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

Runchao Han
Monash University
& CSIRO-Data61
runchao.han@monash.edu

Haoyu Lin
Bytom Foundation
& ZenGo X
chris.haoyul@gmail.com

Jiangshan Yu*
Monash University
jiangshan.yu@monash.edu

Abstract

This paper presents RandChain, a new family of decentralised randomness beacon (DRB) protocol. Unlike existing DRB protocols where nodes are *collaborative*, i.e., contributing local entropy and aggregating them to a single output, nodes in RandChain are *competitive*: nodes compete with each other on becoming the next leader, who solely prepares and proposes the next random output. RandChain employs *Sequential Proof-of-Work (SeqPoW)*, a Proof-of-Work (PoW) puzzle that cannot be solved faster by using multiple processors in parallel. To propose a random output, a node needs to solve a SeqPoW puzzle derived from the last random output and the node's identity. Nodes execute Nakamoto consensus to agree on a unique vector of random outputs.

RandChain achieves promising security and performance without relying on strong assumptions. With Nakamoto consensus, RandChain does not rely on trustworthy leaders or lock-step synchrony, and achieves linear communication complexity. Biasing or predicting future random outputs is infeasible, as nodes cannot choose their own SeqPoW inputs and SeqPoW solutions are unpredictable. Also, RandChain, for the first time, achieves *non-parallelisable mining*, where each node can only use a single processor to mine.

1 Introduction

Randomness is a key building block for various protocols and applications. Decentralised Randomness Beacon (DRB) allows a group of participants (aka nodes) to jointly generate random outputs. DRB protocols have been deployed by well-known organisations [8] and integrated into various applications, especially blockchains [31, 54, 68, 72].

Most DRB protocols are constructed from periodically executing a Distributed Randomness Generation (DRG) protocol, where a group of nodes contribute their local entropy and aggregate them into a single random output. DRG protocols can be constructed from various cryptographic primitives, including threshold cryptosystems [41, 49, 68], Verifiable Random Functions (VRFs) [54, 64, 67], and Publicly Verifiable Secret Sharing (PVSS) [42, 43, 72, 74, 93, 95].

Limitations of DRG-based DRBs. DRG-based DRBs suffer from two limitations, namely *round synchronisation* and high communication overhead. First, nodes execute DRG in rounds, so have to ensure they are executing the same round of the protocol. If nodes disagree on the round numbers, then the DRB may lose safety and/or liveness forever [82]. In addition, to agree on random outputs, nodes have to make all-to-all broadcasts, leading to communication complexity of at least $O(n^2)$ [41, 42, 72, 74, 93].

The crux: nodes are collaborative. We attribute the two limitations to the design that nodes are *collaborative*: nodes contribute their local inputs and aggregate them into a single output. The collaborative process ensures that no node can fully control random outputs, making them hard to bias or predict. However, in order to collaborate, nodes should continuously broadcast messages to and synchronise with each other. The former incurs at least quadratic communication complexity, and the latter introduces the round synchronisation problem. All extra designs incorporated with DRG – e.g., using leaders [41–43, 49, 54, 64, 67, 68, 72, 95], sharding [68, 95], cryptographic sortition [67], Byzantine consensus [67, 93], and erasure coding [42, 43] – aim at reducing the impact of the above two limitations. However, since all of them are in the collaborative design, they inherently suffer from the two limitations and cannot address them completely.

Blockchains: collaborative v.s. competitive. DRB can be seen as a blockchain protocol (aka State Machine Replication), the basic paradigm for designing decentralised systems such as distributed ledgers [25, 30, 75, 80, 99]. A blockchain protocol organises system states as a *blockchain*, i.e., a chain of blocks, and nodes continuously append blocks attaching new system states to the blockchain. Blockchain protocols can be constructed from two approaches, namely the traditional Byzantine Fault Tolerant (BFT)-style consensus [44] and Nakamoto-style consensus [80]. Similar to DRG-based DRBs, nodes in

*Corresponding author.

