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 periodically generates publicly verifiable randomness, which plays critical roles in various cryptographic protocols. However, constructing DRB protocols is challenging. Existing DRB protocols suffer from either strong network synchrony assumptions, high communication complexity or attacks.**

In this paper, we propose RANDCHAIN, a new family of permissioned DRB protocols. RANDCHAIN is constructed from Sequential Proof-of-Work (SeqPoW) – a Proof-of-Work (PoW) variant that is sequential, i.e., non-parallelisable – and Nakamoto consensus. In RANDCHAIN, nodes jointly maintain a blockchain, and each block attaches a random output. Given the last block, each node deterministically derives a SeqPoW puzzle, and keeps solving SeqPoW (aka. mining) until finding a valid solution, of which the value is unpredictable. The first node solving SeqPoW includes a block to the blockchain. The SeqPoW solution's hash is the random output in this block. RANDCHAIN applies Nakamoto consensus so that nodes agree on a unique blockchain.

We formalise SeqPoW, propose two constructions, prove their security and evaluate their performance. Our results show that SeqPoW is practical. We then formalise DRBs and analyse RANDCHAIN's security. Our analysis shows that RANDCHAIN implements a DRB protocol and can produce strongly unpredictable and unbiasible randomness. While as simple and scalable as PoW-based consensus, RANDCHAIN achieves better energy efficiency and decentralisation than PoW-based consensus, thanks to the non-parallelisable mining. We also discuss considerations of deploying RANDCHAIN in practice, including adding finality, mining non-outsourceability, and difficulty adjustment to RANDCHAIN.

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, 2, 3, 4].

However, constructing secure and scalable DRBs is challenging, mainly due to four obstacles. First, DRBs usually execute in rounds, leading to the *lock-step synchrony* assumption, 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. Second, DRBs usually suffer from high communication overhead – usually no less than $O(n^2)$ where n is the number of nodes. Third, some DRBs can be vulnerable [5, 2, 6, 7], including being biasible, i.e., the adversary can bias the randomness to its preferred values, and being predictable, i.e., the adversary can predict randomness in the future. Last, some DRBs [6, 8, 9, 10, 7] rely on trustworthy components such as full-fledged blockchains and smart contracts, which are not always realistic.

In this paper, we propose RANDCHAIN– a new family of DRB protocols constructed from Sequential Proof-of-Work (SeqPoW) – a Proof-of-Work (PoW) [11] variant that is non-parallelisable. In RANDCHAIN, nodes jointly maintain a *blockchain*, i.e., a chain of blocks, and each block consists of a random output. Given the blockchain, each participant keeps solving a SeqPoW puzzle derived from the last block. The first participant solving the SeqPoW puzzle proposes a block with the next randomness, of which the value is decided by the SeqPoW solution. RANDCHAIN employs Nakamoto consensus [12] so that nodes agree on the longest blockchain.

While inheriting PoW-based consensus' simplicity, security and scalability, RANDCHAIN is also energy efficient and decentralised. As SeqPoW is non-parallelisable, each node can only use a single processor for SeqPoW mining. This prevents mining from marketisation and thus keeps RANDCHAIN energy efficient, unlike PoW-based blockchains where nodes invest high-end mining hardware with massively parallel processors. In addition, SeqPoW mining can only be accelerated using processors with high clock rate, which is hard to improve further, given the voltage limit of processors [13]. Thus, powerful nodes cannot obtain much speedup on mining, leading to 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 randomness 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 one. Second, RANDCHAIN reuses random entropy from bootstrap, whereas nodes in existing DRBs usually sample new random entropy for

¶ Corresponding author.

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

Sequential Proof-of-Work (SeqPoW). We propose SeqPoW, a PoW [11] variant that is *non-parallelisable*. To solve a PoW puzzle, the prover searches for a random string (aka. nonce) that satisfies a difficulty parameter. The search is done by enumerating all possible nonces, which can be parallelised using multiple processors. Meanwhile in a SeqPoW, the prover takes a predefined nonce, and executes an *iteratively sequential function* over this nonce. The prover keeps incrementing the iteratively sequential function until finding an output that satisfies the difficulty parameter. In this way, one cannot accelerate solving SeqPoW by parallelism. Like VDFs, the only way of accelerating solving SeqPoW is to use a processor with high clock rate, which gives little optimisation space.

We formalise SeqPoW and propose two SeqPoW constructions, one is from Verifiable Delay Functions (VDFs) [14] and the other is from the Sloth hash function [15]. We formally analyse the security and efficiency of both SeqPoW constructions. We also implement them and evaluate their performance. Our results show that SeqPoW from Pietrzak's VDF [16] exhibits the best candidate, given the efficient proof generation and verification and the acceptable proof size.

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– a new family of DRBs that is simple, secure and scalable in the meantime. In RandChain, nodes jointly maintain a *blockchain*, each of which includes a random output. Nodes continuously work on SeqPoW *mining*, where a participant can append a new block only after solving a SeqPoW puzzle. Each participant derives its SeqPoW input from the precedent block and its identity in the system, and cannot choose its own SeqPoW input freely. Thus, given the local view of the blockchain, the participant has only one valid SeqPoW input, and cannot accelerate SeqPoW mining by parallelism. RandChain implements Nakamoto consensus, i.e., the longest-chain rule where nodes agree on the longest blockchain, so that nodes have a consistent view on the blockchain.

We present the detailed construction of RandChain and formally analyse its security. We formally define DRBs with four properties, namely *consistency*, *liveness*, *uniform-distribution* and *unpredictability*. *Consistency* and *liveness* follow the definitions in Nakamoto consensus [12]. *Uniform-distribution* requires each output is pseudorandom, i.e., uniformly distributed. *Unpredictability* requires the adversary cannot predict random outputs either generated by other nodes or by itself. Our analysis shows that RandChain implements a DRB protocol with strong unpredictability guarantee. In addition, we show that while inheriting PoW-based consensus'

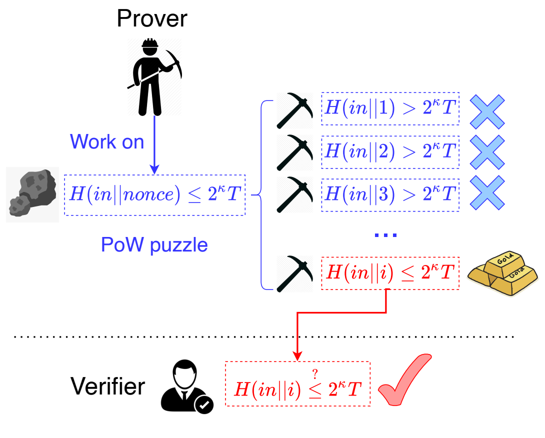


Fig. 1: Proof-of-Work.

simplicity, security and scalability, RandChain is energy efficient and decentralised. We also analyse several considerations for deploying RandChain, including supporting *finality* [17] i.e., no block is committed and later reverted, *non-outsourceability*, i.e., preventing nodes from solving others' SeqPoW puzzles, and dynamic difficulty adjustment.

Paper organisation. Section 2 describes the background. Section 3 introduces SeqPoW, including definitions, constructions and analysis. Section 4-6 describe the construction, security analysis and practical considerations of RandChain, respectively. 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.

