Hey guys, had some thoughts on combining the Nakamoto consensus with verifiable delay function (VDF) and verifiable random function (VRF).

The idea is to use VDF to simulate the PoW puzzle solving, and VRF to create puzzle difficulty variations for different validator nodes

. Combining the VDF puzzle with the Nakamoto longest chain rule, validator nodes are effectively chosen at random as the block proposer for each block height. Compared to BFT based consensus which usually poses an upper limit on the number of validator nodes, this approach can achieve a high level of decentralization like Bitcoin, and yet does not burn nearly as much electricity for potentially the same level of security.

Here are the details. For block height i

, the validator nodes compete to solve the following VDF puzzle. Whichever validator first solves the puzzle can extend the chain.

(r_i, \pi_i^r) \leftarrow VRF_{sk}(seed=d_{i-1} || r_{i-1})

(d_i, \pi_i^d) \leftarrow VDF(x=r_i, T=r_i / S)

Here sk

is the secret key of the validator, and S

is the stake (i.e., number of coins) of the validator. r_i

and \pi_i^r

are the random number and its corresponding proof generated by the VRF. Similarly, d_i

and \pi_i^d

are the output for the VDF function and its proof. These four strings, along with the public key of the block proposer, needs to be included in the block header for other validators to verify the correctness of the proposed block.

In the first formula, the VRF function uses the concatenation of d_{i-1}

and r_{i-1}

as the seed

, where both of the two random numbers can be retrieved from the header of the parent block at height i - 1

. The VRF returns r_i

, which is used in the second formula to modulate the difficulty of the VDF puzzle. Thus, each validator node has to solve a puzzle with different difficulty (and hence takes different amount of time).

The second formula uses Pietrzak's iterative squaring VDF construct, which takes two input parameters, x

and the number of iterations T

. We simply set x

to r_i

, and T

to r_i/S

. Note that T

, and therefore the difficulty of the VDF puzzle is inversely proportional to the validator's stake S

. This can effectively prevent the sybil attack

. A user can attempt to create multiple accounts, each with a fraction of his total stake, and solve the corresponding VDF puzzles in parallel. However, it can be proven using the expectation of the minimum of uniform random variables that by doing this, the expected amount of time to solve the VDF puzzle only increases. Thus, the best strategy is to put all the stake in a single account.

We also note that in the first formula, the concatenation d_{i-1} || r_{i-1}

is used as the seed for the VRF. This is to prevent malicious users from quickly evaluating the difficulty for each branch if he wants to fork the chain.

Update:

Actually we can make the VDF puzzle even more similar to the PoW hashing puzzle. Still based on Pietrzak's scheme, the VDF puzzle asks for integer T

, such that

K(H(r_i)^{2^T}) < M

Here K()

is a one-way hash function. Once T

is found, the validator node also outputs d_i = H(r_i)^{2^T}

, and the corresponding proof \pi_i^d

. With the proof, other validators can very quickly verify that (1) d_i

is indeed equal to H(r_i)^{2^T}

, and (2) K(d_i)< M

.

Such a VDF puzzle has the advantage that the validators don't know how long it takes to obtain the solution until the solution is found. This is similar to the PoW hashing puzzle.

However, we note that this VDF puzzle is inherently sequential, so unlike the PoW hashing puzzle, throwing more hardware cannot shorten the solving time.

Also note that the random number r_i

for different validator nodes are not the same. Hence, each validator is solving a different puzzle. And here is where the randomness/lottery comes from.

Thoughts? Any loophole in the proposal? Any feedback is appreciated!