RANDCHAIN: Decentralised Randomness Beacon from Sequential Proof-of-Work

Runchao Han*†, Jiangshan Yu*¶, Haoyu Lin‡§
*Monash University, {runchao.han, jiangshan.yu}@monash.edu
†CSIRO-Data61
‡Bytom Foundation, chris.haoyul@gmail.com
§ZenGo X

***Abstract*— Decentralised Randomness Beacon (DRB) is a service that generates publicly verifiable randomness. Constructing DRB protocols is challenging. Existing DRB protocols suffer from either strong network synchrony assumptions, high communication complexity or various attacks.**

In this paper, we propose RANDCHAIN, a new family of (permissioned) DRB protocols. To construct RANDCHAIN, we first introduce Sequential Proof-of-Work (SeqPoW), a Proof-of-Work (PoW) variant that is *sequential*, i.e., the work can only be done by a single processor. In RANDCHAIN, nodes jointly maintain a *blockchain*, i.e., a chain of blocks, and each block derives a random output. Each node derives a unique SeqPoW puzzle from the last block and its identity, and keeps *mining*, i.e., solving the SeqPoW puzzle, to append a block to the blockchain. This makes mining *non-parallelisable*. RANDCHAIN applies Nakamoto consensus so that nodes agree on a unique blockchain.

While inheriting simplicity and scalability from Nakamoto consensus, RANDCHAIN produces strongly unpredictable randomness and remains energy-efficient and decentralised. RANDCHAIN does not allow nodes to input local entropy, thus giving no opportunity to bias randomness. Solutions of SeqPoW puzzles are unpredictable, so nodes cannot predict randomness. As each node can use at most a single processor for mining, RANDCHAIN remains energy-efficient. SeqPoW mining can only be accelerated by increasing processors' clock rate, which is bound by processors' voltage limit. Therefore, powerful nodes can only achieve limited speedup compared to normal nodes, leading to a high degree of mining power decentralisation.

1. INTRODUCTION

Randomness beacon (RB) is a service that periodically generates publicly verifiable randomness – an essential building block for various cryptographic primitives. RB is not always reliable: malicious RBs may generate biased randomness to compromise the security and/or fairness of randomness-based cryptographic primitives. To make RB trustworthy, a promising approach is the decentralised randomness beacon (DRB). In a DRB, a group of participants (aka. nodes) generate randomness that they agree on. No party can fully control the generated randomness, so biasing or predicting the randomness can be hard. DRBs have been integrated into various protocols, especially blockchains [1]–[4].

However, constructing secure and scalable DRBs is challenging, mainly due to four obstacles. First, DRBs may execute in rounds so rely on the *lock-step synchrony*, where all messages are delivered at the end of each round. *Lock-step synchrony* is a strong assumption, and is considered unrealistic in real-world networks with delays [5], [6]. Second, DRBs may suffer from high communication complexity of no less than $O(n^2)$, where n is the number of nodes [1], [7]–[9]. Third, DRBs can be vulnerable [2], [10]–[12], including being biasible, i.e., the adversary can choose its preferred random outputs, and being predictable, i.e., the adversary can predict randomness in the future. Last, DRBs may rely on trustworthy components such as full-fledged blockchains and smart contracts [11]–[15], which are not always reliable.

In this paper, we propose RANDCHAIN– a new family of DRB protocols. RANDCHAIN is constructed from Sequential Proof-of-Work (SeqPoW) and Nakamoto consensus [16]. Sequential Proof-of-Work (SeqPoW) is a Proof-of-Work (PoW) variant that is *sequential*, i.e., the work can only be done by a single processor. In RANDCHAIN, nodes jointly maintain a *blockchain*, i.e., a chain of blocks, and each block derives a random output. Each node derives a unique SeqPoW puzzle from its identity and the last block, and keeps *mining*, i.e., solving the SeqPoW puzzle, to append a block to the blockchain. This fixes each node's SeqPoW puzzle and makes mining *non-parallelisable*. RANDCHAIN applies Nakamoto consensus so that nodes agree on a unique blockchain.

While inheriting simplicity and scalability from PoW-based consensus, RANDCHAIN produces secure randomness and remains energy-efficient and decentralised. RANDCHAIN does not allow nodes to input local entropy, thus giving no opportunity to bias randomness. Solutions of SeqPoW puzzles are unpredictable, so nodes cannot predict randomness. As each node can use at most a single processor for mining, RANDCHAIN remains energy-efficient. SeqPoW mining can only be accelerated by increasing processors' clock rate, which is bound by processors' voltage limit. Therefore, powerful nodes can only achieve limited speedup compared to normal nodes, leading to a high degree of mining power decentralisation.

Compared to existing DRBs, RANDCHAIN makes three unique design choices. First, nodes in RANDCHAIN are *competitive*, i.e, each node tries to produce the next random output before other nodes, whereas nodes in existing DRBs are usually *collaborative*, i.e., nodes contribute their local random inputs and combine them to a unique output. Second, RANDCHAIN reuses random entropy from bootstrap, whereas existing DRBs usually allow nodes to sample new random

¶ Corresponding author.

entropy for every round. Last, RANDCHAIN uses uncontrollable random entropy such as nodes' Byzantine behaviours and network delay, whereas existing DRBs usually allow nodes to choose their own inputs. Concretely, we make the following contributions.

Sequential Proof-of-Work (SeqPoW). We propose SeqPoW, a PoW [17] variant that is *sequential*, i.e., the work can only be done by a single processor. To solve a PoW puzzle, the prover should find a string (aka. nonce) that makes the PoW output to satisfy a difficulty parameter. The prover's only strategy is to brute-force search all possible nonces. The searching process can be parallelised using multiple processors. Meanwhile in SeqPoW, the prover keeps incrementing an *iteratively sequential function* with a given nonce as input, until finding an intermediate output that satisfies the difficulty parameter. Given the iteratively sequential function, the prover cannot accelerate solving SeqPoW by parallelism. Instead, the prover can only accelerate solving SeqPoW by increasing the processor's clock rate, which gives little optimisation space.

We formalise SeqPoW and propose two SeqPoW constructions, one is from Verifiable Delay Functions (VDFs) [18] and the other is from the Sloth hash function [19]. We formally analyse the security and efficiency of both SeqPoW constructions. We also implement them and evaluate their performance. Of independent interest, SeqPoW is a powerful primitive and can be applied to various protocols. We discuss potential applications of SeqPoW, including leader election and Proof-of-Stake (PoS)-based consensus.

RANDCHAIN**– DRB from SeqPoW and Nakamoto consensus.** We then construct RANDCHAIN from SeqPoW and Nakamoto consensus. RANDCHAIN works in a *permissioned* network: there is a fixed group of nodes, and each node has a pair of secret key and public key. Nodes jointly maintain a *blockchain*, i.e., a chain of blocks, and each block derives a random output. Similar to PoW-based blockchains, new blocks are generated via *mining*. To append a block, a node should solve a SeqPoW puzzle. Each node derives its SeqPoW puzzle from the last block and its identity in the system. This makes each node's SeqPoW unique. Given a unique SeqPoW puzzle, mining in RANDCHAIN is non-parallelisable. Nodes follow Nakamoto consensus in order to agree on a unique blockchain.

We present the detailed construction of RANDCHAIN and analyse its security. Our analysis shows that RANDCHAIN implements a DRB protocol and produces strongly unpredictable randomness. In addition, we show that RANDCHAIN achieves energy-efficiency and high mining power decentralisation. We also discuss several security concerns of deploying RANDCHAIN in the real world.

Paper organisation. Section 2 describes the background. Section 3 introduces SeqPoW, including definitions, constructions and analysis. Section 4 and 5 describe the construction and security analysis of RANDCHAIN, respectively. Section 6 discusses other security concerns in RANDCHAIN. Section 7 compares RANDCHAIN with existing RBs. Section 8 concludes this paper and discusses future work. We defer some security proofs of our SeqPoW constructions to Appendix A.

TABLE I: Summary of notations. The first section includes frequently used notations. The second and the last sections include notations for Sequential Proof-of-Work and RANDCHAIN, respectively.

Notation	Description
$H(\cdot)$, $H'(\cdot)$	Hash functions mapping $\{0,1\}^*$ to $\{0,1\}^\kappa$
$\mathcal{X}, \mathcal{Y}$	Domains
pp	Public parameter
t	Time parameter of VDFs ($t \in \mathbb{N}^+$)
π	Proof
ψ	Step parameter of SeqPoW ($\psi \in \mathbb{N}^+$)
T	Difficulty parameter of (Seq)PoW ($T \in (1, \infty)$)
sk, pk	Secret key and public key
G, g	Cyclic group and its generator
$H_G(\cdot)$	Hash function mapping $\{0,1\}^*$ to an element on G
S_i, b_i	The i-th SeqPoW solution and its validity
π_i	Proof of i-th SeqPoW solution
$\{p_1, \ldots, p_n\}$	Nodes in RANDCHAIN
sk_k, pk_k	Secret key and public key of node p_k
$\mathcal{C}_k$	The local blockchain of node p_k
$\mathcal{MC}_k$	The the main chain of node p_k
B_ℓ	The ℓ-th block
$\alpha, \beta \in (0,1)$	Mining power portion of Byzantine and correct nodes

2. PRELIMINARIES

Notations. Table I summarises notations in this paper.

PoW and PoW-based consensus. Proof-of-Work (PoW) is a computationally intensive puzzle. To solve a PoW puzzle, a prover should finish an amount of computation. Given a PoW's solution, the verifier can verify whether the prover has finished that amount of computation with negligible overhead. PoW can be constructed from hash functions. As shown in Figure 1, the prover needs to find a nonce x which makes $H(in\|x) \leq \frac{2^\kappa}{T}$, where $H : \{0,1\}^* \to \{0,1\}^\kappa$ is a cryptographic hash function, in is the puzzle, and $T \in (1, \infty)$ is the difficulty parameter. As hash functions produce pseudorandom outputs, brute-force searching is the prover's only strategy. Statistically, with a larger T, the prover needs to try more nonces.

PoW-based consensus – first introduced in Bitcoin [16] – has become a practical and scalable alternative of traditional Byzantine Fault Tolerant (BFT) consensus. In PoW-based consensus, nodes jointly maintain a blockchain. To append a block to the blockchain, a node should solve a computationally hard PoW puzzle [20]. Given the latest block, each node derives its own PoW puzzle then keeps trying to solve it. Once solving the puzzle, the node appends its block to the blockchain. PoW specifies the amount of work needed by a difficulty parameter. By adaptively adjusting the PoW's difficulty parameter, PoW-based consensus ensures only one node solves the puzzle for each time period with high probability. Nodes in PoW-based consensus are known as *miners*, the process of solving PoW

puzzles is known as *mining*, and the computation power used for mining is known as *mining power*.

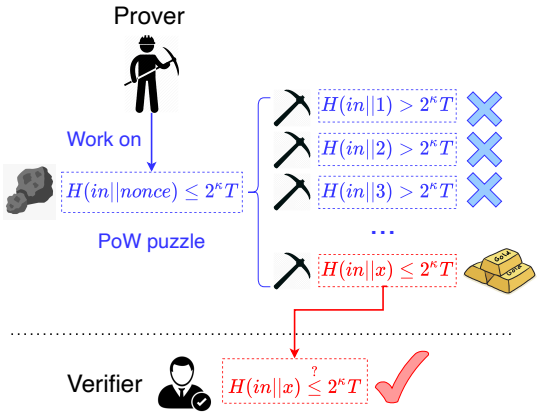


Fig. 1: Proof-of-Work.