BFT-style consensus are *collaborative*: they vote to decide the next block. On the contrary, nodes in Nakamoto consensus are *competitive* with each other: they compete on becoming the next leader, who solely prepares and proposes the next block.

The competitive design addresses several limitations of blockchain protocols with BFT-style consensus. The leader election eliminates the need of leaders and lock-step synchrony. As the leader takes full charge of proposing the next block, the only communication is the leader broadcasting its block, leading to linear communication complexity. Each node keeps extending its local blockchain, so does not need to always synchronise its round number with other nodes. Table 1 classifies existing blockchain protocols. It shows that there exists a gap of knowledge on designing and understanding competitive DRBs.

Table 1: Taxonomy of blockchain protocols.

		Relationship of nodes	
		Collaborative	**Competitive**
Applications	**Distributed Ledger**	Tendermint [75] HL. Fabric [25] Libra [30]	Bitcoin [80] Ethereum [99]
	Decentralised Randomness Beacon	DRG-based DRBs [41–43, 49, 54, 64, 67, 68, 72, 74, 93, 95]	RANDCHAIN **(This work)**

Our proposal: RANDCHAIN**.** To fill this gap, we propose RANDCHAIN, a new family of DRBs where nodes are *competitive*. In RANDCHAIN, nodes run a leader election protocol, and the elected leader derives the next random output based on data generated during the leader election. As 1) nodes cannot predict who will become the next leader, 2) the leader cannot predict random outputs produced by itself, and 3) before being replaced by a new leader, the leader can only produce a limited number of random outputs (following the Poisson distribution like in Nakamoto consensus [66, 80, 91]), random outputs remain unpredictable and unbiasible. Nodes execute Nakamoto consensus to agree on a unique vector of random outputs.

RANDCHAIN can employ any leader election protocol compatible with Nakamoto consensus. We construct RANDCHAIN from Proof-of-Work (PoW)-based leader election: each node keeps solving PoW puzzles (aka mining), and the node who first solves the puzzle becomes the leader. The leader deterministically derives the next random output from its PoW solution.

The mining process is usually centralised and energy-inefficient. A promising solution to keep mining decentralised and energy-efficient is *non-parallelisable mining* [2]: each node can only mine by using a single processor. RANDCHAIN, for the first time, achieves *non-parallelisable* mining in permissioned networks where neach node has a unique identity, by using *Sequential Proof-of-Work* (SeqPoW). SeqPoW is a PoW variant that is *sequential*, i.e., cannot be solved faster by using multiple processors in parallel. Unlike existing time-sensitive cryptographic primitives such as Proof of Sequential Work (PoSW) [51, 78] and Verifiable Delay Functions (VDFs) [34, 87, 98], SeqPoW takes a random and unpredictable (rather than fixed) number of steps, with a mathematical expectation parametrised by a difficulty parameter. This makes SeqPoW useful for constructing other protocols such as leader election and Proof-of-Stake (PoS)-based consensus, while being a building block of RANDCHAIN.

Our contributions are summarised as follows.

- We introduce a new notion and identity new design space for DRBs. Differ from all the existing concepts and DRBs, in our design nodes are competitive rather than collaborative.
- We introduce the concept of SeqPoW, including formalisation, two constructions based on VDFs [87, 98] and Sloth [76], and security and efficiency analysis (§2).
- We provide RANDCHAIN (§3) as a concrete instantiation of DRB in the identified design space, where nodes are competitive. We provide an analysis on its security and efficiency (§4).
- We provide an implementation of SeqPoW and RANDCHAIN, and evaluate their performance (§5). For SeqPoW, we show that SeqPoW with Pietrzak's VDF [87] is most suitable for RANDCHAIN, given its trade-off between computing/verifying proofs and the proof size. For RANDCHAIN, we show that in a cluster with up to *1024 nodes*, a random output can be propagated to the majority of nodes *within 1.3 seconds*; nodes have comparable chance of producing random outputs; and the average bandwidth utilisation is *less than 200KB/s* even when the interval between two random outputs is only one second.
- We establish a unified evaluation framework of DRBs, and compare RANDCHAIN with existing DRBs under this framework (§6). Our comparison results show that RANDCHAIN is the only DRB that is leaderless, secure and efficient simultaneously, without relying on lock-step synchrony or trusted parties.