2. Preliminaries

Notations. Let $H(\cdot)$ and $H'(\cdot)$ be two different hash functions that take an arbitrarily long string $\{0,1\}^*$ and output a fixed-length string $\{0,1\}^\kappa$. Let G be a cyclic group, and $H_G(\cdot)$ be a hash function that takes an arbitrarily long string $\{0,1\}^*$ to a point on G.

PoW and PoW-based consensus. Proof-of-Work (PoW) can be seen as 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 i which makes $H(in\|i) \leq \frac{2^\kappa}{T}$, where 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 [12] – 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 [18]. 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.

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 [16, 19] and repeated square rooting [15] on cyclic groups.

Time-based cryptography. Iteratively sequential functions are the key ingredient of time-based cryptographic primitives. Rivest, Shamir and Wagner first introduce the timelock puzzle and constructs it from repeated squaring on RSA groups [20]. Proofs of Sequential Work (PoSW) [21] allow to prove he has finished an amount of sequential computation with a succinct proof. Verifiable Delay Functions (VDFs) [14] 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

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

$\mathsf{Setup}(\lambda) \to pp$: 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.

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

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

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

that satisfies the following properties

- *VDF-Completeness*: For all λ, x and t,

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

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

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

- *σ-VDF-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 $\sigma(t)$,

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

VDFs can be constructed from two ingredients: 1) an iteratively sequential function $f(\cdot)$, and 2) a succinct argument on $f(\cdot)$'s execution results. For example, two prevalent VDFs [19, 16] 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* [22].

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

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

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

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

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 accelerated by parallelism. We formally define SeqPoW, provide two constructions with security proofs, and evaluate their performance. We also discuss other possible constructions and applications of SeqPoW.

A. 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 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.

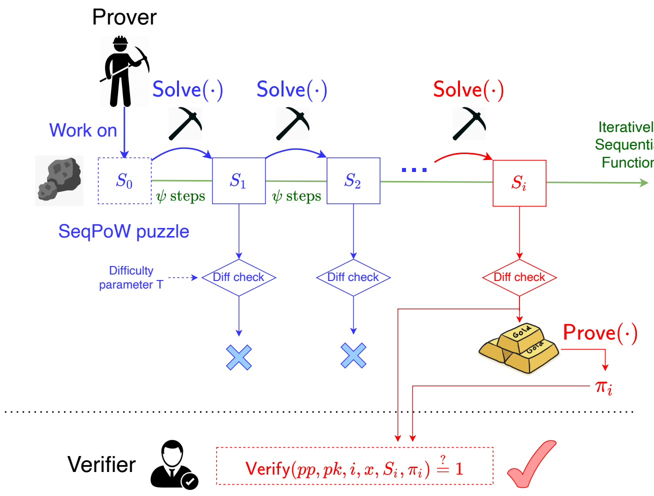


Fig. 2: Sequential Proof-of-Work.

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 only run a single processor, which prevents marketisation of mining and gives little optimisation space of mining speed. Later in §5-E we will discuss this in detail.

PoS-based consensus. In Proof-of-Stake (PoS)-based consensus [23], each node's chance of mining the next block is in proportion to its stake, which can be deposit or balance. In some PoS-based consensus designs [24, 25, 26], 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. To avoid the "winner-takes-all" problem, one can replace VDF with SeqPoW, and specify the difficulty parameter in inverse proportion to each node's stake. As the number of iterations taken to solve a SeqPoW is randomised, the node with most stake is not always the winner, but only with higher chance.

B. 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

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

$\mathsf{Setup}(\lambda, \psi, T) \rightarrow pp$: 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 $H : \mathcal{Y} \rightarrow \mathcal{X}$. We assume $\mathcal{X}$ is efficiently sampleable.

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

$\mathsf{Init}(pp, sk, x) \rightarrow (S_0, \pi_0)$: 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.

$\mathsf{Solve}(pp, sk, S_i) \rightarrow (S_{i+1}, b_{i+1})$: 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 $b_{i+1} \in \{0, 1\}$.

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

$\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i) \rightarrow \{0, 1\}$: 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)**.** For all λ, ψ, T, sk, and x, let $pp \leftarrow \mathsf{Setup}(\lambda, \psi, T)$. Then it holds that

- Let
$$(S_0, \pi_0) \leftarrow \mathsf{Init}(pp, sk, x)$$
Then, tuple $(pp, sk, 0, x, S_0, \pi_0)$ is (λ, ψ, T)-honest.
- For $i \in [1, \infty)$, let $(pp, sk, i-1, x, S_{i-1}, \pi_{i-1})$ be a (λ, ψ, T)-honest tuple, and
$$(S_i, b_i) \leftarrow \mathsf{Solve}(pp, sk, S_{i-1})$$
$$\pi_i \leftarrow \mathsf{Prove}(pp, sk, i, x, S_i)$$
Then, tuple $(pp, sk, i, x, S_i, \pi_i)$ is (λ, ψ, T)-honest.

Definition 5 (Valid tuple)**.** For all λ, ψ, and $x \in \{0, 1\}^*$, let $pp \leftarrow \mathsf{Setup}(\lambda, \psi, T)$. We say 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 *SeqPoW-Completeness*, *SeqPoW-Soundness* and *SeqPoW-Sequentiality*, and one of *SeqPoW-Hardness* or *SeqPoW-Uniqueness*.

Definition 6 (SeqPoW-Completeness)**.** For any (λ, ψ, T)-valid tuple $(pp, sk, i, x, S_i, \pi_i)$,

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

where and pk is the public key associated with sk.

Definition 7 (SeqPoW-Soundness)**.** For all λ, ψ, and T, let $\mathcal{V}$ be the set of (λ, ψ, T)-valid tuples. There exists no tuple $(pp, sk, i, x, S_i, \pi_i) \notin \mathcal{V}$ such that

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

where and pk is the public key associated with sk.

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

Definition 8 (SeqPoW-Hardness)**.** For all (λ, ψ, T)-honest tuple $(pp, sk, i, x, S_i, \pi_i)$,

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

We also consider a stronger notion of *SeqPoW-Hardness*, which we call *SeqPoW-Uniqueness*. *SeqPoW-Uniqueness* requires that, each SeqPoW puzzle only has a single valid solution S_i. Before finding S_i each $\mathsf{Solve}(\cdot)$ attempt follows the definition *SeqPoW-Hardness*, but after finding the solution S_i no $\mathsf{Solve}(\cdot)$ attempt leads to a valid solution.

Definition 9 (SeqPoW-Uniqueness)**.** For all λ, ψ, T, sk, and x, there is only one (λ, ψ, T)-valid tuple $(pp, sk, i, x, S_i, \pi_i)$. In addition, for all (λ, ψ, T)-honest tuple $(pp, sk, j, x, S_j, \pi_j)$,

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

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

Definition 10 (σ-SeqPoW-Sequentiality)**.** 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 $\sigma(\psi i)$,

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

C. Constructions

SeqPoW from self-composable VDFs. We first present a SeqPoW construction $\mathsf{SeqPoW_{VDF}}$ 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 $S_0 = H_G(pk \| x)$. 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. The prover can then compute a succinct argument of the statement $S_i = \mathsf{VDF.Eval}^i(pp, S_0, \psi)$ by running $\mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, S_0, S_i, i\psi)$. Note that by *VDF-Self-Composability*, $\mathsf{VDF.Eval}^i(pp, S_0, \psi) = \mathsf{VDF.Eval}(pp, S_0, i\psi)$. With the pk, i, S_i and π_i, the verifier then can check whether 1) S_i satisfies the difficulty T, and 2) $S_i = \mathsf{Eval}^i(pp, S_0, \psi)$. Figure 3 describes our $\mathsf{SeqPoW_{VDF}}$ construction in detail.