Iteratively sequential functions. A sequential function $g(\cdot)$ is that, one cannot accelerate computing $g(\cdot)$ by parallelism. An iteratively sequential function $f(t, x)$ is implemented by composing a sequential function $g(x)$ for a number t of times, denoted as $g^t(x)$. The iteratively sequential function $f(\cdot)$ inherits the sequentiality of $g(\cdot)$: the fastest way of computing $f(t, x)$ is to iterate $g(x)$ for t times. In addition, $f(\cdot)$ preserves a useful property called *self-composability*: for any x and (t_1, t_2), let $y \leftarrow f(x, t_1)$, we have $f(x, t_1 + t_2) = f(y, t_2)$. Commonly used iteratively sequential functions include repeated squaring [21], [22] and repeated square rooting [19] on cyclic groups.

Time-based cryptography and VDFs. Iteratively sequential functions are the key ingredient of time-based cryptographic primitives. Rivest, Shamir and Wagner first introduce the time-lock puzzle and constructs it from repeated squaring on RSA groups [23]. Proofs of Sequential Work (PoSW) [24] allow to prove he has finished an amount of sequential computation with succinct proof. Verifiable Delay Functions (VDFs) [18] can be seen as PoSW with *uniqueness*. After finishing the sequential computation, the prover in addition produces an output that is *unique*, i.e., it's computationally hard to find two inputs that give the same output. Formally,

Definition 1 (Verifiable Delay Function)**.** A Verifiable Delay Function VDF is a tuple of four algorithms

$$\mathsf{VDF} = (\mathsf{Setup}, \mathsf{Eval}, \mathsf{Prove}, \mathsf{Verify})$$

$\mathsf{Setup}(\lambda) \rightarrow pp$: On input security parameter λ, outputs public parameter pp. Public parameter pp specifies an input domain $\mathcal{X}$ and an output domain $\mathcal{Y}$. We assume $\mathcal{X}$ is efficiently sampleable.

$\mathsf{Eval}(pp, x, t) \rightarrow y$: On input public parameter pp, input $x \in \mathcal{X}$, and time parameter $t \in \mathbb{N}^+$, produces output $y \in \mathcal{Y}$.

$\mathsf{Prove}(pp, x, y, t) \rightarrow \pi$: On input public parameter pp, input x, and time parameter t, outputs proof π.

$\mathsf{Verify}(pp, x, y, \pi, t) \rightarrow \{0, 1\}$: On input public parameter pp, input x, output y, proof π and time parameter t, produces 1 if correct, otherwise 0.

VDF satisfies the following properties

- *Completeness*: For all λ, x and t,

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp, x, y, \\ \pi, t) = 1 \end{array} \middle| \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ y \leftarrow \mathsf{Eval}(pp, x, t) \\ \pi \leftarrow \mathsf{Prove}(pp, x, y, t) \end{array}\right] = 1$$

- *Soundness*: For all λ and adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp, x, y, \pi, t) = 1 \\ \wedge \mathsf{Eval}(pp, x, t) \neq y \end{array} \middle| \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ (x, y, \pi, t) \leftarrow \mathcal{A}(pp) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

- *σ-Sequentiality*: For any λ, x, t, $\mathcal{A}_0$ which runs in time $O(\mathsf{poly}(\lambda, t))$ and $\mathcal{A}_1$ which runs in parallel time $\sigma(t)$,

$$\Pr\left[\begin{array}{c|c} \mathsf{Eval}(x, y, t) = y \end{array} \middle| \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ \mathcal{A}_1 \leftarrow \mathcal{A}_0(\lambda, t, pp) \\ y \leftarrow \mathcal{A}_1(x) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

VDFs can be constructed from two ingredients: 1) an iteratively sequential function $f(\cdot)$, and 2) a succinct proof on $f(\cdot)$'s execution results. For example, two prevalent VDFs [21], [22] employ repeated squaring as the iteratively sequential function. Such VDFs inherit the *self-composability* from iteratively sequential functions, and are known as *self-composable VDFs* [25].

Definition 2 (Self-Composability)**.** A VDF $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Prove}, \mathsf{Verify})$ satisfies self-composability if for all λ, x, (t_1, t_2),

$$\Pr\left[\begin{array}{c|c} \mathsf{Eval}(pp, x, t_1 + t_2) \\ = \mathsf{Eval}(pp, y, t_2) \end{array} \middle| \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ y \leftarrow \mathsf{Eval}(pp, x, t_1) \end{array}\right] = 1$$

Lemma 1. *If a VDF* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Prove}, \mathsf{Verify})$ *satisfies self-composability, then for all* λ, x, (t_1, t_2),

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp, x, y', \\ \pi, t_1 + t_2) = 1 \end{array} \middle| \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda) \\ y \leftarrow \mathsf{Eval}(pp, x, t_1) \\ y' \leftarrow \mathsf{Eval}(pp, y, t_2) \\ \pi \leftarrow \mathsf{Prove}(pp, x, y', t_1 + t_2) \end{array}\right] = 1$$

3. Sequential Proof-of-Work

In this section, we introduce Sequential Proof-of-Work (SeqPoW), a variant of PoW that is sequential and cannot be solved faster by parallelism. We formalise SeqPoW, provide two constructions, analyse their security and evaluate their performance and efficiency.

1. Intuition and applications

Figure 2 gives the intuition of SeqPoW. Given an initial SeqPoW puzzle S_0, the solver keeps solving the puzzle by incrementing an iteratively sequential function. Each solving attempt takes the last output S_{i-1} as input to produce a new output S_i. For each output S_i, the prover checks whether S_i satisfies a difficulty parameter T. If yes, then S_i is a valid

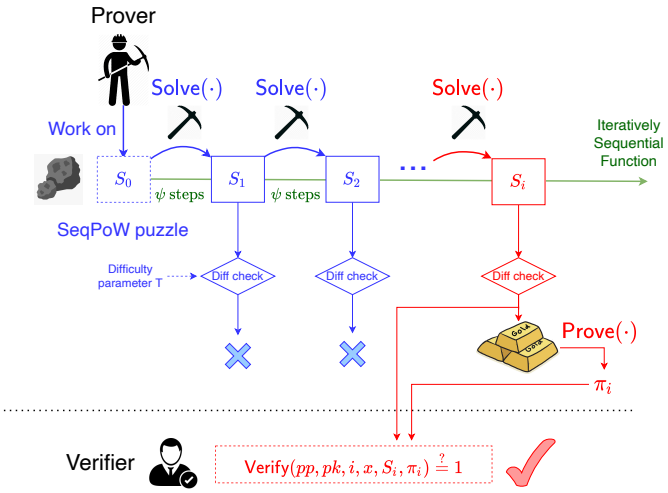


Fig. 2: Sequential Proof-of-Work.

solution of this SeqPoW puzzle. The prover can generate a proof π_i on a valid solution S_i, and the verifier with S_i and π_i can check S_i's correctness without solving the puzzle again. Due to the data dependency, the prover cannot solve a SeqPoW puzzle in parallel.

SeqPoW is the main ingredient of RANDCHAIN. Of independent interest, SeqPoW can also be applied to other protocols, such as leader election and PoS-based consensus.

Leader election. In PoW-based consensus, PoW mining can be seen as a way of electing leaders: given a set of nodes, the first node proposing a valid PoW solution becomes the leader and proposes a block. As a drop-in replacement of PoW, SeqPoW can be used for leader election. Compared to PoW-based leader election, SeqPoW-based leader election is much fairer and energy-efficient. Each node can run at most a single processor, which prevents mining marketisation and gives little optimisation space of mining speed. Later in §5-D we will discuss this in detail.

PoS-based consensus. In Proof-of-Stake (PoS)-based consensus [26], each node's chance of mining the next block is in proportion to its *stake*, e.g, the node's balance. In some PoS-based consensus designs [27], each node solves a VDF with time parameter inversely proportional to its stake, and the first node solving its VDF proposes the next block. However, such designs suffer from the "winner-takes-all" problem, where the node with most stake always solves VDFs faster than others. A way of addressing the "winner-takes-all" problem is to randomise VDFs' time parameters, so that the node with most stake does not always win, but only with a higher chance [28], [29]. SeqPoW provides an alternative approach that, one can replace VDF with SeqPoW and specify the difficulty parameter in inverse proportion to each node's stake. This also randomises the time of solving puzzles.

2. Definition

We formally define SeqPoW as follows.

Definition 3 (Sequential Proof-of-Work (SeqPoW))**.** A Sequential Proof-of-Work SeqPoW is a tuple of algorithms

$$\mathsf{SeqPoW} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Init}, \mathsf{Solve}, \mathsf{Verify})$$

$\mathsf{Setup}(\lambda, \psi, T) \to pp$: On input security parameter λ, step $\psi \in \mathbb{N}^+$ and difficulty $T \in (1, \infty)$, outputs public parameter pp. Public parameter pp specifies an input domain $\mathcal{X}$, an output domain $\mathcal{Y}$, and a cryptographically secure hash function $H : \mathcal{Y} \to \mathcal{X}$. We assume $\mathcal{X}$ is efficiently sampleable.

$\mathsf{Gen}(pp) \to (sk, pk)$: Probabilistic. On input public parameter pp, produces a secret key $sk \in \mathcal{X}$ and a public key $pk \in \mathcal{X}$.

$\mathsf{Init}(pp, sk, x) \to (S_0, \pi_0)$: On input public parameter pp, secret key sk, and input $x \in \mathcal{X}$, outputs initial solution $S_0 \in \mathcal{Y}$ and proof π.

$\mathsf{Solve}(pp, sk, S_i) \to (S_{i+1}, b_{i+1})$: On input public parameter pp, secret key sk, and i-th solution $S_i \in \mathcal{Y}$, outputs $(i+1)$-th solution $S_{i+1} \in \mathcal{Y}$ and result $b_{i+1} \in \{0, 1\}$.

$\mathsf{Prove}(pp, sk, i, x, S_i) \to \pi_i$: On input public parameter pp, secret key sk, i, input x, and i-th solution S_i, outputs proof π_i.

$\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i) \to \{0, 1\}$: On input pp, public key pk, i, input x, i-th solution S_i, and proof π_i, outputs result $\{0, 1\}$.

We define honest tuples and valid tuples as follows.

Definition 4 (Honest tuple)**.** A tuple $(pp, sk, i, x, S_i, \pi_i)$ is (λ, ψ, T)-honest if and only if for all λ, ψ, T and $pp \leftarrow \mathsf{Setup}(\lambda, \psi, T)$, the following holds:

- $i = 0$ and $(S_0, \pi_0) \leftarrow \mathsf{Init}(pp, sk, x)$, and
- $i \in \mathbb{N}^+$, $(S_i, b_i) \leftarrow \mathsf{Solve}(pp, sk, S_{i-1})$ and $\pi_i \leftarrow \mathsf{Prove}(pp, sk, i, x, S_i)$, where $(pp, sk, i-1, x, S_{i-1}, \pi_{i-1})$ is (λ, ψ, T)-honest.

Definition 5 (Valid tuple)**.** For all λ, ψ, T, and $pp \leftarrow \mathsf{Setup}(\lambda, \psi, T)$, a tuple $(pp, sk, i, x, S_i, \pi_i)$ is (λ, ψ, T)-valid if

- $(pp, sk, i, x, S_i, \pi_i)$ is (λ, ψ, T)-honest, and
- $\mathsf{Solve}(pp, sk, S_{i-1}) = (\cdot, 1)$

SeqPoW should satisfy *completeness*, *soundness*, *hardness* and *sequentiality*, plus an optional property *uniqueness*. We start from *completeness* and *soundness*.

