Verifiable delay functions have been popular recently as a strategy for getting entropy for validator and committee selection in PoS networks. A VDF is a function that takes some medium-large quantity of non-parallelizable work

to compute, but can be verified very quickly.

The best known “proto-VDF” is the “iterated modular square root” strategy, where f(x) = g(g(g(g(....g(x)....))))

where g(x) = xor(x^((p+1)/4), 1) mod p

. Because each application of g

depends on the result of the previous, the computation cannot be parallelized. Furthermore, g

has an inverse: h(x) = xor(x, 1)^2 mod p

, and this inverse can be computed more quickly (think ~100x more quickly) than g

. Hence, the function f

can be computed in the backward direction ~100x more quickly than in the forward direction, allowing a solution to f

that takes ~5 seconds to compute to take ~0.05 seconds to verify. However, more recently there have been VDFs with much stronger properties, that can be verified almost instantly.

VDFs have the following advantages:

- Relative to RANDAO and similar schemes, they cannot be manipulated.
- Relative to BLS threshold signatures and similar VRFs, they do not depend on any specific fraction of nodes to be online, and do not require a complicated setup procedure.

Now, enter reality. It seems very plausible that there will be one actor who manages to create ASICs for any given VDF, and be N times faster (eg. N = 20) than the top-of-the-line CPU/GPU implementations. The acceleration factor won’t be remotely close to as high as it is for Bitcoin proof of work, because a large portion of the speedup factor in that case comes from parallelization, but one absolutely can imagine ASICs that involve circuits specifically designed to loop back into themselves as fast as possible.

Suppose that this happens. There are two possibilities:

1. The attacker can compute the VDF so quickly that they can predict its output before they have to commit to some value, giving them the ability to choose the result from a set of possible outputs.
2. The attacker can allow the difficulty adjustment process to adjust to their presence, then suddenly go offline, greatly slowing down the system.

Suppose that we use a VDF as follows. A participant is expected to submit data that determines the source data

for a VDF computation, together with other source data that was revealed at time T, and the submission must appear before time T+D (eg. if the participant waits longer than T+D, then they will lose their chance to get included into the canonical chain). The data starts being used at time T+W. Suppose that the difficulty adjustment algorithm is designed in such a way that it targets the data being computed at time T+N.

Let the attacker speedup (“advantage”) over the rest of the network be A. The attacker can perform the first attack if A > N/D, and the second attack if A > W/N. Hence, we can secure against the highest possible A if we set N/D=W/N, ie. N is the geometric mean of D and W (eg. if D = 6 seconds, W = 1 hour, then N ~= 147 sec, allowing the mechanism to resist an attacker advantage up to sqrt(W/D) ~= 24.5.

You may ask: why not have the VDF difficulty adjustment target N to be close to W, and then have a backstop where if the VDF is not calculated by time W, a committee can approve a backstop that allows an easier VDF solution to be used? However, this is problematic: if you are the fastest VDF producer, then you can check the fast and slow solutions, and then choose which one to use by going offline, manipulating the randomness for free. It’s an open problem to see if we can achieve full safety against attacker advantages higher than sqrt(W/D) as above.