Unique SeqPoW from Sloth. $\mathsf{SeqPoW_{VDF}}$ does not provide *SeqPoW-Uniqueness*: the prover can keep incrementing the iteratively sequential function to find as many valid solutions as possible. We present $\mathsf{SeqPoW_{Sloth}}$ – a SeqPoW construction with *SeqPoW-Uniqueness*. $\mathsf{SeqPoW_{Sloth}}$ is based on Sloth [15], a hash function that supports hashchains with fast verification. Sloth employs repeated square rooting on a cyclic group as the iteratively sequential function. The prover square roots (on a cyclic group G) the input for a predefined number t of times to get the output. To verify the output, the verifier squares – which is $O(\log |G|)$ times faster than square rooting – the output for t times to recover the input. This means Sloth is *reversible* [27]: given the output, the verifier can recover intermediate results and the input. Same as in $\mathsf{SeqPoW_{VDF}}$, $\mathsf{SeqPoW_{Sloth}}$ 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, $\mathsf{SeqPoW_{Sloth}}$ only treats the first solution satisfying the difficulty as valid, which makes the solution unique. Figure 4 describes the detailed construction of $\mathsf{SeqPoW_{Sloth}}$.

Other possible constructions. The main challenge of constructing SeqPoW is the succinct argument of iteratively sequential functions. In addition to VDFs and Sloth, Incremental Verifiable Computation (IVC) [28] can also provide such arguments. IVC is a powerful primitive that, for every step of a computation, a prover can produce a succinct proof by updating the last step's proof rather than computing the proof from scratch. Given an input, an output, a series of computation and the proof, one can verify current output is produced from the series of computation on the input.

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 [28, 29, 30, 31, 32]. This makes the construction inefficient and the 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 $\mathsf{SeqPoW.Prove}(\cdot)$.

D. Security and efficiency analysis

Security. We defer full security proofs to Appendix A. Proofs of *SeqPoW-Completeness* and *SeqPoW-Soundness* directly follow the completeness, soundness and self-composability of Sloth and VDFs. By pseudorandomness of hash functions and sequentiality of Sloth and VDFs, $\mathsf{Solve}(\cdot)$ outputs unpredictable solutions and the probability that the solution satisfies the difficulty is $\frac{1}{T}$, leading to *SeqPoW-Hardness*. *SeqPoW-Sequentiality* directly follows the sequentiality and self-composability of Sloth and VDFs.

Efficiency. Table I summarises the efficiency analysis of two SeqPoW constructions. $\mathsf{SeqPoW_{VDF}}$ and $\mathsf{SeqPoW_{Sloth}}$ employ repeated squaring on an RSA group and repeated square rooting on a group of prime order as the iteratively

```
$\mathsf{SeqPoW_{VDF}.Setup}(\lambda, \psi, T)$
    $pp_{\mathsf{VDF}} \leftarrow \mathsf{VDF.Setup}(\lambda)$
    $G, H_G \leftarrow pp_{\mathsf{VDF}}$
    $pp \leftarrow (\psi, G, H_G, T)$
    **return** $pp$
```

```
$\mathsf{SeqPoW_{VDF}.Gen}(pp)$
    $(\psi, G, H_G, T) \leftarrow pp$
    Sample random $sk \in \mathbb{N}$
    $pk \leftarrow sk \cdot G$
    **return** $(sk, pk)$
```

```
$\mathsf{SeqPoW_{VDF}.Init}(pp, sk, x)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $pk \leftarrow sk \cdot G$
    $S_0 \leftarrow H_G(pk \| x)$
    **return** $S_0$
```

```
$\mathsf{SeqPoW_{VDF}.Solve}(pp, sk, S_i)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $pp_{\mathsf{VDF}} \leftarrow G, H_G$
    $pk \leftarrow sk \cdot G$
    $S_{i+1} \leftarrow \mathsf{VDF.Eval}(pp_{\mathsf{VDF}}, S_i, \psi)$
    $b_{i+1} \leftarrow H(pk \| S_{i+1}) \leq \frac{2^\kappa}{T}\ ?\ 1 : 0$
    **return** $(S_{i+1}, b_{i+1})$
```

```
$\mathsf{SeqPoW_{VDF}.Prove}(pp, sk, i, x, S_i)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $pk \leftarrow sk \cdot G$
    $pp_{\mathsf{VDF}} \leftarrow (G, H_G)$
    $S_0 \leftarrow H_G(pk \| x)$
    $\pi_{\mathsf{VDF}} \leftarrow \mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, S_0, S_i, i\psi)$
    **return** $\pi_{\mathsf{VDF}}$
```

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

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

```
$\mathsf{SeqPoW_{VDF}.Setup}(\lambda, \psi, T)$
    $pp_{\mathsf{VDF}} \leftarrow \mathsf{VDF.Setup}(\lambda)$
    $G, H_G \leftarrow pp_{\mathsf{VDF}}$
    $pp \leftarrow (\psi, G, H_G, T)$
    **return** $pp$
```

```
$\mathsf{SeqPoW_{VDF}.Gen}(pp)$
    $(\psi, G, H_G, T) \leftarrow pp$
    Sample random $sk \in \mathbb{N}$
    $pk \leftarrow sk \cdot G$
    **return** $(sk, pk)$
```

```
$\mathsf{SeqPoW_{Sloth}.Init}(pp, sk, x)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $pk \leftarrow sk \cdot G$
    $g \leftarrow H_G(pk \| x)$
    $S_0 \leftarrow g$
    **return** $S_0$
```

```
$\mathsf{SeqPoW_{Sloth}.Solve}(pp, sk, S_i)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $pk \leftarrow sk \cdot G$
    $S_{i+1} \leftarrow y^{\frac{1}{2^\psi}}$
    $b_{i+1} \leftarrow H(pk \| S_{i+1}) \leq \frac{2^\kappa}{T}\ ?\ 1 : 0$
    **return** $(S_{i+1}, b_{i+1})$
```

```
$\mathsf{SeqPoW_{Sloth}.Prove}(pp, sk, i, x, S_i)$
    **return** $\perp$
```

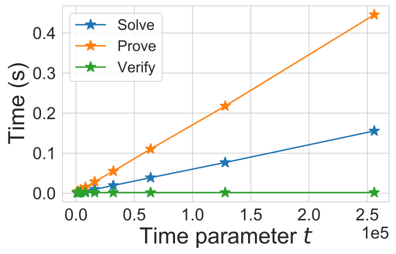
```
$\mathsf{SeqPoW_{Sloth}.Verify}(pp, pk, i, x, S_i, \pi_i)$
    $(\psi, G, H_G, T) \leftarrow pp$
    $y \leftarrow S_i$
    **if** $H(pk \| y) > \frac{2^\kappa}{T}$, **then return** 0
    **repeat** $i$ **times**
        $y \leftarrow y^{2^\psi}$
        **if** $H(pk \| y) \leq \frac{2^\kappa}{T}$ **then return** 0
    $g \leftarrow H_G(pk \| x)$
    **if** $g \neq y$ **then return** 0
    **return** 1
```

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

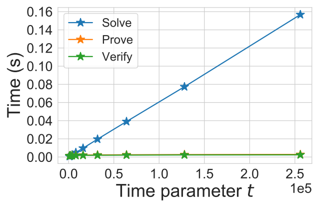
TABLE I: Efficiency of two SeqPoW constructions. ISF stands for iteratively sequential function.