Definition 6 (Completeness)**.** A SeqPoW scheme satisfies completeness if for all λ, ψ, T,

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp, pk, i, & (sk, pk) \leftarrow \mathsf{Gen}(pp) \\ x, S_i, \pi_i) = 1 & (pp, pk, i, x, S_i, \pi_i) \\ & \text{is } (\lambda, \psi, T)\text{-valid} \end{array}\right] = 1$$

Definition 7 (Soundness)**.** A SeqPoW scheme satisfies soundness if for all λ, ψ, T,

$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp, pk, i, & (sk, pk) \leftarrow \mathsf{Gen}(pp) \\ x, S_i, \pi_i) = 1 & (pp, pk, i, x, S_i, \pi_i) \\ & \text{is not } (\lambda, \psi, T)\text{-valid} \end{array}\right] \leq \mathsf{negl}(\lambda)$$

Then, we define *hardness* that, each attempt of $\mathsf{Solve}(\cdot)$ has the success rate of $\frac{1}{T}$.

Definition 8 (Hardness)**.** A SeqPoW scheme satisfies hardness if for all (λ, ψ, T)-honest tuple $(pp, sk, i, x, S_i, \pi_i)$,

$$\Pr\left[b_{i+1} = 1 \;\middle|\; \begin{array}{c}(S_{i+1}, b_{i+1}) \leftarrow \\ \mathsf{Solve}(pp, sk, S_i, \pi_i)\end{array}\right] \leq \frac{1}{T} + \mathsf{negl}(\lambda)$$

Similar to VDF, we define *sequentiality* that, the fastest way of computing S_i is honestly incrementing $\mathsf{Solve}(\cdot)$ for i times, which takes time $\sigma(\psi i)$. *Sequentiality* in SeqPoW also captures the *unpredictability* that, the adversary $\mathcal{A}_1$ cannot predict S_i's value before finishing computing S_i.

Definition 9 (σ-Sequentiality)**.** A SeqPoW scheme satisfies σ-sequentiality if for all λ, ψ, T, i, x, $\mathcal{A}_0$ which runs in less than time $O(\mathsf{poly}(\lambda, \psi, i))$ and $\mathcal{A}_1$ which runs in less than time $\sigma(\psi i)$,

$$\Pr\left[\begin{array}{c}(pp, sk, i, x, S_i, \pi_i) \\ \text{is } (\lambda, \psi, T)\text{-honest}\end{array} \;\middle|\; \begin{array}{c}pp \leftarrow \mathsf{Setup}(\lambda, \psi, T) \\ (sk, pk) \leftarrow \mathsf{Gen}(pp) \\ \mathcal{A}_1 \leftarrow \mathcal{A}_0(pp, sk) \\ S_i \leftarrow \mathcal{A}_1(i, x) \\ \pi_i \leftarrow \mathsf{Prove}(pp, sk, i, x, S_i)\end{array}\right] \leq \mathsf{negl}(\lambda)$$

In addition, we define an optional property *uniqueness* that, each SeqPoW puzzle only has a single valid solution S_i. Before finding S_i each $\mathsf{Solve}(\cdot)$ attempt follows the *hardness* definition, but after finding S_i no $\mathsf{Solve}(\cdot)$ attempt leads to a valid solution. *Uniqueness* implies that, once finding a valid solution S_i, the prover stops incrementing $\mathsf{Solve}(\cdot)$.

Definition 10 (Uniqueness)**.** A SeqPoW scheme satisfies uniqueness if for any two (λ, ψ, T)-valid tuples $(pp, sk, i, x, S_i, \pi_i)$ and $(pp, sk, i, x, S_j, \pi_j)$, $i = j$ holds.

3. Constructions

We formally propose two SeqPoW constructions and discuss other possible constructions. Let $H : \{0,1\}^* \to \{0,1\}^\kappa$ be a cryptographic hash function. Let G be a cyclic group. Let $H_G(\cdot)$ be a hash function that takes an arbitrarily long string $\{0,1\}^*$ to a point on G. Let g be a generator of G.

SeqPoW from VDFs. We first present a SeqPoW construction $\mathsf{SeqPoW}_{\mathsf{VDF}}$ based on self-composable VDFs. Let ψ be a step parameter, x be the input, and T be the difficulty parameter. Let the initial solution $S_0 \leftarrow H_G(pk \| x)$. We take each of $S_i = f(i\psi, S_0) = \mathsf{VDF.Eval}^i(pp, S_0, \psi)$ as an intermediate output. The prover keeps calculating each S_i, and checks if S_i satisfies the difficulty T by calculating $H(pk \| S_i) \overset{?}{\leq} \frac{2^\kappa}{T}$. If true, then S_i is a valid solution.

Once finding the valid solution S_i, the prover runs running $\mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, S_0, S_i, i\psi)$ to compute proof π_i that $S_i = \mathsf{VDF.Eval}^i(pp, S_0, \psi)$. Note that $\mathsf{VDF.Eval}^i(pp, S_0, \psi) = \mathsf{VDF.Eval}(pp, S_0, i\psi)$ for *self-composable* VDFs. With pk, i, x, S_i, and π_i, the verifier can run $S_0 \leftarrow H_G(pk \| x)$ and $\mathsf{VDF.Verify}(pp_{\mathsf{VDF}}, S_0, S_i, \pi_i, i\psi)$ to check whether 1) S_i satisfies the difficulty T, and 2) $S_i = \mathsf{Eval}^i(pp, S_0, \psi)$. Figure 3 describes our $\mathsf{SeqPoW}_{\mathsf{VDF}}$ construction in detail.

Unique SeqPoW from Sloth. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ does not provide *uniqueness*: the prover can keep incrementing the iteratively sequential function to find as many valid solutions as possible. We present $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ – a SeqPoW construction with *uniqueness*. $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ is based on Sloth [19], a *slow-timed* hash function. In Sloth, the prover square roots (on a cyclic group G) the input for t times to get the output. Given an output, the verifier squares the output for t times to recover the input and checks if the input is same as the one from the prover. Verification in Sloth can be fast: on cyclic group G, squaring is $O(\log |G|)$ times faster than square rooting. In addition, Sloth is *reversible* [30]: given an output and t, the verifier can recover all intermediate results and the input.

Same as in $\mathsf{SeqPoW}_{\mathsf{VDF}}$, $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ takes each of $S_i = f(i\psi, S_0)$ as an intermediate output and checks if $H(pk \| S_i) \leq \frac{2^\kappa}{T}$. In addition, $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ only treats the first solution satisfying the difficulty as valid, which makes the solution unique. Figure 4 describes the detailed construction of $\mathsf{SeqPoW}_{\mathsf{Sloth}}$.

Other possible constructions. The main ingredient of SeqPoW is the succinct proof of execution results of iteratively sequential functions. In addition to VDFs and Sloth, Incremental Verifiable Computation (IVC) [31] can also provide such proofs. IVC is a powerful primitive that, a prover can produce a succinct proof on the correctness of any historical execution, and for any further step of computation, the prover can obtain the proof by only updating the last step's proof rather than computing it from scratch.

The advantage of IVC-based SeqPoW is that it supports any iteratively sequential functions. This means IVC-based SeqPoW can be more egalitarian by using iteratively sequential functions that are more hard to parallelise and optimise. However, IVC is usually constructed from complicated cryptographic primitives, such as SNARKs [31]–[35]. This makes the construction inefficient and implementation challenging. In addition, when generating proofs in IVC takes non-negligible time, IVC-based SeqPoW may not be fair, as miners with powerful hardware can take advantage of mining by accelerating $\mathsf{SeqPoW.Prove}(\cdot)$.

4. Security and efficiency analysis

Security. We defer full security proofs to Appendix A. Proofs of *completeness* and *soundness* directly follow the completeness, soundness and self-composability of Sloth and VDFs. By *pseudorandomness* of $H_G(\cdot)$ and *sequentiality* of Sloth and VDFs, $\mathsf{Solve}(\cdot)$ outputs unpredictable solutions. Then, by $H(\cdot)$'s *pseudorandomness* and $\mathsf{Solve}(\cdot)$'s unpredictability, the probability that the solution satisfies the difficulty is $\frac{1}{T}$, leading to *hardness*. *Sequentiality* directly follows the sequentiality and self-composability of Sloth and VDFs.

The underlying VDFs in $\mathsf{SeqPoW}_{\mathsf{VDF}}$ may use cyclic groups that require trusted setup. For example, Wesolowski's VDF (Wes19) [22] and Pietrzak's VDF (Pie19) [21] – which are two commonly used VDFs – use RSA groups. In this case,

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Setup}(\lambda, \psi, T)$

1 : $pp_{\mathsf{VDF}} = (\{0,1\}^*, G, g)$
$\leftarrow \mathsf{VDF}.\mathsf{Setup}(\lambda)$
2 : $pp \leftarrow (pp_{\mathsf{VDF}}, \psi, T)$
3 : **return** pp

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Gen}(pp)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : Sample random $sk \in \mathbb{N}$
3 : $pk \leftarrow g^{sk} \in G$
4 : **return** (sk, pk)

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Init}(pp, sk, x)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : $pk \leftarrow g^{sk} \in G$
3 : $S_0 \leftarrow H_G(pk \| x)$
4 : **return** S_0

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Solve}(pp, sk, S_i)$

1 : $(pp_{\mathsf{VDF}}, \psi, T) \leftarrow pp$
2 : $(\{0,1\}^*, G, g) \leftarrow pp_{\mathsf{VDF}}$
3 : $pk \leftarrow g^{sk} \in G$
4 : $S_{i+1} \leftarrow \mathsf{VDF}.\mathsf{Eval}(pp_{\mathsf{VDF}}, S_i, \psi)$
5 : $b_{i+1} \leftarrow H(pk \| S_{i+1}) \leq \frac{2^\kappa}{T}\ ?\ 1 : 0$
6 : **return** (S_{i+1}, b_{i+1})

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Prove}(pp, sk, i, x, S_i)$

1 : $(pp_{\mathsf{VDF}}, \psi, T) \leftarrow pp$
2 : $(\{0,1\}^*, G, g) \leftarrow pp_{\mathsf{VDF}}$
3 : $pk \leftarrow g^{sk} \in G$
4 : $S_0 \leftarrow H_G(pk \| x)$
5 : $\pi_{\mathsf{VDF}} \leftarrow \mathsf{VDF}.\mathsf{Prove}(pp_{\mathsf{VDF}}, S_0, S_i, i\psi)$
6 : **return** π_{VDF}

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i)$

1 : $(pp_{\mathsf{VDF}}, \psi, T) \leftarrow pp$
2 : $(\{0,1\}^*, G, g) \leftarrow pp_{\mathsf{VDF}}$
3 : $S_0 \leftarrow H_G(pk \| x)$
4 : **if** $\mathsf{VDF}.\mathsf{Verify}(pp_{\mathsf{VDF}}, S_0, S_i, \pi_i, i\psi) = 0$
5 : **return** 0
6 : **if** $H(pk \| S_i) > \frac{2^\kappa}{T}$ **then return** 0
7 : **return** 1

Fig. 3: Construction of $\mathsf{SeqPoW}_{\mathsf{VDF}}$.