We conclude the paper in §7, and provide additional details in the appendix. In particular, Appendix A provides preliminary formal definitions; Appendix B-C provides security proofs for SeqPoW and RANDCHAIN; and Appendix D discusses some practical considerations.

2 Sequential Proof-of-Work

Sequential Proof-of-Work (SeqPoW) is the key building block of RANDCHAIN. We formalise SeqPoW, provide two constructions, and formally analyse their security and efficiency.

2.1 Preliminaries

Notations. Table 2 summarises notations in this paper.
Proof-of-Work (PoW) [59] is a computationally hard puzzle that is hard to solve but efficient to verify. PoW can be constructed from hash functions: the prover searches for a nonce

Table 2: Summary of notations.

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 from $\{0,1\}^*$ 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
$\{p_1,\dots,p_n\}$	Nodes in the network
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
α,β	Fraction of faulty and correct nodes ($\alpha+\beta=1$)

x such that $H(in\|x) \leq \frac{2^\kappa}{T}$, where $H:\{0,1\}^* \to \{0,1\}^\kappa$ is a cryptographic hash function, in is the input, 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.

Verifiable Delay Function (VDF) [34, 87, 98] allows a prover to evaluate an input, and produce a unique output with a succinct proof. The evaluation process takes non-negligible and parameterisable time to execute, even with parallelism. With the input, output and proof, anyone can verify whether the output is generated from the input. Specifically, a 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.

VDF satisfies three properties:

- *completeness*: all outputs from honest evaluations can pass the verification;
- *soundness*: all outputs from malicious evaluations cannot pass the verification; and
- σ-*sequentiality*: $\mathsf{Eval}(\cdot,\cdot,t)$ cannot be evaluated within parallel time $\sigma(t)$.

VDFs are usually constructed from an iteratively sequential function (ISF) and a succinct proof attesting the ISF's execution results [87, 98]. An ISF $f(t,x) = g^t(x)$ is implemented by composing $g(x)$ for t times, where $g(\cdot)$ is a sequential function that cannot be solved faster by using multiple processors in parallel. Given $g(\cdot)$'s sequentiality, the fastest way of computing $f(t,x)$ is to iterate $g(x)$ for t times. In addition, $f(\cdot)$ is *self-composable*: for any x and (t_1,t_2), let $y \leftarrow f(x,t_1)$, we have $f(x,t_1+t_2) = f(y,t_2)$. VDFs usually inherit the *self-composability* from ISFs, i.e., for all λ, t_1, t_2, let $pp \leftarrow \mathsf{Setup}(\lambda)$ and $y \leftarrow \mathsf{Eval}(pp,x,t_1)$, it holds that $\mathsf{Eval}(pp,x,t_1+t_2) = \mathsf{Eval}(pp,y,t_2)$. Such VDFs are known as *self-composable VDFs* [57]. Appendix A provides formal definitions of VDF.

2.2 Basic idea

SeqPoW is a PoW that cannot be solved faster by using multiple processors in parallel. As shown in Figure 1, given an initial SeqPoW puzzle S_0, the prover keeps solving it by incrementing an ISF. Each iteration takes the last output S_{i-1} as input and produces a new output S_i. For each output S_i, the prover checks whether it satisfies a difficulty parameter T. If yes, then S_i is a valid solution, and the prover can generate a proof π_i on it. Given S_i and π_i, the verifier can check S_i's correctness without solving the puzzle again.