	Construction		**Efficiency**			
	ISF	Proof	Solve$(\cdot)$	Prove$(\cdot)$	Verify$(\cdot)$	Proof size (Bytes)
$\mathsf{SeqPoW_{VDF}}$	Repeated SQ.	Wes19 [19]	$O(\psi)$	$O(\psi T)$	$O(\log \psi T)$	s
		Pie19 [16]	$O(\psi)$	$O(\sqrt{\psi T} \log \psi T)$	$O(\log \psi T)$	$s \log_2 \psi T$
$\mathsf{SeqPoW_{Sloth}}$	Repeated SQRT.	Repeated SQ.	$O(\psi)$	0	$O(\psi T)$	0

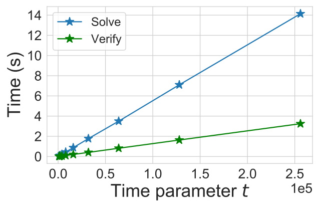
sequential function, respectively. Let s be the size (in Bytes) of an element on the group, and ψ be the step parameter. Each $\mathsf{Solve}(\cdot)$ executes ψ steps of the iteratively sequential function. By *SeqPoW-Hardness* and *SeqPoW-Uniqueness*, a node attempts $\mathsf{Solve}(\cdot)$ for T times 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. $\mathsf{SeqPoW_{VDF}}$ employs either Wesolowski's VDF (Wes19) [19] or Pietrzak's VDF (Pie19) [16] for succinct arguments of repeated squaring. According to existing analysis [33], the proving complexity, verification complexity and proof size of Wes19's succinct argument are $O(\psi T)$, $O(\log \psi T)$ and s Bytes, respectively; and the proving complexity, verification complexity and proof size of Pie19's succinct argument 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 $\mathsf{VDF.Prove}(\cdot)$ for both VDFs [19, 16, 34]. $\mathsf{SeqPoW_{Sloth}}$ does not have proofs; instead, the verifier keeps squaring the solution to recover the input and check whether the recovered input equals to the real one. This leads to verification complexity of $O(\psi T)$.



(a) SeqPoW$_{\mathsf{VDF}}$ + Wes19 [19].



(b) SeqPoW$_{\mathsf{VDF}}$ + Pie19 [16].



(c) SeqPoW$_{\mathsf{Sloth}}$.

Fig. 5: Evaluation of SeqPoW constructions.

E. Performance evaluation

We evaluate the performance of our two SeqPoW constructions. The code is available at Github[1]. Our results show that SeqPoW$_{\mathsf{VDF}}$ with Pie19 is the best among these constructions, given its efficient generation and verification of proofs and acceptable proof size.

Implementation. We implement the two SeqPoW constructions in Rust programming language[2]. We use the `rug`[3] crate for big integer arithmetic. We implement the RSA group with 1024-bit keys and the group of prime order. We implement the two 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 [16, 19, 15] without any optimisation.

Experimental setup. We benchmark Solve$(\cdot)$, Prove$(\cdot)$ and 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 SeqPoW$_{\mathsf{Sloth}}$ does not have Prove$(\cdot)$. We benchmark the performance using Rust's native benchmarking suite `cargo-bench`[4] and `criterion`[5]. 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 Solve$(\cdot)$ increases linearly with time parameter t. This is as expected as Solve$(\cdot)$ is dominated by the iteratively sequential function. For SeqPoW$_{\mathsf{VDF}}$ with Wes19, Prove$(\cdot)$ takes more time than Solve$(\cdot)$. This is because we implement the non-optimised proof generation algorithm. There have been optimised and parallelisable alternatives for Wes19's Prove$(\cdot)$ [34]. SeqPoW$_{\mathsf{VDF}}$ with Pie19 achieves ideal performance: Prove$(\cdot)$ and Verify$(\cdot)$ takes negligible time compared to Solve$(\cdot)$. Meanwhile, the proof size of Pie19 is also acceptable. Thus, without optimisation, SeqPoW$_{\mathsf{VDF}}$ with Pie19 more practical than with Wes19. For SeqPoW$_{\mathsf{Sloth}}$, Solve$(\cdot)$ is approximately five times slower than Verify$(\cdot)$. Although this is far from theoretically optimal value, i.e, $\log_2 |G| = 2048$ in our case [27], the verification overhead is acceptable when random outputs are not generated frequently.

[1]ANONYMISED
[2]https://www.rust-lang.org/
[3]https://crates.io/crates/rug
[4]https://doc.rust-lang.org/cargo/commands/cargo-bench.html
[5]https://github.com/bheisler/criterion.rs

4. RandChain: DRB from SeqPoW

Based on SeqPoW, we construct RandChain, a new family of DRBs. Figure 6 describes the full construction of RandChain.

System setting. RandChain works in a permissioned system. Nodes $\mathcal{P} = \{p_1, \dots, p_n\}$ register themselves into the system through a Public Key Infrastructure (PKI). PKI assigns each registered node $p_i \in \mathcal{P}$ with a pair of secret key sk_i and public key pk_i. 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 PKI. There can be unlimited number of registered nodes. Each node does not 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_i has its local view $\mathcal{C}_i$ of the blockchain. The local view $\mathcal{C}_i$ 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 mainChain$(\cdot)$ of Figure 6, node p_i considers the longest fork of $\mathcal{C}_i$ as the main chain $\mathcal{MC}_i$.

Process (Figure 7b). We describe the process of RandChain from a node p_i's perspective. Node p_i runs two routines: the synchronisation routine SyncRoutine$(\cdot)$ and the mining routine MineRoutine$(\cdot)$. In SyncRoutine$(\cdot)$, node p_i synchronises its local blockchain $\mathcal{C}_i$ with other nodes. The synchronisation process is same as in other blockchains: node p_i keeps receiving blocks from other nodes, verifying them, and adding valid blocks to its local blockchain $\mathcal{C}_i$.