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Setup}(\lambda, \psi, T)$

1 : $pp \leftarrow (\{0,1\}^*, G, g, \psi, T)$
2 : **return** pp

$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Gen}(pp)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : Sample random $sk \in \mathbb{N}$
3 : $pk \leftarrow g^{sk} \in G$
4 : **return** (sk, pk)

$\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Init}(pp, sk, x)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : $pk \leftarrow g^{sk} \in G$
3 : $S_0 \leftarrow H_G(pk \| x)$
4 : **return** S_0

$\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Solve}(pp, sk, S_i)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : $pk \leftarrow g^{sk} \in G$
3 : $S_{i+1} \leftarrow S_i^{\frac{1}{2^\psi}}$
4 : $b_{i+1} \leftarrow H(pk \| S_{i+1}) \leq \frac{2^\kappa}{T}\ ?\ 1 : 0$
5 : **return** (S_{i+1}, b_{i+1})

$\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Prove}(pp, sk, i, x, S_i)$

1 : **return** $\perp$

$\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i)$

1 : $(\{0,1\}^*, G, g, \psi, T) \leftarrow pp$
2 : $y \leftarrow S_i$
3 : **if** $H(pk \| y) > \frac{2^\kappa}{T}$, **then return** 0
4 : **repeat** i **times**
5 : $y \leftarrow y^{2^\psi}$
6 : **if** $H(pk \| y) \leq \frac{2^\kappa}{T}$ **then return** 0
7 : $g \leftarrow H_G(pk \| x)$
8 : **if** $g \neq y$ **then return** 0
9 : **return** 1

Fig. 4: Construction of $\mathsf{SeqPoW}_{\mathsf{Sloth}}$.

TABLE II: Efficiency of two SeqPoW constructions. ISF stands for iteratively sequential function.

	Construction		Efficiency			
	ISF	Proof	$\mathsf{Solve}(\cdot)$	$\mathsf{Prove}(\cdot)$	$\mathsf{Verify}(\cdot)$	Proof size (Bytes)
$\mathsf{SeqPoW}_{\mathsf{VDF}}$	Repeated SQ.	Wes19 [22]	$O(\psi)$	$O(\psi T)$	$O(\log \psi T)$	s
		Pie19 [21]	$O(\psi)$	$O(\sqrt{\psi T} \log \psi T)$	$O(\log \psi T)$	$s \log_2 \psi T$
$\mathsf{SeqPoW}_{\mathsf{Sloth}}$	Repeated SQRT.	Repeated SQ.	$O(\psi)$	0	$O(\psi T)$	0

we assume the underlying VDFs are securely bootstrapped. In practice, one can either employ a trusted third party, or use specialised multi-party protocols [36], [37] for trusted setup.

Efficiency. Table II summarises the efficiency analysis of two SeqPoW constructions. For iteratively sequential functions, $\mathsf{SeqPoW}_{\mathsf{VDF}}$ and $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ employ repeated squaring on an RSA group and repeated square rooting on a prime-order group, respectively. Let s be the size (in Bytes) of an element on the group, and ψ be the step parameter. Each $\mathsf{Solve}(\cdot)$ executes ψ steps of the iteratively sequential function. By *hardness* and *uniqueness*, a node attempts $\mathsf{Solve}(\cdot)$ for T times to find a valid solution on average. Thus, $\mathsf{Prove}(\cdot)$ generates proofs on ψT steps, and $\mathsf{Verify}(\cdot)$ verifies proofs of ψT steps.

We analyse the efficiency of $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with two prevalent VDF constructions, namely Wesolowski's VDF (Wes19) [22] and Pietrzak's VDF (Pie19) [21]. According to the existing analysis [38], the proving complexity, verification complexity and proof size of Wes19 are $O(\psi T)$, $O(\log \psi T)$ and s Bytes, respectively; and the proving complexity, verification complexity and proof size of Pie19 are $O(\sqrt{\psi T} \log \psi T)$, $O(\log \psi T)$ and $s \log_2 \psi T$, respectively. There have been techniques for optimising and parallelising $\mathsf{VDF}.\mathsf{Prove}(\cdot)$ for both

VDFs [21], [22], [39]. The proof sizes of both $\mathsf{SeqPoW}_{\mathsf{VDF}}$ constructions are acceptable: when $\psi T = 2^{40}$ and $s = 32$ Bytes, the proof sizes of $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Wes19 [22] and with Pie19 [21] are 32 and 1280 Bytes, respectively.

$\mathsf{SeqPoW}_{\mathsf{Sloth}}$ does not have proofs: the verifier keeps squaring the solution to recover the input and checks whether the recovered input equals to the real one. This leads to verification complexity of $O(\psi T)$.

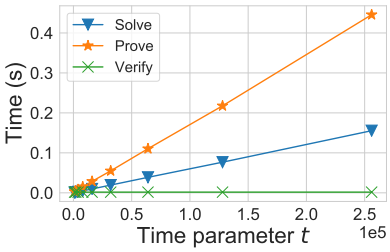
5. Performance evaluation

We evaluate the performance of our two SeqPoW constructions. We implement both $\mathsf{SeqPoW}_{\mathsf{VDF}}$ and SeqPoW.Sloth without optimisation, and the code is available at Github [40]. For $\mathsf{SeqPoW}_{\mathsf{VDF}}$, Pie19 [21] is a better candidate compared to Wes19 [22], given Pie19's efficient proof generation and verification and acceptable proof size. For SeqPoW.Sloth, verifying puzzles is approximately five times faster than solving puzzles. Although far from the theoretically optimal value (i.e., 1024 in our case), the verification efficiency is acceptable.

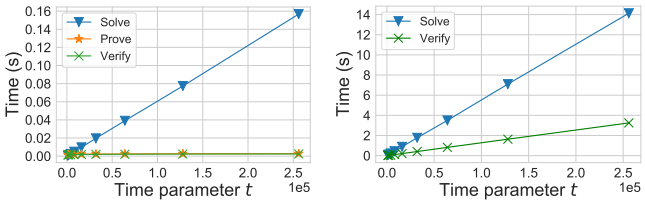
Implementation. We implement the two SeqPoW constructions in Rust programming language. We use the `rug` [41] crate for big integer arithmetic. With big integer arithmetic, we implement the RSA group with 1024-bit keys and the group of prime order. We implement the two $\mathsf{SeqPoW}_{\mathsf{VDF}}$ constructions based on the RSA group, and SeqPoW.Sloth based on the group of prime order. Our implementations strictly follow their original papers [19], [21], [22] without any optimisation.

Experimental setup. We benchmark $\mathsf{Solve}(\cdot)$, $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ for each SeqPoW construction. We test $\psi \in [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000]$ and assume $i = 1$. Note that $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ does not have $\mathsf{Prove}(\cdot)$. We benchmark the performance using Rust's native benchmarking suite `cargo-bench` [42] and `criterion` [43]. We sample ten executions for each configuration, i.e., a function with a unique group of parameters. We specify O3-level optimisation when compiling. All experiments were conducted on a MacBook Pro with a 2.2 GHz 6-Core Intel Core i7 Processor and a 16 GB 2400 MHz DDR4 RAM.

Results (Figure 5). For all SeqPoW constructions, the running time of $\mathsf{Solve}(\cdot)$ increases linearly with time parameter t. This is as expected as $\mathsf{Solve}(\cdot)$ is dominated by the iteratively sequential function. For $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Wes19, $\mathsf{Prove}(\cdot)$ takes more time than $\mathsf{Solve}(\cdot)$, as our implementation follows the original algorithm rather than considering other optimisations (such as [39]). $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Pie19 achieves ideal performance: $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ takes negligible time compared to $\mathsf{Solve}(\cdot)$. In addition, according to the efficiency analysis, the proof size of Pie19 is also acceptable. Thus, without optimisation, $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Pie19 is more practical than with Wes19. For $\mathsf{SeqPoW}_{\mathsf{Sloth}}$, $\mathsf{Solve}(\cdot)$ is approximately five times slower than $\mathsf{Verify}(\cdot)$. Although this is far from theoretically optimal value, i.e, $\log_2 |G| = 1024$ in our case [30], the verification overhead is acceptable when random outputs are not generated frequently.



(a) $\mathsf{SeqPoW}_{\mathsf{VDF}}$ + Wes19 [22].



(b) $\mathsf{SeqPoW}_{\mathsf{VDF}}$ + Pie19 [21]. (c) $\mathsf{SeqPoW}_{\mathsf{Sloth}}$.

Fig. 5: Evaluation of SeqPoW constructions.

4. RandChain: DRB from SeqPoW

Based on Sequential Proof-of-Work (SeqPoW), we construct RandChain, a new family of Decentralised Randomness Beacons (DRBs). Figure 6 describes the full construction of RandChain.

System setting. Let $H, H' : \{0,1\}^* \to \{0,1\}^\kappa$ be two different hash functions. RandChain works in a permissioned system. Nodes $\mathcal{P} = \{p_1, \ldots, p_n\}$ register themselves into the system through an identity management service such as a Public Key Infrastructure. The identity management service assigns each registered node $p_k \in \mathcal{P}$ with a pair of secret key sk_k and public key pk_k. Each node is identified by its public key in the system. Given a public key, any node in the system can verify whether the node is in the system by querying the identity management service. There can be an unlimited number of registered nodes. Each node does not know the exact number of nodes in the system, and is only directly connected to a subset of peers.

Structure (Figure 7a). All nodes jointly maintain a *blockchain*. Each block B is of the format (h^-, h, i, S, pk, π), where h^- is the previous block ID, h is the current block ID, i is the SeqPoW solution index, S is the SeqPoW solution, pk is the public key of this block's creator, and π is the proof that S is a valid SeqPoW solution on input h^-. The block ID $B.h$ is calculated as $B.h = H(pk \| S)$. The block's random output $B.rand$ is calculated as $B.rand = H'(pk \| S)$.

Each node p_k has its local view $\mathcal{C}_k$ of the blockchain. The local view $\mathcal{C}_k$ may have *forks*: there may be multiple blocks following the same block. RandChain applies Nakamoto consensus that, nodes considers the longest chain among all forks as valid. As described in $\mathsf{mainChain}(\cdot)$ of Figure 6, node p_k considers the longest fork of $\mathcal{C}_k$ as the main chain $\mathcal{MC}_k$.

Process (Figure 7b). We describe the process of RandChain from a node p_k's perspective. Node p_k runs two

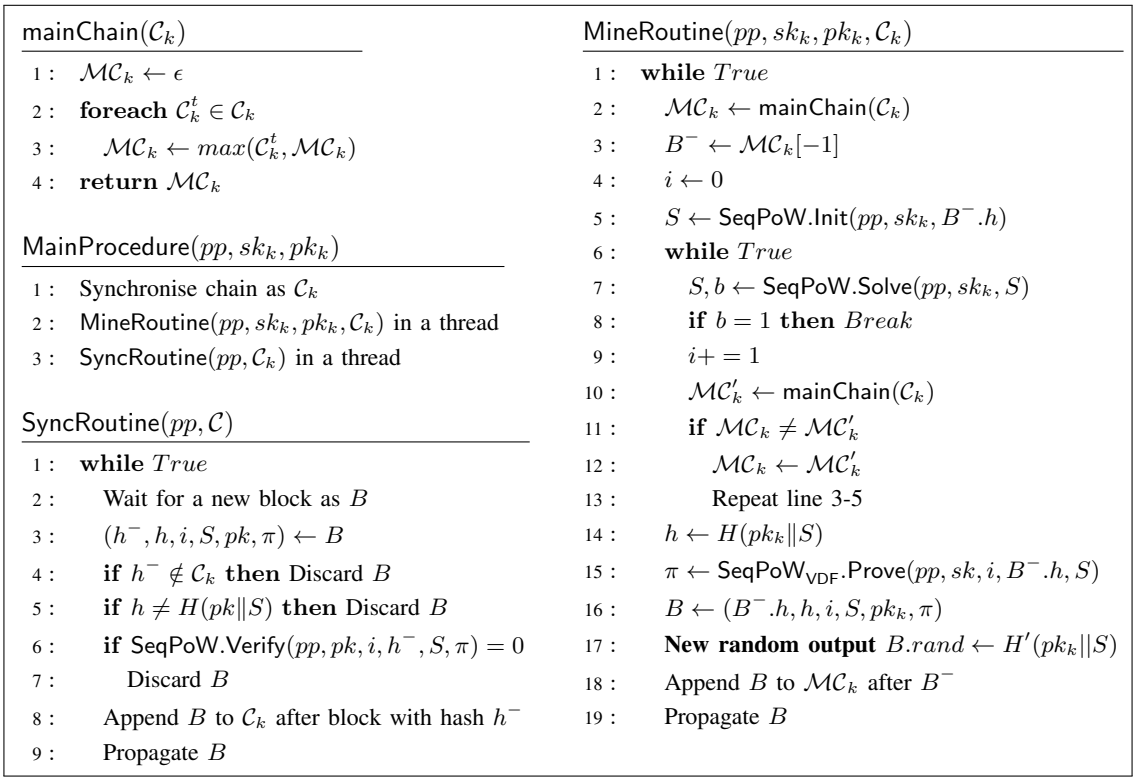


Fig. 6: Construction of RANDCHAIN.

routines: the synchronisation routine SyncRoutine$(\cdot)$ and the mining routine MineRoutine$(\cdot)$. In SyncRoutine$(\cdot)$, node p_k synchronises its local blockchain $\mathcal{C}_k$ with other nodes. The synchronisation process is the same as in other blockchains: node p_k keeps receiving blocks from other nodes, verifying them, and adding valid blocks to its local blockchain $\mathcal{C}_k$.