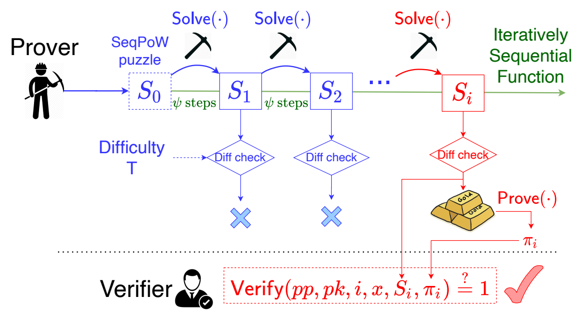


Figure 1: Sequential Proof-of-Work.

Comparisons with relevant primitives (Table 3). SeqPoW differs from VDFs and other time-sensitive cryptographic primitives, e.g., Timelock Puzzle (TLP) [89] and Proofs of Sequential Work (PoSW) [51, 78], that, the SeqPoW prover iterates an ISF for a *randomised* (rather than given) number of times. In addition, compared to TLP where outputs are not publicly verifiable, SeqPoW outputs are publicly verifiable. Compared to existing PoSW constructions [51, 78] where proofs are not unique, $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ provides unique outputs. SeqPoW differs from PoW that SeqPoW is sequential. SeqPoW differs from *memory-hard functions* (MHFs) [24, 32, 86] that, SeqPoW is bottlenecked by the processor's frequency, whereas MHF is bottlenecked by the memory bandwidth.

Two concurrent works [55, 77] suggest ways of randomising the number of iterations of VDFs. While they use ideas similar to SeqPoW, we are the first to formally study such primitives, including formal definitions, concrete constructions with security proofs, implementation and evaluation. We also provide SeqPoW with *uniqueness* that cannot be achieved in their constructions.

Table 3: SeqPoW v.s. relevant primitives.

Primitive		Execution			Output	
		Sequential	#Steps	Bottleneck	Unique	Verifiable
Time-sensitive	TLP	✓	Fixed	Proc. freq.	✓	✗
	PoSW	✓	Fixed	Proc. freq.	✗	✓
	VDF	✓	Fixed	Proc. freq.	✓	✓
Resource-consuming	MHF	✓or ✗	Fixed	Mem. bandw.	✓	✓
	PoW	✗	Random	Proc. freq. + # of procs.	✗	✓
Our work	$\mathsf{SeqPoW}_{\mathsf{VDF}}$	✓	Random	Proc. freq.	✗	✓
	$\mathsf{SeqPoW}_{\mathsf{Sloth}}$	✓	Random	Proc. freq.	✓	✓

Applications. The *hardness* property implies that, the prover cannot predict in which step it will solve the SeqPoW puzzle. This makes SeqPoW useful in various protocols, such as leader election and Proof-of-Stake (PoS)-based consensus.

1) **Leader election.** Mining in PoW-based consensus 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. SeqPoW can be a drop-in replacement of PoW for the leader election purpose. Later in §4, we will show that compared to parallelisable PoW, SeqPoW-based leader election can be fairer and more energy-efficient.
2) **PoS-based consensus.** In Proof-of-Stake (PoS)-based consensus [73], each node's chance of mining a block is in proportion to its *stake*, e.g, the node's balance. Most PoS-based consensus protocols [28, 53, 54, 63, 72] select block proposers in a *predictable* [29] way: for every round, a node can predict whether it will be the block proposer before mining the block. Predictable PoS-based consensus is vulnerable to a class of prediction-based attacks, and therefore tolerates less Byzantine mining power [29] than PoW-based consensus. To make PoS-based consensus unpredictable, one can randomise the process of selecting block proposers. SeqPoW can provide such functionality: each node solves a SeqPoW with its identity, the last block, and the difficulty parameter inversely proportional to its stake as input, and the first node solving its SeqPoW becomes the block proposer. Two concurrent and independent works [55, 77] provide concrete constructions following the similar idea.