In MineRoutine$(\cdot)$, node p_i keeps *mining*, i.e., adding new blocks, on the main chain $\mathcal{MC}_i$. To append a block to the blockchain, node p_i should solve a SeqPoW puzzle. In particular, node p_i finds its latest main chain $\mathcal{MC}_i$, and derives a SeqPoW puzzle from $\mathcal{MC}_i$'s last block. Then, node p_i executes SeqPoW.Init$(\cdot)$ to find the starting point

```
mainChain($\mathcal{C}_i$)
1 : $tmp \leftarrow \epsilon$
2 : for $\mathcal{C}_i^k \in \mathcal{C}_i$
3 :     $tmp \leftarrow max(\mathcal{C}_i^k, tmp)$
4 : return $\mathcal{MC}_i = tmp$

MainProcedure($pp, sk_i, pk_i$)
1 : Synchronise chain as $\mathcal{C}_i$
2 : MineRoutine($pp, sk_i, pk_i, \mathcal{C}_i$) in a thread
3 : SyncRoutine($pp, \mathcal{C}_i$) in a thread

SyncRoutine($pp, \mathcal{C}$)
1 : while $True$
2 :     Wait for a new block as $B$
3 :     $(h^-, h, i, S, pk, \pi) \leftarrow B$
4 :     if $h^- \notin \mathcal{C}_i$ then Discard $B$
5 :     if $h \neq H(pk\|S)$ then Discard $B$
6 :     if SeqPoW.Verify($pp, pk, i, h^-, S, \pi$) $= 0$
7 :         Discard $B$
8 :     Append $B$ to $\mathcal{C}_i$ after block with hash $h^-$
9 :     Propagate $B$

MineRoutine($pp, sk_i, pk_i, \mathcal{C}_i$)
1 : while $True$
2 :     $\mathcal{MC}_i \leftarrow$ mainChain($\mathcal{C}_i$)
3 :     $B^- \leftarrow \mathcal{MC}_i[-1]$
4 :     $i \leftarrow 0$
5 :     $S \leftarrow$ SeqPoW.Init($pp, sk_i, B^-.h$)
6 :     while $True$
7 :         $S, b \leftarrow$ SeqPoW.Solve($pp, sk_i, S$)
8 :         if $b = 1$ then $Break$
9 :         $i += 1$
10 :        $\mathcal{MC}_i' \leftarrow$ mainChain($\mathcal{C}_i$)
11 :        if $\mathcal{MC}_i \neq \mathcal{MC}_i'$
12 :            $\mathcal{MC}_i \leftarrow \mathcal{MC}_i'$
13 :            Repeat line3-6
14 :     $h \leftarrow H(pk_i\|S)$
15 :     $\pi \leftarrow$ SeqPoW$_{\mathsf{VDF}}$.Prove($pp, sk, i, B^-.h, S$)
16 :     $B \leftarrow (B^-.h, h, i, S, pk_i, \pi)$
17 :     New random output $B.rand \leftarrow H'(pk_i\|S)$
18 :     Append $B$ to $\mathcal{MC}_i$ after $B^-$
19 :     Propagate $B$
```

Fig. 6: Construction of RANDCHAIN.

of mining. Node p_i 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_i constructs a block B consisting of a random output, and appends B to $\mathcal{MC}_i$.

Similar to PoW-based blockchains, RANDCHAIN's blockchain may have forks, and it's possible that B is committed then reverted by a longer chain. By applying a high difficulty parameter T, the average time of mining a new block can be set much longer than network delay. This gives nodes sufficient time to propagate new blocks, and thus the fork rate can be reduced. Meanwhile, by Nakamoto consensus, if the majority of nodes honestly mine the longest chain and a sufficient number of valid blocks are appended after B, then B cannot be reverted, i.e., is *stable*, except for negligible probability [35]. We analyse RANDCHAIN's consistency guarantee in §5, and discuss how to achieve *finality*, i.e., make block irreversible in §6-A.

5. SECURITY AND EFFICIENCY OF RANDCHAIN

In this section, we analyse the security and efficiency of RANDCHAIN. We formalise the notion of Decentralised Randomness Beacon (DRB), and prove RANDCHAIN implements a DRB. In addition, we show that although constructed from PoW variants and Nakamoto consensus, RANDCHAIN is energy efficient and decentralised.

A. Security model

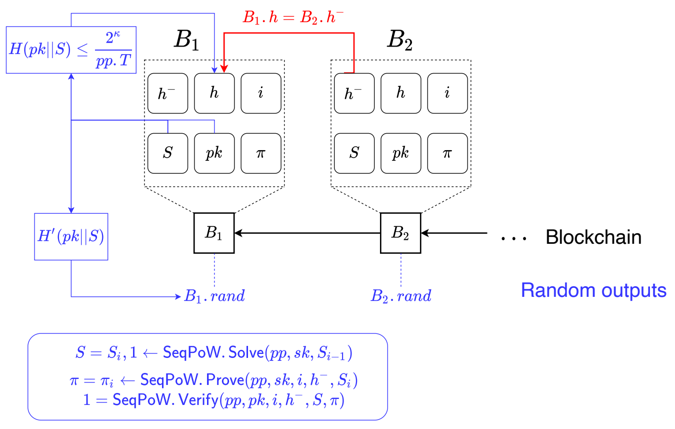
We consider the network is *synchronous*: all messages are delivered within a known time bound Δ. We do not assume rounds or *lock-step synchrony* [35]. The network consists of n nodes, each of which controls the same amount of mining power. We consider an adaptive Byzantine adversary who can corrupt 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.

B. Defining Decentralised Randomness Beacon

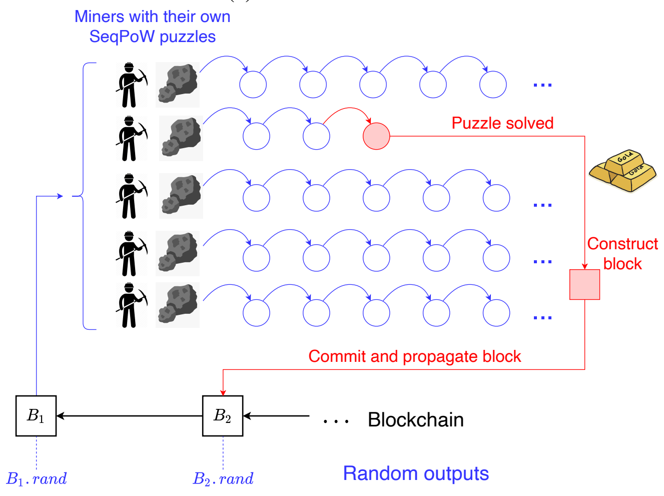
We start from defining security properties of DRB. First, similar to Nakamoto consensus, DRB should satisfy *consistency* and *liveness*. *DRB-Consistency* requires nodes to have consistent view on the blockchain. Without *DRB-Consistency*, nodes may revert random outputs arbitrarily.

Definition 11 (DRB-Consistency)**.** Parametrised by $k \in \mathbb{N}_{\geq 0}$. We say a DRB satisfies k-DRB-Consistency if for any two correct nodes at any time, their chains can differ only in the last k blocks, except for negligible probability.

The parameter k defines the degree of consistency guarantee. Some DRB-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-DRB-


$B_1.h = B_2.h^-$
B_1
B_2
$H(pk||S) \le \frac{2^\kappa}{pp.T}$
h^- h i
S pk π
$H'(pk||S)$
Blockchain
$B_1.rand$
$B_2.rand$
Random outputs
$S = S_i, 1 \leftarrow$ SeqPoW.Solve(pp, sk, S_{i-1})
$\pi = \pi_i \leftarrow$ SeqPoW.Prove(pp, sk, i, h^-, S_i)
$1 =$ SeqPoW.Verify(pp, pk, i, h^-, S, π)

(a) Beacon structure.



(b) Process of mining.

Fig. 7: Illustration of RANDCHAIN beacon.

Consistency. In §6 we discuss two approaches for adding *finality* to RANDCHAIN.

DRB-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* [5, 36, 37] or *Guaranteed Output Delivery (G.O.D.)* [38, 39, 40, 41] 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 (DRB-Liveness)**.** Parametrised by $t, \tau \in \mathbb{R}^+$. We say a DRB satisfies (t, τ)-DRB-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 (DRB-Uniform-Distribution)**.** 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. In some papers [42, 43, 40, 41], unpredictability in the first scenario is refereed as *bias-resistance*. We follow the *IND1-secrecy* (Indistinguishability of secrets) notion in SCRAPE [38] to define *DRB-Unpredictability*. IND1-secrecy requires that each node cannot learn anything about the final output before finishing the protocol.