In MineRoutine$(\cdot)$, node p_k keeps *mining*, i.e., adding new blocks, on the main chain $\mathcal{MC}_k$. To append a block to the blockchain, node p_k should solve a SeqPoW puzzle. In particular, node p_k finds its latest main chain $\mathcal{MC}_k$, and derives a SeqPoW puzzle from $\mathcal{MC}_k$'s last block. Then, node p_k executes SeqPoW.Init$(\cdot)$ to find the starting point of mining. Node p_k keeps solving SeqPoW by iterating SeqPoW.Solve$(\cdot)$ until finding a solution S satisfying the difficulty. As SeqPoW.Init$(\cdot)$ takes each node's secret key as input, nodes' SeqPoW puzzles start from different points and take different steps to solve. With a valid solution S, node p_k constructs a block B consisting of a random output, and appends B to $\mathcal{MC}_k$.

As RANDCHAIN employs Nakamoto consensus, RANDCHAIN's blockchain may have forks, and it's possible that B is committed then reverted by a longer chain. If a block has a sufficient number of succeeding blocks, then B is considered *stable*, i.e., cannot be reverted, except for negligible probability [44]. The fork rate can be reduced by applying a high difficulty parameter T. With a high T, the average time of mining a new block can be long, giveing nodes sufficient time to propagate new blocks. We analyse RANDCHAIN's consistency guarantee in §5, and discuss how to achieve *finality*, i.e., make blocks irreversible in §6-B.

5. SECURITY AND EFFICIENCY OF RANDCHAIN

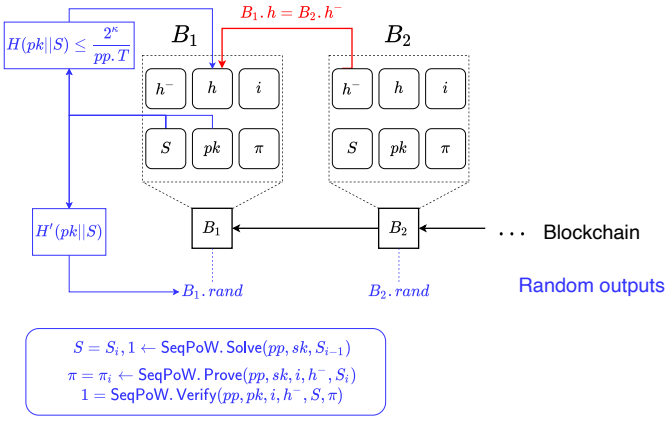
In this section, we analyse the security and efficiency of RANDCHAIN. We define the notion of Decentralised Randomness Beacon (DRB), and prove that RANDCHAIN implements a DRB and produces strongly unpredictable randomness. In addition, we show that while inheriting simplicity and scalability from PoW-based consensus, RANDCHAIN remains energy-efficient and decentralised.

1. Security model

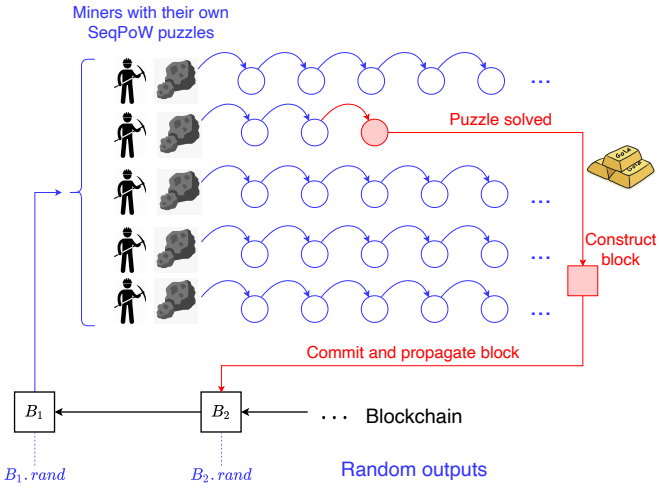
We consider the network is *synchronous*: all messages are delivered within a known time bound Δ. We do not assume rounds or *lock-step synchrony* [44]. The network consists of n nodes, each of which controls the same amount of mining power. We consider an adaptive Byzantine adversary who can corrupt any of αn nodes at any time, where $\alpha \in [0, 1]$. The adversary can coordinate its corrupted nodes in real-time without delay, and can arbitrarily delay, drop, forge and modify messages sent from its corrupted nodes. Let $\beta = 1 - \alpha$ be the percentage of correct nodes.

2. Defining Decentralised Randomness Beacon

We start from defining security properties of DRB. First, similar to Nakamoto consensus, DRB should satisfy *consistency* and *liveness*. We follow existing papers [5], [45]–[47] for defining *consistency* and *liveness*. *Consistency* requires



(a) Beacon structure.



(b) Process of mining.

Fig. 7: Illustration of RANDCHAIN beacon.

nodes to have a consistent view on the blockchain. Without *consistency*, nodes may revert random outputs arbitrarily.

Definition 11 (Consistency)**.** Parametrised by $\ell \in \mathbb{N}$. A DRB satisfies ℓ-consistency if for any two correct nodes at any time, their chains can differ only in the last ℓ blocks, except for negligible probability.

The parameter ℓ defines the degree of consistency guarantee. Some based applications require RB to have *finality*, i.e., at any time, correct nodes do not have conflicted views on the blockchain. The *finality* here is equivalent to 0-consistency. In §6 we discuss how to add *finality* to RANDCHAIN.

Liveness requires DRB to produce no less than $\lfloor \tau \cdot t \rfloor$ random outputs for every time period of t. Without *liveness*, DRB may stop producing randomness forever. Existing papers usually define *termination* [7], [10], [48] or *Guaranteed Output Delivery (G.O.D.)* [8], [49]–[51] that, for every round, there will always be a new random output. We do not follow their definitions, as RANDCHAIN neither iterates single-shot DRG protocols nor uses the concept of rounds. In §7 we will compare RANDCHAIN with existing DRBs in detail.

Definition 12 (Liveness)**.** Parametrised by $t, \tau \in \mathbb{R}^+$. A DRB satisfies (t, τ)-liveness if for any time period t, every correct node receives at least $\lfloor \tau \cdot t \rfloor$ new outputs.

Then, as a DRB, each output should be pseudorandom, i.e., uniformly distributed.

Definition 13 (Uniform distribution)**.** A DRB satisfies uniform distribution if every output is indistinguishable with a uniformly random string with the same length, except for negligible probability.

Last, each output should be *unpredictable*: given the current blockchain, no one can learn any knowledge of the next output. If one can predict the next output, it may take advantage in protocols based on the DRB. From a node's perspective, this includes two scenarios: 1) the next output is generated by itself, and 2) the next output is generated by other nodes. Some papers [9], [50]–[52] refers *unpredictability* in the first scenario as *bias-resistance*. We follow the *IND1-secrecy* (Indistinguishability of secrets) notion in SCRAPE [8] to define *unpredictability*. IND1-secrecy requires that each node cannot learn anything about the final output before finishing the protocol.

Definition 14 (Unpredictability)**.** A DRB satisfies unpredictability if no adversary can obtain non-negligible advantage on the following game. Assuming all messages are delivered instantly and nodes agree on a blockchain of length ℓ. Before the $(\ell+1)$-th block is mined (either by other nodes or by the adversary), the adversary makes a guess on the random output in the $(\ell+1)$-th block. Let r be the guessed random output, and r' be the real random output. The adversary's advantage is quantified as $\Pr[r = r']$.

3. Security analysis

Consistency, liveness and uniform distribution. RANDCHAIN's *consistency* and *liveness* are guaranteed by Nakamoto consensus. There have been extensive works [5], [44]–[46], [53]–[56] analysing and proving consistency and liveness guarantee of Nakamoto consensus. As RANDCHAIN works in the same system model as that of Ren [5], RANDCHAIN at least satisfies the *consistency* and *liveness* bound proved by Ren [5]. *Uniform distribution* is guaranteed by hash functions. For every block B, $B.rand$ is produced by the hash function $H'(\cdot)$. By pseudorandomness of hash functions, $B.rand$ indistinguishable with a uniformly random κ-bit string.

Unpredictability. Assuming all messages are delivered instantly, and nodes agree on a blockchain of length ℓ. In the prediction game, the $(\ell+1)$-th block is either produced by correct nodes or the adversary's nodes. If the adversary's advantage is negligible for both cases, then RANDCHAIN satisfies *unpredictability*. We prove the adversary's advantage for both cases. For the first case that the $(\ell+1)$-th block is produced by correct nodes, the adversary's best strategy is guessing, leading to negligible advantage.

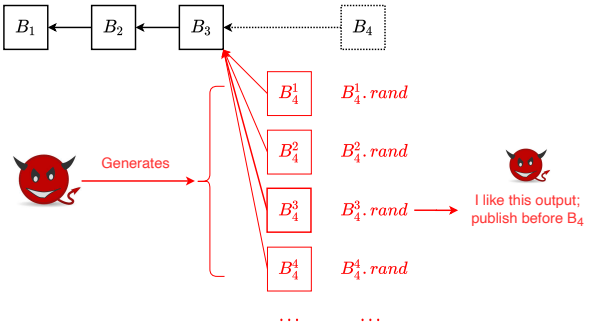


Fig. 8: Unpredictability game.

Lemma 2. *Assuming all messages are delivered instantly and nodes agree on a blockchain of length ℓ. If the $(\ell+1)$-th block is produced by a correct node, then the adversary's advantage on the prediction game is $\frac{1}{2^\kappa}$.*

We then consider the next output is produced by the adversary's nodes. By *sequentiality*, the adversary cannot predict its SeqPoW solutions. As shown in Figure 8, the adversary's best strategy is to produce as many blocks before the next honest block as possible. With more blocks, the adversary has more random outputs to choose, leading to higher advantage. We first analyse RANDCHAIN using SeqPoW without *uniqueness*, e.g., $\mathsf{SeqPoW}_{\mathsf{VDF}}$.

