(m), hence prove it was selected. But nobody can guess who was selected before signed(m) is actually published.

This scheme has obvious issues: anyone can send data d and pretend it's some hash(signed(m)). Thus a byzantine actor can crash the protocol by sending a lot of false data. Furthermore, since we want to keep these submissions anonymous (in particular: unsigned), there is no way of knowing whether data was produced by an eligible participant or someone else, nor whether it is the hash of a signature. We need to be sure that the hash(signed(m)) is legitimate, without identifying the participant. We also need to be sure that even a legitimate but byzantine participant cannot send invalid or multiple messages.

We use a ZK-SNARK (or ZK-STARK) to solve this. We put the eligible public keys in a Merkle tree. Participants will not only send their hash(signed(m)), but also a proof that they are an eligible participants and that the value they sent is valid. Hence there will be one proof per participant. Participants will prove that they own an eligible public key by providing the Merkle path of its public key. They will then prove that the message is actually signed by the corresponding private key.

Public parameters:

- A public random number m (emitted by the random beacon).
- The roothash of all the participants' public keys: rh
- h = hash(signed(m))

Secret parameters:

- signed(m)
- The signer's public key: pk
- Merkled path: mp

Checks performed by the ZK-SNARK:

- The public key belongs to one of the participants: the Merkle path mp leads from the public key pk to the root hash rh,
- signed(m) checks out against the public key pk and the random number m
- hash(signed(m)) given in the public parameters is the hash of signed(m) given in the secret parameters
- signed(m) in the secret parameters is the same m given in the public parameters.

Participants must send their public parameters along with the proof π. A message gets accepted if and only if the public parameters m and roothash are valid and the proof can be verified.

There is still one problem: we generate as many proofs as there are participants, and these proofs would all need to be included on chain and verified for each round. It's possible to generate these proofs in advance without compromising security (eg. have a long delay between the publication of the random number m to sign and the time when proposals (h, π) are committed to the chain), but the workload remains.

There are two possible solutions here.

First, we can dramatically limit the number of proofs to be both generated and committed by combining this protocol with Micali's protocol. This works by only accepting pairs (h, π) where ($\star$) hash(h) < t for a threshold t selected in such a way that with overwhelming probability (1) there will be a participant that satisfies ($\star$), and (2) (say) 20, 100 or 1000 participants will satisfy ($\star$), and hence be allowed to participate. As a result, even if there are thousands of possible participants, just a few of them will actually have to generate a proof. This is equivalent of creating a self selected committee whose members will be legitimate in submitting zk-snark proofs onto chain. Note that this implicitly creates a subcommittee of likely BPs that are aware of their increased odds, so they should be given a limited time to publish their proof.

A second, complementary solution, is to have all proposals (h, π) accepted, and to reuse them over several rounds, by excluding the previously selected participants. For example, we could have the first selection executed between 1000 participants, then the second between 999 participants, then 998 and so one until we have 980, 950 or 900 participants. Only at this point do we re-execute the protocol.

That's the principles. We have not yet implemented it, but we plan to do a PoC of it. Before that, if you identify any potential problems or improvements, we're interested! On paper, we find that this is a solution with acceptable drawbacks for several use cases.

Olivier Bégassat, Nicolas Liochon