Sequential Proof-of-Work for Fair Staking and Distributed Randomness Beacons

José I. Orlicki, Engineer, cLabs

Abstract—We propose a new Proof-of-Stake consensus protocol based on a Sequential Proof-of-Work constructed with a verifiable random function (VRF) and a verifiable delay function (VDF) that has the following properties: a) all addresses with positive stake can participate; b) is fair because the coin stake is proportional to the distribution of rewards; c) is resistant to several classic blockchain attacks such as Sybil attacks, “Nothing-at-stake” attacks and “Winner-takes-all” attacks. We call it Vixify.

Index Terms—blockchain, proof-of-work, verifiable delay function, verifiable random function, proof-of-stake, distributed consensus, distributed randomness beacon

I. INTRODUCTION

One of the most interesting abstract properties of Nakamoto distributed consensus is that this algorithm is very similar to implementing random clocks (with time inversely proportional to computing power) for the miners with the first clock to stop determining the block proposer. If we can implement this property in a non-parallelizable version we will save a lot of energy wasted on traditional Proof-of-Work and we might open the door also to Fair Proof-of-Stake with an unlimited number of block proposers.

A robust blockchain avoids centralization of stake and mining power, then we want to design a distributed consensus that discourages centralized stake or mining pools, hardware parallelization, energy waste and Sybil-attacks [1], [2]. Also, classic Proof of Stake attack such as nothing-at-stake attacks [3]. Then we want the following properties for our Proof-of-Stake consensus.

The design objectives of this blockchain consensus for Proof of Stake [4] are:

1) **Mining-is-validating**: similarly to Proof-of-Work regarding transaction validation. While mining new blocks nodes are also validating transactions as the same time. Nodes work as both block proposers and validators.

2) **Stake-aligned**: Complete alignment of stake distribution with rewards distribution during consensus (Rewards(Stake) = 0 for Stake = 0).

3) **Independent Aggregation**: Aggregating or separating stake into one or multiple accounts does not change the reward size (Rewards(S_1 + S_1) ≈ Rewards(S_1) + Rewards(S_2)). This property can be separated into the two better known properties following.

- **Sybil-tolerant**: Not susceptible to Sybil attacks, miners spawning multiple parallel block proposers (Rewards(S_1 + S_1) >= Rewards(S_1) + Rewards(S_2)).

- **Pool-neutral**: Aggregating stake into pools does not provide any advantage (Rewards(S_1 + S_1) <= Rewards(S_1) + Rewards(S_2)).

4) **Consensus-scalability**: The consensus remains secure with arbitrary number of nodes.

5) **Permissionless**: Any node can join or leave the consensus at any time.

6) **Fair Mining**: For any mining hardware, its mining speed eventually converges to a single predefined value.

7) **Unbiased**: No adversary can manipulate who generates the next block, even equipped with powerful hardware [5].

8) **Unpredictable**: The probability that the adversary makes an accurate guess on the next block proposer is in proportion to the guessed nodes voting power. The more economic way to predict the next block winner given some stake, is to mine it [3].

9) **Fair Rewards**: Once a miner mines a block, its mining reward is in proportion to its stake.

One the most important properties is number [3] because this property implies that the mining computation cannot be parallelized, i.e. is non-parallelizable.

II. BASIC DEFINITIONS

A. Verifiable random functions

Verifiable Random Functions (VRFs) are now common, such as the one for Elliptic Curve secp256k1, a new standard for VRFs [6], and are defined using a public-key pair sk, pk having the property that using a private key sk allows to hash ones of nodes voting power. The more economic way to predict the next block winner given some stake, is to mine it [3].

A VRF is a triple of algorithms VRFkeygen, VRFeval, and VRFverify:

- **VRFkeygen** \( r \rightarrow (pk, sk) \). On a random input, the key generation algorithm produces a verification key pk and a secret key sk pair.

- **VRFeval** \( (sk, x) \rightarrow (h, \pi) \). The evaluation algorithm takes as input the secret key sk, a message x and produces a pseudorandom output string h and a proof \( \pi \).

- **VRFverify** \( (pk, x, h, \pi) \rightarrow \{0, 1\} \). The verification algorithm takes as input the verification key pk, the message x, the output h and the proof \( \pi \). It outputs 1 if and only if it verifies that pseudo-random output h is the output.
produced by the evaluation algorithm on inputs \( sk \) and \( x \).

VRF functions should satisfy also the properties VRF-Uniqueness, VRF-Collision-Resistance and VRF-Pseudorandomness. In a few words, VRF functions are public-key signing schemes where the signature is unique and pseudo-random.