Lemma 3. *Consider* RANDCHAIN *using SeqPoW without uniqueness. Assuming all messages are delivered instantly and nodes agree on a blockchain of length ℓ. If the $(\ell+1)$-th block is produced by the adversary, then the adversary's advantage on the prediction game is $\frac{k}{2^\kappa}$ with $\alpha^k\beta$ probability, where $k \leq \alpha n$.*

Proof. The adversary controls αn nodes, and $k \leq \alpha n$. Let V_k be the event that "the adversary mines k blocks before correct nodes mine the first block". By *hardness*, each node can find unlimited valid SeqPoW solutions given a fixed input. Then, we have

$$Pr[V_k] = \alpha^k\beta$$

When V_k happens, the adversary's advantage is $\frac{k}{2^\kappa}$.

Therefore, with the probability $\alpha^k\beta$, the adversary mines k blocks before correct nodes mine a block, leading to the advantage of $\frac{k}{2^\kappa}$ (where $k \leq \alpha n$). □

We then analyse RANDCHAIN with *unique* SeqPoW, e.g., $\mathsf{SeqPoW}_{\mathsf{Sloth}}$. In RANDCHAIN with *unique* SeqPoW, given the latest blockchain with height ℓ, each node can only mine a single block at height $\ell+1$. Thus, the chance that the adversary mines blocks at height $\ell+1$ decreases while mining more blocks at height $\ell+1$. In addition, the adversary can mine at most αn blocks at height $\ell+1$.

Lemma 4. *Consider* RANDCHAIN *using SeqPoW with uniqueness. Assuming all messages are delivered instantly and nodes agree on a blockchain of length ℓ. If the $(\ell+1)$-th block is produced by the adversary, then the adversary's advantage on the prediction game is $\prod_{i=0}^{k-1}\frac{\alpha n-i}{n-i}\cdot\beta$ with $\alpha^k\beta$ probability.*

Proof. The adversary controls αn nodes, and $k \leq \alpha n$. Let V'_k be the event that "the adversary mines k blocks before correct nodes mine the first block". By *uniqueness*, each node can find only one valid SeqPoW solutions given a fixed input. Let α_k be the mining power of the adversary's k-th node, and $\sum\alpha_k = \alpha$. Then, we have

$$Pr[V'_0] = \beta \tag{1}$$
$$Pr[V'_1] = \alpha\beta \tag{2}$$
$$Pr[V'_2] = \frac{\alpha n-1}{n-1}\alpha\beta \tag{3}$$
$$Pr[V'_3] = \frac{\alpha n-2}{n-2}\cdot\frac{\alpha n-1}{n-1}\alpha\beta \tag{4}$$
$$\dots \tag{5}$$
$$Pr[V'_k] = \prod_{i=0}^{k-1}\frac{\alpha n-i}{n-i}\cdot\beta \tag{6}$$

When V'_k happens, the adversary's advantage is $\frac{k}{2^\kappa}$.

Therefore, with less than the probability $\prod_{i=0}^{k-1}\frac{\alpha n-i}{n-i}\cdot\beta$, the adversary mines k blocks before correct nodes mine a block, leading to the advantage of $\frac{k}{2^\kappa}$ (where $k \leq \alpha n$). □

Remark 1. Note that the probability that the adversary achieves a certain advantage in RANDCHAIN with *unique* SeqPoW is always smaller than in RANDCHAIN with *non-unique* SeqPoW. In particular, for every k, $Pr[V'_k] \leq Pr[V_k]$. Given k, we have

$$Pr[V'_k] = \prod_{i=0}^{k-1}\frac{\alpha n-i}{n-i}\cdot\beta \tag{7}$$
$$= \prod_{i=0}^{k-1}\left[\alpha\cdot\frac{\alpha n-i}{\alpha n-\alpha i}\right]\cdot\beta \tag{8}$$

As $i \leq 0$ and $\alpha \in [0,1]$, $\frac{\alpha n-i}{\alpha n-\alpha i} \leq 1$. Thus,

$$Pr[V'_k] \leq \prod_{i=0}^{k-1}\alpha\cdot\beta \tag{9}$$
$$= \alpha^k\beta = Pr[V_k] \tag{10}$$

4. *Efficiency*

Mining in RANDCHAIN is non-parallelisable: 1) SeqPoW is sequential, and 2) given the last block, each node's input of SeqPoW is fixed. Thanks to non-parallelisable mining, RANDCHAIN is more energy-efficient and decentralised than PoW-based consensus.

Parallelisable PoW mining has led to mining power centralisation and huge energy consumption. In order to maximise mining reward, miners have been employing specialised mining hardware such as GPUs, FPGAs and ASICs for PoW mining. Currenlty, PoW mining has been marketised: people manufacture, invest and trade high-end mining hardware for

profit. Mining power marketisation leads to mining power centralisation: most blocks are mined only by several poweful miners. Mining power centralisation weakens PoW-based consensus' security, as powerful miners can perform various attacks, e.g., 51% attacks to break PoW-based consensus. In addition, mining enforces hardware to fully operate all the time. As miners deploy powerful mining hardware, mining costs a great amount of electricity. For example, in 2019, Bitcoin cost 58.93 KWh electricity – more than Switzerland which cost 58.46 KWh [57].

Compared to PoW-based consensus, RANDCHAIN remains energy-efficient and decentralised. As RANDCHAIN does not allow nodes to choose their own entropy and SeqPoW is sequential, SeqPoW mining is *non-parallelisable*: each miner can only use a single processor for mining. Compared to powerful specialised hardware in PoW mining, a single fully operating processor costs negligible electricity. In addition, high-end mining hardware can only achieve limited speedup on running $\mathsf{SeqPoW.Solve}(\cdot)$. $\mathsf{SeqPoW.Solve}(\cdot)$ can only be accelerated by using processors with higher clock rate. The highest clock rate achieved by processors is 8.723 GHz, while laptops' clock rate is usually more than 2 GHz [58]. Given the voltage limit of processors, the current clock rate is hard to improve further [59]. This means one can speed up $\mathsf{SeqPoW.Solve}(\cdot)$ for less than five times. The limited speedup is also evidenced by the recent *VDF Alliance FPGA Contest* [60]–[62], where optimised VDF implementations are approximately four times faster than the baseline implementation. Compare to PoW mining where powerful nodes can mine thousands of times faster than normal nodes, RANDCHAIN achieves a high degree of mining power decentralisation.

6. DISCUSSION

1. Other security issues in Nakamoto consensus

RANDCHAIN employs Nakamoto consensus, so inherits most security issues from Nakamoto consensus. Nonetheless, RANDCHAIN is a DRB rather than a transaction ledger, and security issues for transaction ledgers may not be critical for DRB. For example, RANDCHAIN is immune to all attacks on Nakamoto consensus' incentive mechanisms [63]–[66], as it works in permissioned networks. In addition, RANDCHAIN can resist some attacks in Nakamoto consensus. SeqPoW can be used for detecting withholding behaviours. If a node broadcasts a valid block with solution index i after another valid block with solution index $j < i$, other nodes can know that the node withholds the block with i.

2. Adding finality

Finality [67] is a property for Byzantine consensus that, previously agreed proposals cannot be reverted. RANDCHAIN does not satisfy *finality* due to the probabilistic consensus guarantee of Nakamoto consensus. In some scenarios, we require a DRB to satisfy finality. For example, considering a leader election scheme where a group of nodes elect a leader with a random output from RANDCHAIN as input, if the used random output is reverted before the end of leader election, then nodes may stop working and lose liveness. RANDAO [12] and Proofs-of-Delay [15] bypass this by replacing $H'(\cdot)$ with a VDF, of which the execution time is longer than generating ℓ blocks, where ℓ is the degree of ℓ-*consistency*. Nodes can reveal every random output only after the block deriving this random output becomes stable. However, this enables *frontrunning* – nodes with fast processors always learn random outputs earlier than normal nodes – which may lead to serious fairness issues in time-sensitive applications such as decentralised exchanges [68].

We consider principled approaches for adding *finality*. Adding *finality* is equivalent to achieve 0-*consistency*: correct nodes decide the same block at every height. This is further equivalent to making RANDCHAIN to execute in rounds, in each of which nodes execute a *Byzantine agreement* [69] to agree on a block. RANDCHAIN satisfies *validity* and *termination*, but provides eventual consistency rather than the *agreement* property of *Byzantine agreement*, as nodes may temporarily agree on different blocks at the same height. We discuss two approaches to achieve *agreement*, namely the quorum-based approach and the herding-based consensus.

Quorum-based approach. In Byzantine agreement, quorum [70] is the minimum number of votes that a proposal has to obtain for being agreed by nodes. If a proposal obtains a quorum of votes in a view, then this means that nodes have reached an agreement on this proposal. The quorum size is $n-f$, where n and f be the number of nodes and faulty nodes in the system, respectively. Existing research [70], [71] shows that $n \geq 3f+1$ is required to achieve agreement in partially synchronous networks, and $n \geq 2f+1$ is required to achieve agreement in synchronous networks. A quorum certificate of a proposal is a quorum of votes on this proposal. A vote is usually represented as a digital signature on the proposal, view ID and other metadata.

To achieve *agreement* in RANDCHAIN, we can apply the quorum mechanism as follows. The system should additionally assume $n \geq 3f+1$. A node signs a block to vote this block. The node's view is represented as the previous block hash, which is inside the signed block. Nodes actively propagate their votes – i.e., signatures on blocks – the same way as propagating blocks. Each node keeps received votes locally, and considers a block as finalised if collecting a quorum certificate, i.e., no less than $2f+1$ signatures on this block. RANDCHAIN still keeps Nakamoto consensus as a fallback solution. If there are multiple forks without quorum certificates, nodes mine on the longest fork. A block can be considered finalised with a sufficiently long sequence of succeeding blocks, even without a quorum certificate. This resembles the Streamlet blockchain [47].

Herding-based consensus. There have been a family of consensus protocols based on *herding*. *Herding* is a social phenomenon where people make choices according to other people's preference. In herding-based consensus, nodes keep exchanging their votes with each other and decide the proposal with most votes. Existing research [72], [73] shows that,

with overwhelming probability, nodes will eventually agree on a proposal by herding in a short time period. In addition, herding-based consensus introduces much less communication overhead than traditional Byzantine consensus.

To achieve *agreement* in RANDCHAIN, we can apply the herding-based consensus as follows. Upon a new block, nodes execute a herding-based consensus on it. If a block is the only block in a long time period, then nodes will agree on this block directly. If there are multiple blocks within a short time period, then nodes will agree on the most popular block among them with overwhelming probability. This approach has also been discussed in Bitcoin Cash community, who seeks to employ Avalanche [73] as a finality layer for Bitcoin Cash [74].

3. Making SeqPoW mining non-outsourceable

RANDCHAIN does not prevent *outsourcing*: the adversary can solve others' SeqPoW puzzles. If the adversary controls massive processors with high clock rate, then it can mine for all nodes simultaneously and know SeqPoW solutions of all nodes earlier than others. Such adversary can always frontrun other nodes, or even bias random outputs by selectively publishing its preferred ones.

To prevent outsourcing, we can adopt the idea of *VRF-based mining* [75]. Verifiable Random Function (VRF) – which can be seen as a public key version of hash functions – takes the prover's secret key sk and an input x, and outputs a random string y. The prover can also generates a proof π that, 1) y is a valid output of x, and 2) y is generated by the owner of sk. We replace $H(pk\|S_{i+1}) \leq \frac{2^\kappa}{T}$ with $\mathsf{VRFHash}(sk, S_{i+1}) \leq \frac{2^\kappa}{T}$ in SeqPoW's difficulty mechanism, where $\mathsf{VRFHash}(\cdot)$ is a VRF [76]. As correct nodes do not share their secret keys to others, the adversary cannot execute $\mathsf{VRFHash}(\cdot)$ for others. While this modification adds negligible overhead to $\mathsf{SeqPoW}_{\mathsf{VDF}}$, it greatly increases the proof size of $\mathsf{SeqPoW}_{\mathsf{Sloth}}$, as the proof should carry all VRF outputs and proofs for proving no prior solution satisfies the difficulty. More efficient SeqPoW constructions with *uniqueness* are considered as future work.