Definition 14 (DRB-Unpredictability)**.** We say a DRB satisfies DRB-Unpredictability if no adversary can obtain non-negligible advantage on the following game. Assume all nodes are correct and all messages are delivered instantly. Given an agreed blockchain, the adversary makes a guess on the output of the next block. The advantage is quantified as the probability that the adversary makes an accurate guess on the output.

As RANDCHAIN is built upon Nakamoto consensus, RANDCHAIN 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, *selfish mining* [44] – which prevents Nakamoto consensus from achieving ideal *chain quality* – does not affect DRB's security guarantee. In addition, RANDCHAIN does not consider incentive, so is secure against all attacks on Nakamoto consensus' incentive mechanisms [45, 46, 47, 48].

C. Proofs of Consistency, Liveness and Uniform Distribution

As each random output is produced by a hash function, RANDCHAIN satisfies *DRB-Uniform-distribution*.

Lemma 2 (DRB-Uniform-distribution)**.** RANDCHAIN *satisfies DRB-Uniform-distribution.*

Proof. In RANDCHAIN, 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. □

RANDCHAIN's *DRB-Consistency* and *DRB-Liveness* are guaranteed by Nakamoto consensus. There have been extensive works [35, 49, 50, 51, 52, 53, 54, 55] analysing and proving consistency and liveness guarantee of Nakamoto consensus. As RANDCHAIN works in the same system model as that of Ren [52], RANDCHAIN at least satisfies the *consistency* and *liveness* bound proved by Ren [52].

Lemma 3 (DRB-Consistency)**.** *Let $g = e^{-\alpha\Delta}$. If $g^2\alpha > (1-\delta)\beta$, then* RANDCHAIN *satisfies k-DRB-Consistency except for $e^{-\Omega(\delta^2 g^2 k)}$ probability.*

Lemma 4 (DRB-Liveness)**.** *Let $g = e^{-\alpha\Delta}$. If $g\alpha > (1+\delta)\beta$, then* RANDCHAIN *satisfies $(t, \frac{\delta}{6}g\alpha t - k - 1)$-DRB-Liveness except for $e^{-\Omega(\delta^2 g\alpha t)}$ probability.*

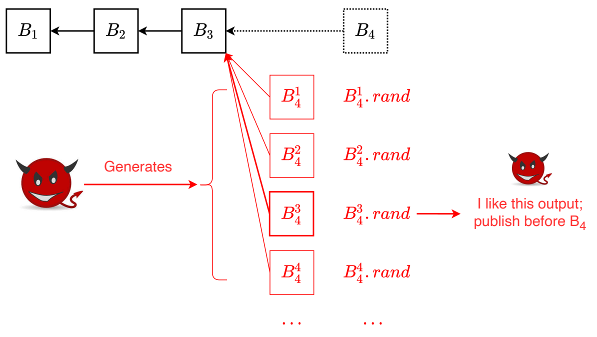


Fig. 8: Unpredictability game.

D. Proofs of Unpredictability

In the prediction game, the next output is either produced by correct nodes or the adversary's nodes. If the adversary's advantage is negligible for both cases, then RANDCHAIN satisfies *DRB-Unpredictability*. We prove the adversary's advantage for both cases. For the first case that the next output is produced by correct nodes, the adversary's best strategy is guessing, leading to negligible advantage.

Lemma 5. *Given a set of correct nodes and the latest valid ledger they agree on, if the next output is produced by a correct node, then the adversary's advantage on the prediction game is $\frac{1}{2^\kappa}$.*

We then consider the case that the next output is produced by the adversary's nodes. By the *SeqPoW-Sequentiality*, the adversary cannot predict each of its SeqPoW solution. Thus, 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. Recall that SeqPoW has two levels of hardness guarantee, namely SeqPoW-Hardness and SeqPoW-Uniqueness. We first analyse RANDCHAIN using SeqPoW with *SeqPoW-Hardness*, e.g., $\mathsf{SeqPoW_{VDF}}$.

Lemma 6. *Consider RANDCHAIN using SeqPoW with SeqPoW-Hardness. Given a set of correct nodes and the latest valid ledger they agree on, if the next output 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 SeqPoW-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 $\frac{k}{2^\kappa}$.

Thus, 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 using SeqPoW with *SeqPoW-Uniqueness*, e.g., $\mathsf{SeqPoW_{Sloth}}$.

Lemma 7. *Consider RANDCHAIN using SeqPoW with SeqPoW-Uniqueness. Given a set of correct nodes and the latest valid ledger they agree on, if the next output 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 SeqPoW-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

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

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 SeqPoW-Uniqueness is always smaller than in RANDCHAIN without SeqPoW-Uniqueness. 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,

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

E. 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.

Mining in PoW-based consensus is parallelisable. Parallelisable mining leads to mining marketisation, which further leads to centralisation and huge energy consumption. When mining is parallelisable, miners can employ specialised mining hardware such as GPUs, FPGAs and ASICs to maximise their mining reward. Due to the mining reward, blockchain mining has been marketised: people manufacture, invest and trade high-end mining hardware for higher profit. Eventually, most blocks are mined by several miners with massive high-end mining hardware, while few blocks are mined by normal miners. Mining power centralisation weakens PoW-based consensus' security, as powerful miners can perform various attacks, e.g., 51% attacks [12] to break PoW-based consensus. In addition, mining – which enforces hardware to fully operate all the time – is energy inefficient. As miners deploy powerful mining hardware, mining has cost a great amount of electricity. For example, in 2019, Bitcoin costs 58.93 KWh energy – more than Switzerland which costs 58.46 KWh [56].

RandChain is more energy efficient and decentralised compared to PoW-based consensus. In RandChain, miners cannot parallelise mining: each miner can only use a single processor to mine. Compared to massively parallel PoW mining, a single fully operating processor costs negligible electricity. In addition, the performance gap on $\mathsf{SeqPoW.Solve}(\cdot)$ between high-end and low-end hardware is small. Given the sequentiality, $\mathsf{SeqPoW.Solve}(\cdot)$ 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 [57]. This means one can only speed up SeqPoW mining for $4 \sim 5$ times. In addition, given the voltage limit of processors, high clock rate is hard to improve further [13]. Thus, powerful nodes cannot obtain much speedup on mining, leading to high degree of mining power decentralisation.

6. Practical considerations

A. Adding finality

Finality [17] is a property for Byzantine consensus that, previously agreed proposals cannot be reverted. RandChain does not satisfy *finality* due to its Nakamoto consensus nature. In some scenarios, we require a DRB to satisfy finality. For example, consider leader election 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 end up with no leader. RANDAO [7] and Proofs-of-Delay [10] bypass this by replacing $H'(\cdot)$ with a VDF which takes time more than generating k blocks to execute, where k is the degree of k-*DRB-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.

We consider principled approaches for adding *finality*. Adding *finality* is equivalent to achieve 0-*DRB-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* [58] to agree on a block. RandChain satisfies *validity* and *termination*, but does not satisfy 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 [59] 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 nodes agree 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 [59, 60] 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 collection of $n-f$ 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 $2f+1$ nodes are correct. 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., $\geq 2f+1$ signatures on this block.

If the system allows temporary non-finalised blocks for better liveness, then RandChain can still keep Nakamoto consensus: even without quorum certificate, a block is considered finalised with a sufficiently long sequence of succeeding blocks. This resembles the Streamlet blockchain [61]. If not, then nodes should obtain a quorum certificate for every block before proposing succeeding blocks.