Because pseudo-random \( h \) outputs can be interpreted as fixed-size integer we can also generate more narrow range integer with modulo. Including integer \( i \) described by Pietrzak and Wesolowski [8], [9], [10]. Also we can be verified much faster, or very fast [7]. They have been proposed as solution to energy inefficient parallelizable Proof-of-Work consensus because of their non-paralelizable properties but they raised some concerns regarding "winner-takes-all" scenarios for nodes with very fast specialized hardware, such as ASIC hardware.

**Definition 1 (Verifiable Delay Function):** A Verifiable Delay Function is a tuple of three algorithms \((\text{VDFSetup}, \text{VDFEval}, \text{VDFVerify})\)

- \( \text{VDFSetup}(\lambda, T) \rightarrow pp \) is a randomised algorithm that takes a security parameter \( \lambda \) and a time bound \( T \), and outputs public parameters \( pp \)
- \( \text{VDFEval}(pp, x) \rightarrow (y, \pi) \) takes an input \( x \) and outputs \( y \) and a proof \( \pi \)
- \( \text{VDFVerify}(pp, x, y, \pi) \rightarrow \{0, 1\} \) outputs 1 if \( y \) is the correct evaluation, otherwise 0.

that satisfies the following three properties

- \( \epsilon \)-evaluation time: \( \text{VDFEval}(pp, x) \) runs in time at most \((1 + \epsilon)T\) for all \( x \) and \( pp \) output by \( \text{VDFSetup}(\lambda, T) \).
- Sequentiality: No adversary \( A \) using at most \( \text{poly}(\lambda) \) processors can compute \( \text{VDFEval} \) in time less than \( T \).
- Uniqueness: For any adversary \( A \):
  \[
  Pr \left[ \begin{array}{c}
  \text{VDFEval}(pp, x) = y \\
  \text{VDFVerify}(pp, x, y, \pi) = 1 \\
  pp \leftarrow \text{VDFSetup}(\lambda, T) \\
  (x, y, \pi) \leftarrow A(\lambda)
  \end{array} \right] \leq \text{negl}(\lambda)
  \]

Practical VDFs with current working implementations were described by Pietrzak and Wesolowski [8], [9], [10]. Also a pseudo-VDF that does not scales shows and interesting asymmetry between proving and validating the proof is Sloth [2].

### III. Consensus

Although we discuss a non-prefixed number of steps VDF puzzle in this section (Algorithm 3), the final consensus proposed is not search for an output bigger than a difficult but just generated a VDF proof using a pre-determined number of steps cased on a VRF proof, i.e. a personalized random seed. Then we are most interested in the case of sequential Proof-of-Work based on randomized number of steps.

### A. Block structure

A special block structure with subblock independant of VDF outputs and inputs. We segregate the fields that are dependent on the Merkle tree root chosen by the miner so that these fields are not inputs of the linear puzzles that decide who is the block proposer. This way any miner cannot control Merkle tree root to generate many parallel copies of linear VDF mining and increase its chances of being selected as block proposer.

**Algorithm 1:** Proof-of-Work puzzle from miner \( k \)'s perspective [2].

**Input:**

**Output:**

\[
\begin{align*}
  t & \leftarrow 0 \\
  \text{while } \text{True do} \\
  & \quad \text{hash}_{N,k}(t) \leftarrow H(\text{merkle}_{N,k}, H_{N-1}, t) \\
  & \quad \text{if } \text{hash}_{N,k}(t) > T_N \text{ then} \\
  & \quad \quad \text{Nonce}_{N,k} \leftarrow t \\
  & \quad \quad \text{return } \text{Nonce}_{N,k} \\
  & \quad t+ = 1 \\
\end{align*}
\]

From the binary point of view, we are searching for a VDFStep() output with a number of leading 1s plus other binary conditions on the rest of the bits. Each block proposer is mining VDFStep() until they find the first nonce big enough to satisfy the conditions including the RandomSlot.

So, current block proposer will be the one with the small number of steps satisfying their specific nonce restriction (there is no global condition for nonces):

\[
\text{Proposer}_N \leftarrow \{ k \in \text{Miners} : \forall r \in \text{Miners} : \text{Nonce}_{N,k} \leq \text{Nonce}_{N,r} \}
\]
Algorithm 2: Vixify: sequential Proof-of-Work puzzle with pre-calculated number of VDF steps (no need for continuous VDF in this case)