4. Dynamic difficulty

PoW-based blockchains employ difficulty adjustment mechanism for stabilising the block rate, i.e., the average number of new blocks in a time unit. This is particularly useful when churn [77] is high and/or the network size is frequently changing. Although we analyse RANDCHAIN while assuming a fixed difficulty and a fixed set of nodes, RANDCHAIN can support dynamic difficulty adjustment with little change. First, similar to PoW-based blockchains, RANDCHAIN can include a timestamp to each block, so that RANDCHAIN can infer historical block rate using timestamps. In addition, RANDCHAIN includes the number i of iterations running $\mathsf{SeqPoW.Solve}(\cdot)$, and i can also infer the historical block rate. If historical values of i are large, then this means that mining is too hard and the difficulty should be reduced, and vice versa.

7. COMPARISON WITH EXISTING DRB PROTOCOLS

In this section, we compare RANDCHAIN with existing DRB protocols. There are three paradigms of constructing Decentralised Randomness Beacons (DRBs), namely 1) DRB from external randomness source, 2) DRB from Distributed Randomness Generation (DRG), and 3) DRB from iteratively sequential functions. RANDCHAIN– which is constructed from SeqPoW and Nakamoto consensus – does not belong to any of them. Compared to existing paradigms, RANDCHAIN is simple, secure, scalable, energy-efficient and decentralised while relying on weak assumptions.

DRBs from external randomness source. Some DRBs extract randomness from real-world randomness source, including financial data [13] and public blockchains [11], [14], [15]. Such DRBs introduce little communication overhead. However, these DRB protocols' security relies on the randomness source's security. For example, if the randomness source is biasible, then these DRB protocols are likely to be biasible as well. In addition, clients and/or servers should access the randomness source from trustworthy communication channels. Otherwise, an adversary who hijacks the channels can bias the randomness arbitrarily.

DRG-based DRBs. A large number of DRB protocols are constructed by executing a Distributed Randomness Generation (DRG) protocol in rounds. DRG allows a group of nodes to generate a random output or a batch of random outputs. It has a well-known variant called *multi-party coin tossing/flipping* [78]–[81], where the random output is only a binary bit. DRG can be constructed from various cryptographic primitives, such as commitments [12], [82], threshold signatures [3], [7], [48], VRFs [2], [10], [83]–[85], secret sharing [1], [8], [9], [49], [52] and homomorphic encryption [50].

There are some issues in DRG-based DRBs. First, if the DRG relies on a leader, then a leader should be elected for every round. The leader is elected either by a trusted third party or running a leader election protocol. Existing research [86] shows that constructing leader election protocols is challenging. Boneh et al. [87] propose two leader election constructions, which however rely on an RB in return. Second, if the network is not synchronous, then the DRG-based DRB should rely on a *pacemaker* [88] for liveness. The pacemaker is responsible to inform nodes when to start a new round. Without synchrony and pacemaker, a node cannot know whether other nodes have received its message, and nodes may not agree on the round number. When this happens, the system will lose liveness. Last, a DRG-based DRB inherits the assumption, security and performance from its used DRG. If the DRG does not scale, then DRBs based on this DRG cannot scale as well. Existing research [9], [89] shows that existing DRG protocols either rely on strong assumptions, fail to be unpredictable, or suffer from high communication complexity of more than $O(n^2)$. This makes DRG-based DRBs hard to scale.

DRBs from iteratively sequential functions. DRBs can also be constructed from iteratively sequential functions $f(\cdot)$. Given a random seed, $f(\cdot)$ can produce random outputs

continuously. As $f(\cdot)$ is deterministic, no one can bias random outputs. As $f(\cdot)$ is sequential, no one can obtain outputs without computing $f(\cdot)$ honestly. Lenstra and Wesolowski [19] and Ephraim et. al. [90] construct DRBs from Sloth and Continuous VDFs, respectively.

The main challenge of this paradigm is *frontrunning*, where some nodes always learn random outputs earlier than others. In time-sensitive applications such as decentralised exchanges [68], nodes may frontrun for gaining extra profit, which compromises fairness. In this paradigm of DRBs, two scenarios lead to frontrunning behaviours. First, if one can learn the random seed earlier than others, then it can precompute and learn random outputs earlier than others. This can be solved by a trusted setup, e.g., the Sapling ceremony for Zcash [91]. Second, nodes with faster processors learn random outputs earlier than others. This is challenging to solve given the deterministic nature of iteratively sequential functions.

RandRunner [51] extends this paradigm by allowing nodes to execute iterations of $f(\cdot)$ in turn. RandRunner does not prevent frontrunning either: for each round, the leader always learns the random output earlier than other nodes. In addition, RandRunner relies on a leader election, which as discussed can be challenging in dynamic networks.

8. Conclusion and future work

In this paper, we propose RandChain, a new family of Decentralised Randomness Beacon (DRB) protocols that are simple, secure and scalable. To construct RandChain, we introduced Sequential Proof-of-Work (SeqPoW), a variant of Proof-of-Work that is sequential, i.e., non-parallelisable. For SeqPoW, we provided concrete constructions of SeqPoW, and showed that SeqPoW is practical and useful for various cryptographic protocols. For RandChain, we showed that while inheriting simplicity and scalability from PoW-based consensus, RandChain remains energy-efficient and decentralised. Compared to existing DRBs, RandChain explores a new direction with several unique design choices, which may inspire new research in designing secure DRBs.

Competitive nodes v.s. collaborative nodes. In RandChain, nodes are *competitive* with each other: each node tries to propose the next random output earlier than others. Meanwhile, in most existing DRBs – especially DRG-based DRBs – nodes are *collaborative*: nodes contribute their local random outputs and combine them to a unique one.

Compared to collaborative DRBs, competitive DRBs introduce less communication overhead. In competitive DRBs, each random output is generated by a single node, and a single message, i.e., a block, needs to be propagated for each random output. Meanwhile in collaborative DRBs, the majority of nodes should contribute to the randomness, and all of these nodes need to broadcast some messages. This introduces non-negligible communication overhead.

Reusing entropy v.s. regenerating entropy. Consider RandChain works in an ideal setting, where all nodes mine at the same speed and are correct, and all messages are delivered instantly. Given the latest block, who solves SeqPoW first is deterministic. Given the SeqPoW puzzle, the solution and the random output are also deterministic. Thus, in this ideal setting, RandChain resembles a DRB based on iteratively sequential functions: it takes the genesis block as the random seed and nodes in turn increments the iteratively sequential function. As analysed in §7, DRBs based on iteratively sequential functions are strongly unpredictable, but suffer from frontrunning attacks.

Uncontrollable entropy v.s. node-chosen entropy. Then, consider RandChain works in real-world settings where nodes can be Byzantine and messages are delivered with random latency. Given the latest block, the next output are no longer determined. First, the node A that should have found the first SeqPoW solution may be offline or crashed. Second, A may propagate its block slower than another node B who finds a SeqPoW solution a bit later than A.

This means that in addition to the genesis block, nodes' Byzantine behaviours and network delay also affect the outputs of RandChain. The adversary cannot control both entropy sources, so frontrunning won't always succeed. This makes RandChain even more secure than DRBs based on iteratively sequential functions.

Acknowledgement

We thank Jieyi Long for discussions on the "winner-takes-all" problem, and Omer Shlomovtis for insightful comments.

References

[1] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*, Springer, 2017, pp. 357–388.

[2] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2018, pp. 66–98.

[3] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.

[4] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[5] L. Ren, "Analysis of nakamoto consensus.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 943, 2019.

[6] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: Simple and practical synchronous state machine replication," *IACR Cryptology ePrint Archive*, vol. 2019, p. 270, 2019.

[7] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[8] I. Cascudo and B. David, "Scrape: Scalable randomness attested by public entities," in *International Conference on Applied Cryptography and Network Security*, Springer, 2017, pp. 537–556.

[9] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 32–48.

[10] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[11] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source.," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 1015, 2015.

[12] *Randao: A dao working as rng of ethereum*, last accessed on 02/08/2020. [Online]. Available: https://github.com/randao/randao.

[13] J. Clark and U. Hengartner, "On the use of financial data as a random beacon.," *EVT/WOTE*, vol. 89, 2010.

[14] M. Andrychowicz and S. Dziembowski, "Distributed cryptography based on the proofs of work.," *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 796, 2014.

[15] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.

[16] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[17] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Secure information networks*, Springer, 1999, pp. 258–272.

[18] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*, Springer, 2018, pp. 757–788.

[19] A. K. Lenstra and B. Wesolowski, "A random zoo: Sloth, unicorn, and trx.," *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 366, 2015.

[20] C. Dwork and M. Naor, "Pricing via processing or combatting junk mail," in *Annual International Cryptology Conference*, Springer, 1992, pp. 139–147.

[21] K. Pietrzak, "Simple verifiable delay functions," in *10th innovations in theoretical computer science conference (itcs 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[22] B. Wesolowski, "Efficient verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 379–407.

[23] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[24] M. Mahmoody, T. Moran, and S. Vadhan, "Publicly verifiable proofs of sequential work," in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, 2013, pp. 373–388.

[25] N. Döttling, S. Garg, G. Malavolta, and P. N. Vasudevan, "Tight verifiable delay functions.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 659, 2019.

[26] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake," *self-published paper, August*, vol. 19, p. 1, 2012.

[27] *Pos based on synthetic pow using vdf and vrf*, last accessed on 01/08/20. [Online]. Available: https://ethresear.ch/t/pos-based-on-synthetic-pow-using-vdf-and-vrf/7271.

[28] *Nakamoto consensus with vdf and vrf*, last accessed on 01/08/20. [Online]. Available: https://ethresear.ch/t/nakamoto-consensus-with-vdf-and-vrf/5671.

[29] J. Long and R. Wei, "Nakamoto consensus with verifiable delay puzzle," *arXiv preprint arXiv:1908.06394*, 2019.

[30] H. Abusalah, C. Kamath, K. Klein, K. Pietrzak, and M. Walter, "Reversible proofs of sequential work," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 277–291.

[31] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of Cryptography Conference*, Springer, 2008, pp. 1–18.

[32] M. Naor, O. Paneth, and G. N. Rothblum, "Incrementally verifiable computation via incremental pcps," in *Theory of Cryptography Conference*, Springer, 2019, pp. 552–576.

[33] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer, "Recursive composition and bootstrapping for snarks and proof-carrying data," in *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, 2013, pp. 111–120.

[34] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1021, 2019.

[35] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Proof-carrying data from accumulation schemes.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 499, 2020.

[36] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang, "Diogenes: Lightweight scalable rsa modulus generation with a dishonest majority.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 374, 2020.

[37] M. Chen, R. Cohen, J. Doerner, Y. Kondi, E. Lee, S. Rosefield, and A. Shelat, "Multiparty generation of an rsa modulus.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 370, 2020.

[38] D. Boneh, B. Bünz, and B. Fisch, "A survey of two verifiable delay functions.," *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[39] V. Attias, L. Vigneri, and V. Dimitrov, "Implementation study of two verifiable delayfunctions.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 332, 2020.

[40] "Randchain," [Online]. Available: ANONYMISEDFORSUBMISSION.

[41] "Crates/rug," last accessed on 01/08/20. [Online]. Available: https://crates.io/crates/rug.