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 [62, 63] 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 [63] as a finality layer for Bitcoin Cash [64].

B. Making SeqPoW mining non-outsourceable

RandChain does not prevent *outsourcing*: the adversary can solve others' SeqPoW puzzles. An adversary with massive processors and others' public keys can mine for all nodes. If the adversary's processors are with high clock rate, then it can always learn randomness earlier than others, or even publish its preferred random outputs only.

To avoid this, we can adopt the idea of *VRF-based mining* [65]. 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 [66]. 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 *SeqPoW-Uniqueness* are considered as future work.

C. 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 [67] 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, same as in 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 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 DRB, 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 does not rely on strong assumptions such as trusted third party and lock-step synchrony, while being secure, scalable, energy efficient and decentralised.

DRBs from external randomness source. Some DRBs that derive randomness from real-world random entropy, including financial data, e.g., stock prices [8] and public blockchains [9, 6, 10] 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* [68], where the random output is only a binary bit. DRG can be constructed from various cryptographic primitives, such as commitments [69, 7], threshold signatures [36, 3, 37], VRFs [70, 71, 2, 5, 72], secret sharing [1, 42, 38, 43, 39] and homomorphic encryption [40].

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. Previous research [73] shows that constructing leader election is challenging. Boneh et al. [74] propose two leader election constructions, which however rely on a RB in return. Second, if the network is not synchronous, then the DRG-based DRB should rely on a *pacemaker* [75] for liveness. without synchrony, a node cannot know whether other nodes receive its message, and nodes may not agree on which round they reach. When this happens, nodes may stop working forever. The pacemaker is responsible to inform nodes when to start a new round. 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 [76, 43] 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 an iteratively sequential function $f(\cdot)$. 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 [15] and Ephraim et. al. [77] construct DRBs from Sloth and Continuous VDFs, respectively.

The biggest challenge of this paradigm is *frontrunning*. If the random seed is previously known by someone, then it can pre-compute and learn random outputs earlier than others. This can be solved by a trusted setup, e.g., the Sapling ceremony for Zcash [78]. In addition, nodes with faster processors learn random outputs earlier than others.

RandRunner [41] extends this paradigm by allowing nodes to execute iterations of $f(\cdot)$ in turn. In each round, nodes elect a leader with the last random output as input. With the last output as input, the leader runs the *trapdoor* VDF while others run the normal VDF to obtain the next random output. Due to the trapdoor, the leader learns the next random output earlier than others. If the leader is malicious, other nodes can still learn the next random output, but slower. This does not prevent frontrunning: the leader always learns randomness earlier than others. 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 introduce Sequential Proof-of-Work (SeqPoW), a variant of Proof-of-Work that is sequential, i.e., non-parallelisable. For SeqPoW, we provide concrete constructions of SeqPoW, and show that SeqPoW is practical and useful for various cryptographic protocols. For RandChain, we show that while inheriting simplicity, security, and scalability from PoW-based consensus, RandChain remains energy efficient and decentralised.

Compared to existing DRBs, RandChain explores a different approach with several unique design choices. In the future, we will explore these design choices further and propose more secure and scalable DRBs based on them.

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, which takes the genesis block as random seed and is run by multiple nodes in turn. 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 can be no longer realistic. This makes RandChain even more secure than DRBs based on iteratively sequential functions.

References

[1] Aggelos Kiayias et al. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[2] Bernardo David et al. "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] Timo Hanke, Mahnush Movahedi, and Dominic Williams. "Dfinity technology overview series, consensus system". In: *arXiv preprint arXiv:1805.04548* (2018).

[4] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[5] Yossi Gilad et al. "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. 2017, pp. 51–68.

[6] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. "On Bitcoin as a public randomness source." In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 1015.

[7] *RANDAO: A DAO working as RNG of Ethereum*. last accessed on 02/08/2020. URL: https://github.com/randao/randao.

[8] Jeremy Clark and Urs Hengartner. "On the Use of Financial Data as a Random Beacon." In: *EVT/WOTE* 89 (2010).

[9] Marcin Andrychowicz and Stefan Dziembowski. "Distributed Cryptography Based on the Proofs of Work." In: *IACR Cryptol. ePrint Arch.* 2014 (2014), p. 796.

[10] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. "Proofs-of-delay and randomness beacons in ethereum". In: *IEEE Security and Privacy on the blockchain (IEEE S&B)* (2017).

[11] Markus Jakobsson and Ari Juels. "Proofs of work and bread pudding protocols". In: *Secure information networks*. Springer, 1999, pp. 258–272.

[12] Satoshi Nakamoto et al. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).

[13] "Why has CPU frequency ceased to grow?" In: (). last accessed on 01/08/20. URL: https://software.intel.com/content/www/us/en/develop/blogs/why-has-cpu-frequency-ceased-to-grow.html.

[14] Dan Boneh et al. "Verifiable delay functions". In: *Annual international cryptology conference*. Springer. 2018, pp. 757–788.

[15] Arjen K Lenstra and Benjamin Wesolowski. “A random zoo: sloth, unicorn, and trx.” In: *IACR Cryptol. ePrint Arch.* 2015 (2015), p. 366.

[16] Krzysztof Pietrzak. “Simple verifiable delay functions”. In: *10th innovations in theoretical computer science conference (itcs 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[17] Vitalik Buterin and Virgil Griffith. “Casper the friendly finality gadget”. In: *arXiv preprint arXiv:1710.09437* (2017).

[18] Cynthia Dwork and Moni Naor. “Pricing via processing or combatting junk mail”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 139–147.

[19] Benjamin Wesolowski. “Efficient verifiable delay functions”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 379–407.

[20] Ronald L Rivest, Adi Shamir, and David A Wagner. “Time-lock puzzles and timed-release crypto”. In: (1996).

[21] Mohammad Mahmoody, Tal Moran, and Salil Vadhan. “Publicly verifiable proofs of sequential work”. In: *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*. 2013, pp. 373–388.

[22] Nico Döttling et al. “Tight Verifiable Delay Functions.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 659.

[23] Sunny King and Scott Nadal. “Ppcoin: Peer-to-peer crypto-currency with proof-of-stake”. In: *self-published paper, August* 19 (2012), p. 1.

[24] *Nakamoto Consensus with VDF and VRF*. last accessed on 01/08/20. URL: https://ethresear.ch/t/nakamoto-consensus-with-vdf-and-vrf/5671.

[25] *PoS based on Synthetic PoW using VDF and VRF*. last accessed on 01/08/20. URL: https://ethresear.ch/t/pos-based-on-synthetic-pow-using-vdf-and-vrf/7271.

[26] Jieyi Long and Ribao Wei. “Nakamoto Consensus with Verifiable Delay Puzzle”. In: *arXiv preprint arXiv:1908.06394* (2019).

[27] Hamza Abusalah et al. “Reversible proofs of sequential work”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 277–291.

[28] Paul Valiant. “Incrementally verifiable computation or proofs of knowledge imply time/space efficiency”. In: *Theory of Cryptography Conference*. Springer. 2008, pp. 1–18.

[29] Moni Naor, Omer Paneth, and Guy N Rothblum. “Incrementally verifiable computation via incremental PCPs”. In: *Theory of Cryptography Conference*. Springer. 2019, pp. 552–576.

[30] Nir Bitansky et al. “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.

[31] Sean Bowe, Jack Grigg, and Daira Hopwood. “Halo: Recursive Proof Composition without a Trusted Setup.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1021.