Input: 
Output:
\[ \text{in}^{\text{VRF}} \leftarrow H_{\text{ash}}AB(N - 1) \oplus \text{MerkleRoot}(N) \]
\[ \text{in}^{\text{VDF}} \leftarrow \text{VRFEval}(sk_k, \text{in}^{\text{VRF}}) \]
\[ \text{Hash}_{N,k}(t) \leftarrow \text{VDFStep}^{\text{VDFInput}_{N,k}, t} \text{ for } t \geq 0 \]
\[ \text{Range}_k \leftarrow \lceil \frac{1}{S_k} \rceil \]
\[ \text{Slot}_{N,k} \leftarrow \text{VRFEval}(sk_k, H'_{N-1}) \mod \text{Range}_k \]
\[ \text{Nonce}_{N,k} \leftarrow \text{Hash}_{N,k}(Q_N R_N^{\text{RandomSlot}_{N,k}}) \]

Algorithm 3: Sequential Proof-of-Work puzzle with continuous VDF.

Input: 
Output:
\[ \text{VDFInput}_{N,k} \leftarrow \text{VRFEval}(sk_k, H_{\text{ash}}A(N - 1)) \]
\[ \text{Hash}_{N,k}(t) \leftarrow \text{VDFStep}^{\text{VDFInput}_{N,k}, t} \]
\[ \text{Range}_k \leftarrow \lceil \frac{1}{S_k} \rceil \]
\[ \text{RandomSlot}_{N,k} \leftarrow \text{VRFEvalInt}(sk_k, H'_{N-1}, \text{Range}_k) \]
\[ \text{Nonce}_{N,k} \leftarrow \min\{ t : \text{Hash}_{N,k}(t) > Q_N R_N^{\text{RandomSlot}_{N,k}} \} \]

Given the rare case there is more than one proposer per block, we can choose the one with the smaller number for steps. If there is also a collision on the number of steps then we can choose randomly based on \text{VDFInput}_{N,k} that is pseudorandom.

Difficulty is personalized for the stake of the miner and is dynamic. For example, if the current pseudorandom discrete slot of the exponent is 0, then we have the linear mining for the first slot as:

\[ \min_t \{ \text{VDFStep}^{\text{VDFInput}_{N,k}, t} > Q_N \} \]

were \( Q_N \) is a discrete quantum, same for all miners, that is dynamically adjusted each block based on average block time considering a large number of previous blocks.

Also \( B_N \) is the same for all miners on the current block, and allows an exponential adjustment that can protect the consensus from persistent strong hardware or strong optimized software attacks, miners with a VDF speed substantially faster than the rest of the miners.

D. Dynamic Difficulty

There is no question that stake fragmentation increases block time because having many miners with little stake allows only more difficult puzzles than a small number of miners with large stake portions. If average block time get bigger we assume this is because the stake fragmentation has increased. Then is enough to reduce the difficulty linearly. The base quantum of VDF difficulty \( Q_N \) is reduced by a very fraction \( \alpha_N \) dynamic as per every block. This tendency can be reverted if stakes are being consolidated at some point, then we need increase linearly \( Q_N \) (See Algorithm 4). With these changes we want an stable average block time around a fixed number of seconds.

Algorithm 4: Vixify difficulty adjustment for average block time. Similar to traditional Proof-of-Work.

Input: \( N \) block number, \( Q_N \) current block time difficulty, \( \alpha \) fractional change per block, \( A^0 \) target block time, \( A \) current moving windows average block time for fixed windows size \( a \)

Output: \( Q_{N+1} \)

\[ \text{if } A >= A^0 \text{ then } Q_{N+1} \leftarrow Q_N \ast (1 - \alpha) \]
\[ \text{else } Q_{N+1} \leftarrow Q_N \ast (1 + \alpha) \]

Average block time is also influenced by VDF speed, average number of VDF steps per second. We need to account for that also using timestamps in blocks. If average VDF steps per second get smaller we assume that this is because hardware or software optimizations have made the VDF computation faster. This tendency can be economically reverted if big stakeholders sell their stake and stop mining, then is bi-directional. Then we can use the maximum speed to date as the reference. Then the algorithmic adjustment for this will be exponential on the exponential base \( R_N \) only in the fractional increment \( \beta_N \) if the maximum speed has been surpassed (see Algorithm 5). Because there is not target for VDF speed then we must choose a big moving average window \( b \) for VDF speed and a small change fraction \( \beta \) so this difficulty \( R_N \) moves very slowly. This allows miners investing in faster VDF hardware or software profitable for small time span like minutes or hours, allowing a miner \( k \) on block \( N \) to jump from one random slot \( \text{VRFEvalInt}(sk_k, H'_{N-1}, [1/S_k]) \) to a smaller one only for such a small time.