[42] "Cargo-bench," last accessed on 01/08/20. [Online]. Available: https://doc.rust-lang.org/cargo/commands/cargo-bench.html.

[43] "Criterion.rs," last accessed on 01/08/20. [Online]. Available: https://github.com/bheisler/criterion.rs.

[44] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2015, pp. 281–310.

[45] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2017, pp. 643–673.

[46] J. A. Garay, A. Kiayias, and N. Leonardos, "Full analysis of nakamoto consensus in bounded-delay networks.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 277, 2020.

[47] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 88, 2020.

[48] E. Kokoris-Kogias, A. Spiegelman, D. Malkhi, and I. Abraham, "Bootstrapping consensus without trusted setup: Fully asynchronous distributed key generation.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1015, 2019.

[49] I. Cascudo and B. David, "Albatross: Publicly attestable batched randomness based on secret sharing,"

[50] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1320, 2019.

[51] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, *Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness*, Cryptology ePrint Archive, Report 2020/942, https://eprint.iacr.org/2020/942, 2020.

[52] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, Ieee, 2017, pp. 444–460.

[53] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol with chains of variable difficulty," in *Annual International Cryptology Conference*, Springer, 2017, pp. 291–323.

[54] L. Kiffer, R. Rajaraman, and A. Shelat, "A better method to analyze blockchain consistency," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 729–744.

[55] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Everything is a race and nakamoto always wins," *arXiv preprint arXiv:2005.10484*, 2020.

[56] P. Gazi, A. Kiayias, and A. Russell, "Tight consistency bounds for bitcoin," 2020.

[57] "Bitcoin's energy consumption 'equals that of switzerland'," last accessed on 01/08/20. [Online]. Available: https://www.bbc.com/news/technology-48853230#:~:text=Bitcoin%20uses%20as%20much%20energy,the%20University%20of%20Cambridge%20shows.&text=Currently%2C%20the%20tool%20estimates%20that,0.21%25%20of%20the%20world's%20supply..

[58] *Amd breaks 8ghz overclock with upcoming fx processor, sets world record*, last accessed on 01/08/20. [Online]. Available: http://hothardware.com/News/AMD-Breaks-Frequency-Record-with-Upcoming-FX-Processor/.

[59] "Why has cpu frequency ceased to grow?," last accessed on 01/08/20. [Online]. Available: https://software.intel.com/content/www/us/en/develop/blogs/why-has-cpu-frequency-ceased-to-grow.html.

[60] "Supranational/vdf-fpga-round1-results," last accessed on 01/08/20. [Online]. Available: https://github.com/supranational/vdf-fpga-round1-results.

[61] "Supranational/vdf-fpga-round2-results," last accessed on 01/08/20. [Online]. Available: https://github.com/supranational/vdf-fpga-round2-results.

[62] "Supranational/vdf-fpga-round3-results," last accessed on 01/08/20. [Online]. Available: https://github.com/supranational/vdf-fpga-round3-results.

[63] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*, Springer, 2014, pp. 436–454.

[64] J. Bonneau, E. W. Felten, S. Goldfeder, J. A. Kroll, and A. Narayanan, "Why buy when you can rent? bribery attacks on bitcoin consensus," 2016.

[65] K. Liao and J. Katz, "Incentivizing blockchain forks via whale transactions," in *International Conference on Financial Cryptography and Data Security*, Springer, 2017, pp. 264–279.

[66] A. Judmayer, N. Stifter, A. Zamyatin, I. Tsabary, I. Eyal, P. Gazi, S. Meiklejohn, and E. R. Weippl, "Pay-to-win: Incentive attacks on proof-of-work cryptocurrencies.," *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 775, 2019.

[67] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[68] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 566–583.

[69] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[70] D. Malkhi and M. Reiter, "Byzantine quorum systems," *Distributed computing*, vol. 11, no. 4, pp. 203–213, 1998.

[71] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[72] T.-H. H. Chan, R. Pass, and E. Shi, "Consensus through herding," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 720–749.

[73] T. Rocket, M. Yin, K. Sekniqi, R. van Renesse, and E. G. Sirer, "Scalable and probabilistic leaderless bft consensus through metastability," *arXiv preprint arXiv:1906.08936*, 2019.

[74] "Bch avalanche transactions show finality speeds 10x faster than ethereum," last accessed on 01/08/20. [Online]. Available: https : / / news . bitcoin . com / bch - avalanche - transactions - show - finality - speeds - 10x - faster-than-ethereum/.

[75] "Vrf-based mining: Simple non-outsourceable cryptocurrency mining," last accessed on 01/08/20. [Online]. Available: https://github.com/DEX-ware/vrf-mining/blob/master/paper/main.pdf.

[76] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, IEEE, 1999, pp. 120–130.

[77] D. Stutzbach and R. Rejaie, "Understanding churn in peer-to-peer networks," in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, ACM, 2006, pp. 189–202.

[78] M. Blum, "Coin flipping by telephone a protocol for solving impossible problems," *ACM SIGACT News*, vol. 15, no. 1, pp. 23–27, 1983.

[79] T. Moran, M. Naor, and G. Segev, "An optimally fair coin toss," in *Theory of Cryptography Conference*, Springer, 2009, pp. 1–18.

[80] A. Beimel, E. Omri, and I. Orlov, "Protocols for multiparty coin toss with dishonest majority," in *Annual Cryptology Conference*, Springer, 2010, pp. 538–557.

[81] K.-M. Chung, Y. Guo, W.-K. Lin, R. Pass, and E. Shi, "Game theoretic notions of fairness in multi-party coin toss," in *Theory of Cryptography Conference*, Springer, 2018, pp. 563–596.

[82] B. Awerbuch and C. Scheideler, "Robust random number generation for peer-to-peer systems," in *International Conference On Principles Of Distributed Systems*, Springer, 2006, pp. 275–289.

[83] Y. Dodis, "Efficient construction of (distributed) verifiable random functions," in *International Workshop on Public Key Cryptography*, Springer, 2003, pp. 1–17.

[84] V. Kuchta and M. Manulis, "Unique aggregate signatures with applications to distributed verifiable random functions," in *International Conference on Cryptology and Network Security*, Springer, 2013, pp. 251–270.

[85] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, "Fully distributed verifiable random functions and their application to decentralised random beacons.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 96, 2020.

[86] C. Gómez-Calzado, A. Lafuente, M. Larrea, and M. Raynal, "Fault-tolerant leader election in mobile dynamic distributed systems," in *2013 IEEE 19th Pacific Rim International Symposium on Dependable Computing*, IEEE, 2013, pp. 78–87.

[87] D. Boneh, S. Eskandarian, L. Hanzlik, and N. Greco, "Single secret leader election.," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 25, 2020.

[88] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[89] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 41–61.

[90] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass, "Continuous verifiable delay functions," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2020, pp. 125–154.

[91] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox, "Zcash protocol specification," *GitHub: San Francisco, CA, USA*, 2016.

APPENDIX

1. Security proof of two SeqPoW constructions

We formally prove security guarantee of two SeqPoW constructions. We start from $\mathsf{SeqPoW}_{\mathsf{VDF}}$.

Lemma 5. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies completeness.*

Proof. Assuming there is an (λ, ψ, T)-valid tuple $(pp, sk, i, x, S_i, \pi_i)$. By *completeness* and Lemma 1, $\mathsf{VDF.Verify}(\cdot)$ will pass. As hash functions are deterministic, difficulty check will pass. Therefore,

$$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i) = 1$$

□

Lemma 6. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies soundness.*

Proof. We prove this by contradiction. Assuming there exists a tuple $(pp, sk, i, x, S_i, \pi_i)$ that is not (λ, ψ, T)-valid such that

$$\mathsf{SeqPoW}_{\mathsf{VDF}}.\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i) = 1$$

By *soundness* and Lemma 1, if (y, y^+, π^+, ψ) is generated by $\mathcal{A}$, $\mathsf{VDF.Verify}(\cdot)$ will return 0. As hash functions are deterministic, if $S_i > \frac{2^\kappa}{T}$, difficulty check will return 0. Thus, if $(pp, sk, i, x, S_i, \pi_i)$ is not (λ, ψ, T)-valid, then the adversary should break *soundness*. Thus, this assumption contradicts *soundness*. □

Lemma 7. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies hardness.*

Proof. We prove this by contradiction. Assuming

$$\Pr\left[b_{i+1}=1 \;\middle|\; \begin{array}{c} S_{i+1}, b_{i+1} \leftarrow \\ \mathsf{Solve}(pp, sk, T, S_i) \end{array}\right] > \frac{1}{T}$$

By *sequentiality*, the value of S_{i+1} is unpredictable before finishing $\mathsf{Solve}(\cdot)$. By pseudorandomness of hash functions, $H(pk\|S_{i+1})$ is uniformly distributed, and the probability that $H(pk\|S_{i+1}) \leq \frac{2^\kappa}{T}$ is $\frac{1}{T}$ with negligible probability. This contradicts the assumption. □

Lemma 8. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *does not satisfy uniqueness.*

Proof. By *hardness*, each of S_i has the probability $\frac{1}{T}$ to be a valid solution. As i can be infinite, with overwhelming probability, there exists more than one honest tuple $(pp, sk, i, x, S_i, \pi_i)$ such that $H(pk\|S_i) \leq \frac{2^\kappa}{T}$. □

Lemma 9. *If the underlying VDF satisfies σ-sequentiality, then* $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies σ-sequentiality.*

Proof. We prove this by contradiction. Assuming there exists $\mathcal{A}_1$ which runs in less than time $\sigma(\psi i)$ such that

$$\Pr\left[\begin{array}{c} (pp, sk, i, x, S_i, \pi_i) \\ \in \mathcal{H} \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \mathsf{Setup}(\lambda, \psi, T) \\ (sk, pk) \xleftarrow{R} \mathsf{Gen}(pp) \\ \mathcal{A}_1 \leftarrow \mathcal{A}_0(\lambda, pp, sk) \\ S_i \leftarrow \mathcal{A}_1(i, x) \\ \pi_i \leftarrow \mathsf{Prove}(pp, sk, i, x, S_i) \end{array}\right]$$

By *σ-sequentiality*, $\mathcal{A}_1$ cannot solve $\mathsf{VDF.Eval}(pp_{\mathsf{VDF}}, y, \psi)$ within $\sigma(\psi)$. By Lemma 1, S_i can and only can be computed by composing $\mathsf{VDF.Eval}(pp_{\mathsf{VDF}}, y, \psi)$ for i times, which cannot be solved within $\sigma(\psi i)$. This contradicts the assumption. □

The completeness, soundness, hardness and sequentiality proofs of $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ are identical with $\mathsf{SeqPoW}_{\mathsf{VDF}}$'s. We prove $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ satisfies uniqueness below.

Lemma 10. $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ *satisfies uniqueness.*

Proof. We prove this by contradiction. Assuming there exists two (λ, ψ, T)-valid tuples $(pp, sk, i, x, S_i, \pi_i)$ and $(pp, sk, i, x, S_i, \pi_i)$ where $j < i$. According to $\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Solve}(\cdot)$, we have $H(pk\|S_i) \leq \frac{2^\kappa}{T}$ and $H(pk\|S_j) \leq \frac{2^\kappa}{T}$, and initial difficulty check in $\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Verify}(\cdot)$ will pass. However, in the for loop of $\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Verify}(\cdot)$, if S_i is valid, then verification of S_j will fail. Then, $\mathsf{SeqPoW}_{\mathsf{Sloth}}.\mathsf{Verify}(\cdot)$ returns 0, which contradicts the assumption. □