[32] Benedikt Bünz et al. “Proof-Carrying Data from Accumulation Schemes.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 499.

[33] Dan Boneh, Benedikt Bünz, and Ben Fisch. “A Survey of Two Verifiable Delay Functions.” In: *IACR Cryptol. ePrint Arch.* 2018 (2018), p. 712.

[34] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. “Implementation Study of Two Verifiable DelayFunctions.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 332.

[35] Juan Garay, Aggelos Kiayias, and Nikos 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.

[36] Christian Cachin, Klaus Kursawe, and Victor Shoup. “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography”. In: *Journal of Cryptology* 18.3 (2005), pp. 219–246.

[37] Eleftherios Kokoris-Kogias et al. “Bootstrapping Consensus Without Trusted Setup: Fully Asynchronous Distributed Key Generation.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1015.

[38] Ignacio Cascudo and Bernardo David. “SCRAPE: Scalable randomness attested by public entities”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 537–556.

[39] Ignacio Cascudo and Bernardo David. “ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing”. In: ().

[40] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. “Homomorphic Encryption Random Beacon.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 1320.

[41] Philipp Schindler et al. *RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness*. Cryptology ePrint Archive, Report 2020/942. https://eprint.iacr.org/2020/942. 2020.

[42] Ewa Syta et al. “Scalable bias-resistant distributed randomness”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. Ieee. 2017, pp. 444–460.

[43] Philipp Schindler et al. “HydRand: Efficient Continuous Distributed Randomness”. In: *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 32–48.

[44] Ittay Eyal and Emin Gün Sirer. “Majority is not enough: Bitcoin mining is vulnerable”. In: *International conference on financial cryptography and data security*. Springer. 2014, pp. 436–454.

[45] Joseph Bonneau et al. “Why buy when you can rent? Bribery attacks on Bitcoin consensus”. In: (2016).

[46] Kevin Liao and Jonathan Katz. “Incentivizing blockchain forks via whale transactions”. In: *International Conference on Financial Cryptography and Data Security*. Springer. 2017, pp. 264–279.

[47] Aljosha Judmayer et al. “Pay-To-Win: Incentive Attacks on Proof-of-Work Cryptocurrencies.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 775.

[48] Runchao Han et al. “Fact and Fiction: Challenging the honest majority assumption of permissionless blockchains”. In: ().

[49] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. “The bitcoin backbone protocol with chains of variable difficulty”. In: *Annual International Cryptology Conference*. Springer. 2017, pp. 291–323.

[50] Rafael Pass, Lior Seeman, and Abhi 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.

[51] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi 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.

[52] Ling Ren. “Analysis of Nakamoto Consensus.” In: *IACR Cryptol. ePrint Arch.* 2019 (2019), p. 943.

[53] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. “Full Analysis of Nakamoto Consensus in Bounded-Delay Networks.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 277.

[54] Amir Dembo et al. “Everything is a Race and Nakamoto Always Wins”. In: *arXiv preprint arXiv:2005.10484* (2020).

[55] Peter Gazi, Aggelos Kiayias, and Alexander Russell. “Tight Consistency Bounds for Bitcoin”. In: (2020).

[56] “Bitcoin’s energy consumption ’equals that of Switzerland’”. In: (). last accessed on 01/08/20. URL: 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..

[57] *AMD Breaks 8GHz Overclock with Upcoming FX Processor, Sets World Record*. last accessed on 01/08/20. URL: http://hothardware.com/News/AMD-Breaks-Frequency-Record-with-Upcoming-FX-Processor/.

[58] Christian Cachin, Rachid Guerraoui, and Luís Rodrigues. *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[59] Dahlia Malkhi and Michael Reiter. “Byzantine quorum systems”. In: *Distributed computing* 11.4 (1998), pp. 203–213.

[60] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony”. In: *Journal of the ACM (JACM)* 35.2 (1988), pp. 288–323.

[61] Benjamin Y Chan and Elaine Shi. “Streamlet: Textbook Streamlined Blockchains.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 88.

[62] T-H Hubert Chan, Rafael Pass, and Elaine Shi. “Consensus through herding”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2019, pp. 720–749.

[63] Team Rocket et al. “Scalable and probabilistic leaderless bft consensus through metastability”. In: *arXiv preprint arXiv:1906.08936* (2019).

[64] “BCH Avalanche Transactions Show Finality Speeds 10x Faster Than Ethereum”. In: (). last accessed on 01/08/20. URL: https://news.bitcoin.com/bch-avalanche-transactions-show-finality-speeds-10x-faster-than-ethereum/.

[65] “VRF-Based Mining: Simple Non-Outsourceable Cryptocurrency Mining”. In: (). last accessed on 01/08/20. URL: https://github.com/DEX-ware/vrf-mining/blob/master/paper/main.pdf.

[66] Silvio Micali, Michael Rabin, and Salil Vadhan. “Verifiable random functions”. In: *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE. 1999, pp. 120–130.

[67] Daniel Stutzbach and Reza Rejaie. “Understanding churn in peer-to-peer networks”. In: *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. ACM. 2006, pp. 189–202.

[68] Manuel Blum. “Coin flipping by telephone a protocol for solving impossible problems”. In: *ACM SIGACT News* 15.1 (1983), pp. 23–27.

[69] Baruch Awerbuch and Christian Scheideler. “Robust random number generation for peer-to-peer systems”. In: *International Conference On Principles Of Distributed Systems*. Springer. 2006, pp. 275–289.

[70] Yevgeniy Dodis. “Efficient construction of (distributed) verifiable random functions”. In: *International Workshop on Public Key Cryptography*. Springer. 2003, pp. 1–17.

[71] Veronika Kuchta and Mark Manulis. “Unique aggregate signatures with applications to distributed verifiable random functions”. In: *International Conference on Cryptology and Network Security*. Springer. 2013, pp. 251–270.

[72] David Galindo et al. “Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 96.

[73] Carlos Gómez-Calzado et al. “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.

[74] Dan Boneh et al. “Single Secret Leader Election.” In: *IACR Cryptol. ePrint Arch.* 2020 (2020), p. 25.

[75] Maofan Yin et al. “Hotstuff: Bft consensus with linearity and responsiveness”. In: *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 2019, pp. 347–356.

[76] Gang Wang et al. “Sok: Sharding on blockchain”. In: *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 2019, pp. 41–61.

[77] Naomi Ephraim et al. “Continuous verifiable delay functions”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2020, pp. 125–154.

[78] Daira Hopwood et al. “Zcash protocol specification”. In: *GitHub: San Francisco, CA, USA* (2016).

Appendix

A. Security proof of two SeqPoW constructions

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

Lemma 8. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies SeqPoW-Completeness.*

Proof. Assuming there is an (λ, ψ, T)-valid tuple $(pp, sk, i, x, S_i, \pi_i)$. By *VDF-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 9. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies SeqPoW-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 *VDF-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 *VDF-Soundness*. Thus, this assumption contradicts *VDF-Soundness*. □

Lemma 10. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies SeqPoW-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 *VDF-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 11. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *does not satisfy SeqPoW-Uniqueness.*

Proof. By *SeqPoW-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 12. *If the underlying VDF satisfies σ-VDF-Sequentiality, then* $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies σ-SeqPoW-Sequentiality.*

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

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

By *σ-VDF-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 SeqPoW-Uniqueness below.

Lemma 13. $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ *satisfies SeqPoW-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. □