Algorithm 5: Vixify difficulty adjustment for average VDF steps per second. Affects slots exponentially to quickly reduce optimization advantage of any miner.

Input: \( N \) block number, \( R_N \) current block VDF speed difficulty, \( \beta \) fractional change per block, \( B_N \) current moving windows average VDF steps per second for fixed windows size \( b \)

Output: \( B_{N+1} \)

\[ \text{if } B_N >= B_{N-1} \text{ then } R_{N+1} \leftarrow R_N \ast (1 - \beta) \]
\[ \text{else } R_{N+1} \leftarrow R_N \ast (1 + \beta) \]

Block timestamps can be manipulated by miners but in a very limited way. If they lie and produce bigger timestamps, other peers will detect that and will not propagate those blocks, if the produce smaller timestamps it will increase the difficulty then making mining harder for everyone including themselves. Same with VDF speed, because the only slack for miners is
distorting timestamps (they cannot distort the VDF number of steps) without risking losing the opportunity to propose a winning block.

E. Distributed Randomness Beacons

A Distributed Randomness Beacon (DRB) is a stripped-down type of blockchain where there is no payload outside the minimal components of the protocol and there is a focus on using the hash of the blocks as a reliable source of randomness, a source of entropy [11]. These pseudo-random numbers generated can be used by other layers of a complex protocol to do random-coin tossing for leader or validator set selection [12].

A simplified version of Algorithm 2 can be applied to implement a Distributed Randomness Beacon (DRB), see Algorithm 6. The exponential protection for fair staking is removed because we assume the minimal case of a DRB without block rewards. So without staking and rewards there is not need to protect rewards fairness. The focus is on using for each miner a different randomized number of steps for the VDF. For randomization we used the VRF seed generated for the private key of the miner.

Algorithm 6: Vixify linear puzzle with pre-calculated number of VDF steps (no need for continuous VDF in this case)

\[
\begin{align*}
\text{Input:} & \\
\text{Output:} & \\
\text{in}^{VDF}_{N,k} & \leftarrow \text{VRFEval}(sk_k, HashAB(N - 1)) \\
\text{Hash}_{N,k}(t) & \leftarrow \text{VDFEval}(\text{in}^{VDF}_{N,k}, t) \text{ for } t \geq 0 \\
\text{Range}_k & \leftarrow \left\lfloor \frac{1}{S_k} \right\rfloor \\
\text{Slot}_{N,k} & \leftarrow \text{VRFEval}(sk_k, H'_N) \mod \text{Range}_k \\
\text{Nonce}_{N,k} & \leftarrow \text{Hash}_{N,k}(Q_N R_{Slot_{N,k}})
\end{align*}
\]

IV. VERIFICATION

A. Mining-is-validating

Proof: The Merkle-tree root hash is a cryptography digest of the transactions payload. This hash is included as part of the input of the VDF function for each block and for each miner, see Algorithm 2. This proves that the each valid block proposed to be accepted must be a block with a valid set of transactions.

B. Stake-aligned

Proof: This is trivially true because it can be checked that if the stake of the miner is zero it cannot propose any valid block. Otherwise we will find a divided-by-zero error when calculating the miner slot $1/S_k$.

C. Independent Aggregation

The most important property for this Proof-of-Stake consensus is Independent Aggregations (i.e. of stake). The proof of this Property 3 of Proof-of-Stake (Section I) can be sketched in the following way:

Proof:

1) Prove Property 3 for stakes of the form $1/2^k$ that are split into two ($\geq$, Sybil-tolerant).
2) Prove Property 3 for stakes of the form $1/2^k$ that are aggregated ($\leq$, Pool-neutral).
3) Prove Property 3 for all stake because they have the form $\sum_{k \in 1/2^k}$...

D. Consensus-scalability and Permission-less

Proof: Our distributed consensus is very similar to Nakamoto consensus so these properties are directly satisfied.

We are not using a Voting Committee as other protocol, then, there is not limit to the number of miners proposing blocks as long as they have a positive balance of coins, also known as stake.

Any fraction of nodes with any given fraction of stake can leave the consensus at any time and the protocol is robust to continue operating with a block time that will be bigger for some time.

E. Fair Mining

This property is proved based on the dynamic exponential difficulty adjustment based on the current mean VDF speed (see Algorithm 5).

F. Unbiased and Unpredictable

Proof: Sketch: is very similar to Nakamoto Consensus but in this case the probability of being the first to propose a block and win is proportional to the stake of the miner.

G. Fair Rewards

Block proposer sequential difficulty (number of VDF steps) is based on slots $1/S_k$ determined by stake $S_k$. This is designed to be a very good approximation of stake itself.

