TL;DR:

A proposal for a cryptoeconomic vdf scheme that allows the use of proto-vdfs on the beacon chain despite the threat of superfast asics. The scheme has the following properties: Fast verification is possible on asics, challenges to vdf claims can be verifiable quickly by CPU nodes, and the overhead on chain compared to vdf data only is moderate (10s of kilobytes).

proto-VDFs:

Verifiable delay functions are invertible functions of which the inverse is much faster to compute than the function itself. Ideally the factor should be exponential in the number of bits. However, some proto-vdfs only have a linear time factor in the number of bits, for example sloth (https://pdfs.semanticscholar.org/f9b1/d432d2b8098be527f1ece5706b7d328753dc.pdf) and MiMC. These are called proto-vdfs.

One of the challenges with vdfs is that a no-expenses-spared attacker might be able to build an asic that is much faster than a CPU at computing either direction of the vdf. This makes it potentially necessary to develop a commodity asic that comes within a certain range of the best possible asic, in order to avoid that the attacker can use vdf results early to manipulate entropy.

Let's assume that the difficulty of the vdf chosen is $d\approx8000$

, that a commodity asic is able to achieve a speedup $S\approx1000$

vs CPU computation of the vdf, and a no expenses spared asic archieves a $B\approx5$

speedup vs the commodity asic. Further we assume entropy is needed every $t=600\text{s}$

and block time is $b=5\text{s}$

. This means in order to get good safety margins, the vdf target should be $\sqrt{st}\approx60\text{s}$

.

In the scenario without an asic vdf attacker (someone building a superfast asic without sharing it), this means asic nodes can compute in 60s

, verify in $60\text{s}/d\approx0.075\text{s}$

. CPU nodes can verify vdf results in 7.5s. If an attacker with a no-expenses-spared asic is present, they can drive up the difficulty by a factor of $B\approx5$

. Now commodity asic verifiers need 0.05s to verify and pure CPU nodes 37.5s, a similar order as the target computation time. Verifying one days worth of vdf results needs 1.5h CPU time, clearly longer than is reasonable to verify just the beacon chain.

Cryptoeconomic VDF Proposal:

Assuming there are enough commodity asics around, we can use these to relieve the CPU nodes from ever having to compute a full vdf verification.

In order to do this, the claimer of the vdf computation needs to publish not just the vdf result, but also M

hashes of intermediate results. M

should be $\approx S\approx 1000$

and assuming 256 bit hashes this needs 32 kbytes per vdf output which adds 4.6 MB per day to the beacon chain. **

After a vdf result has been published, any commodity asic node can verify it and add its stake to vouch for its correctness. Note it should verify the intermediate hashes as well, as it will also vouch for them.

If a node encounters an incorrect intermediate result hash at position m

while verifying, it publishes the intermediate result at m+1

(thus giving a correct preimage of the $m+1\text{st}$

intermediate result hash).

Any CPU node can verify that the challenge is justified by just checking that the given intermediate result is a preimage of the $m+1\text{st}$

intermediate result hash, and that its vdf preimage is not the preimage of the $m\text{th}$

intermediate result hash. In the worst case assumed above, this would take 37.5\text{s}/M= 0.0375\text{s}

, so it's very cheap for even CPU nodes to do. This means challenges can be instantly verified and we don't need to play elaborate bidding games on the result (as is necessary for the state root claims, see [Delayed state execution in practice])

This challenges the vdf result provider as well as anyone who confirmed its correctness so far, slashing all of them and giving the challenger a fraction of their deposit as a reward.

Implementation on chain:

In practice, we can set up a scheme as follows: After the vdf computation/entropy collection is launched, any validator can post a vdf result (including the intermediate result hashes) and stake their deposit that it is correct. However, the result only becomes "valid" on the chain once a certain number of validators K

have published a confirmation, again staking their deposits on it. Other results/confirmations can be published at the same time. If several vdf results are valid according to this rule, the one with the higher total number of confirmations counts as the current valid one (this should never happen, as it would be very easy for any of them to find the error in the competing results, unless there is an extreme shortage of asic nodes). At the end of the vdf collection time t

, if there is a valid vdf result, it can be used as the new entropy. If there is no valid result, this should be treated the same as there not being a vdf result at all and lead to prolonging the epoch.

Alternatives:

1. Do not store intermediates. In this case, CPU nodes will be relying on claims and counterclaims for their decisions, making vdf verifications only if it can't be settled. A blueprint for this can be found in the discussion on state root claims: Delayed state execution in practice

As the asic gap gets larger, this might lead to a "he said, she said" dilemma where the winner of the verification game might not be the one with the correct vdf output but the one who can stake more money.

1. Do not require the vdf intermediates on chain, but store them off chain instead. While nodes are verifying the non-finalized part of the chain, they will check data availability of the checkpoints. Advantage is that we can potentially store many more checkpoints.

2. Store the full intermediates, probably best used in combination with 2. Then verification becomes possible from any checkpoint, so even CPU nodes can verify vdf claims probabilistically by just computing a few random intermediates. If all CPU nodes do this, they will also find incorrect claims even if all asic nodes collude. (It is even possible to construct a pure cryptoeconomic vdf in this way in the absence of any function with the required asymmetry properties)

** As @JustinDrake pointed out to me, we only need second preimage resistance here, which may be feasible using 128 bit hashes.