V. SECURITY ANALYSIS

A. Nothing-at-Stake

Miners cannot generate new addresses to do mining on each block, because each new address requires a number of staked coins on the address balance, then due to Property 3 of Proof-of-Stake (Section I) there is not rewards gain in splitting the stake into several addresses.

B. Winner-takes-all

The exponential difficulty parameter (Algorithm 2) and its slow but steady adjustment (Section I) makes software or hardware optimizations of VDF speed only profitable for a short time. After this short time of adjustment is very difficult to jump from the assigned slot to a smaller one.
VI. CONCLUSION

In summary, this paper addresses using a Sequential Proof-of-Work Consensus, called Vixify. It is based on a verifiable delay function (VDF) and verifiable random function (VRF) to simulate a distributed consensus very similar to Nakamoto Consensus but energy-efficient, fair with stakes, and resistant to Sybil attacks and hardware optimizations.

This distributed consensus we proposed has satisfied the abstract property of Nakamoto Consensus of simulating random clocks running on each miner but without the possibility of parallelizing the computation, then the timer for each miner is not inversely proportional to their computing power but directly proportional to their stake in coins.

ACKNOWLEDGMENT

The authors would like to thank Santiago Bazerque from HyperHyperSpace for interesting conversations on Blockchain Scaling and Sharding that motivated the formalization of this distributed consensus algorithm.

REFERENCES

[1] J. Douceur, “The sybil attack,” 01 2002, pp. 251–260.
[2] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2009. [Online]. Available: [http://www.bitcoin.org/bitcoin.pdf]
[3] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, “A survey on long-range attacks for proof of stake protocols,” IEEE Access, vol. PP, 02 2019.
[4] S. Zhang and J.-H. Lee, “Analysis of the main consensus protocols of blockchain,” ICT Express, vol. 6, no. 2, pp. 93 – 97, 2020. [Online]. Available: [http://www.sciencedirect.com/science/article/pii/S240595951930164X]
[5] D. Boneh, S. Easkandarian, L. Hanzlik, and N. Greco, “Single secret leader election,” Cryptology ePrint Archive, Report 2020/025, 2020, [https://eprint.iacr.org/2020/025](https://eprint.iacr.org/2020/025)
[6] Goldberg, S. and Reyzin, L., “Verifiable Random Functions (VRFs) draft-irtf-cfrg-vrf-05," online, accessed 20/08/2020, August 2019, [https://tools.ietf.org/pdf/draft-irtf-cfrg-vrf-05.pdf](https://tools.ietf.org/pdf/draft-irtf-cfrg-vrf-05.pdf).
[7] D. Boneh, J. Bonneau, B. Bnz, and B. Fisch, “Verifiable delay functions,” in Advances in Cryptology CRYPTO 2018, ser. Lecture Notes in Computer Science, vol. 10991. Springer, 2018, pp. 757–788.
[8] K. Pietrzak, “Simple verifiable delay functions,” Cryptology ePrint Archive, Report 2018/627, 2018, [https://eprint.iacr.org/2018/627](https://eprint.iacr.org/2018/627)
[9] B. Wesolowski, “Efficient verifiable delay functions,” vol. 11478, pp. 379–407, 2019.
[10] Chia Network Inc and POA Networks Ltd, “Verifiable Delay Function (VDF) Implementation in Rust,” online, accessed 20/08/2020, 2018, [https://github.com/poanetwork/vdf](https://github.com/poanetwork/vdf).
[11] B. B. unz, S. Goldfeder, and J. Bonneau, “Proofs-of-delay and randomness beacons in Ethereum,” in IEEE Security & Privacy on the Blockchain (IEEE S&B), 2017.
[12] Ben Edgington, “State of Ethereum Protocol #2: The Beacon Chain,” online, accessed 20/08/2020, October 2018, [https://media.consensys.net/state-of-ethereum-protocol-2-the-beacon-chain-c6b6a9a69129](https://media.consensys.net/state-of-ethereum-protocol-2-the-beacon-chain-c6b6a9a69129).

José I. Orlicki received the 6-year B.S.C.S degree from Universidad de Buenos Aires, Buenos Aires, CABA, in 2006, and a MSc in Financial Engineering degree from the Stevens Institute of Technology, Hoboken, NJ, in 2017. From 2006 to 2009, he was a Computer Security Reserarch with Core Security Technologies, Buenos Aires, CABA, where he was involved with network attack simulations and automated attack planning. His current research interests include algorithmic trading, cryptoeconomics, stablecoins and distributed consensus. He is currently employed by CLabs, working as part of the permanent Protocol Economics Team of the Celo Blockchain.