2.3 Definition

We formally define SeqPoW as follows.

Definition 1 (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 $H : \mathcal{Y} \to \mathcal{X}$, where $\mathcal{X}$ is efficiently sampleable.

$\mathsf{Gen}(pp) \to (sk, pk)$: A probabilistic function, which 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_0. Some constructions may use public key pk as input rather than sk. This also applies to $\mathsf{Solve}(\cdot)$ and $\mathsf{Prove}(\cdot)$.

$\mathsf{Solve}(pp, sk, S_i) \to (S_{i+1}, b_{i+1})$: 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 public parameter 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 2 (Honest tuple)**.** A tuple $(pp, sk, i, x, S_i, \pi_i)$ is (λ, ψ, T)-honest if and only if for all $pp \leftarrow \mathsf{Setup}(\lambda, \psi, T)$, the following holds:

- $i = 0$ and $(S_0, \pi_0) \leftarrow \mathsf{Init}(pp, sk, x)$, and
- $\forall i \in \mathbb{N}^+$, $(S_i, b_i) \leftarrow \mathsf{Solve}(pp, sk, S_{i-1})$ 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 3 (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 4 (Completeness)**.** A SeqPoW scheme satisfies completeness if for all λ, ψ, T,

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

Definition 5 (Soundness)**.** A SeqPoW scheme satisfies soundness if for all λ, ψ, T,

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

We define *hardness* with difficulty T such that each attempt of $\mathsf{Solve}(\cdot)$ has the success rate of $\frac{1}{T}$.

Definition 6 (Hardness)**.** A SeqPoW scheme satisfies hardness if for all (λ,ψ,T)-honest tuple (pp,sk,i,x,S_i,π_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*: even with parallel processors, the fastest way of computing S_i is incrementing $\mathsf{Solve}(\cdot)$ for i times, which takes time $\sigma(i\cdot\psi)$. *Sequentiality* also captures the *unpredictability* that, the adversary cannot predict S_i's value before finishing computing S_i.

Definition 7 (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 $\sigma(i\cdot\psi)$ with at most $\mathsf{poly}(\lambda)$ processors,

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

We also 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 8 (Uniqueness)**.** A SeqPoW scheme satisfies uniqueness if for any two (λ,ψ,T)-valid tuples (pp,sk,i,x,S_i,π_i) and (pp,sk,i,x,S_j,π_j), $i=j$ holds.

2.4 Constructions

We propose two SeqPoW 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 an element of G. Let g be a generator of G. Let sk be the secret key, and $pk=g^{sk}$ be the public key.

SeqPoW from VDFs (Figure 2a). Let ψ be a step parameter, x be the input, and T be the difficulty parameter. The prover runs $\mathsf{Init}(\cdot)$, which generates the initial solution $S_0=H_G(pk\|x)$. Then, the prover keeps running $\mathsf{Solve}(\cdot)$, which calculates an intermediate output $S_i=\mathsf{VDF.Eval}^i(pp,S_0,\psi)$ and checks whether $H(pk\|S_i)\leq\frac{2^\kappa}{T}$. If true, then S_i is a valid solution, and the prover runs $\mathsf{Prove}(\cdot)$, which outputs proof π_i attesting $S_i=\mathsf{VDF.Eval}^i(pp,S_0,\psi)$. Note that $\mathsf{VDF.Eval}^i(pp,S_0,\psi)=\mathsf{VDF.Eval}(pp,S_0,i\cdot\psi)$ for *self-composable* VDFs. The verifier runs $\mathsf{Verify}(\cdot)$, which checks 1) whether $S_i=\mathsf{Eval}^i(pp,S_0,\psi)$ by running $\mathsf{VDF.Verify}(pp_{\mathsf{VDF}}, pk, i\cdot\psi, x, S_i, \pi_i)$, and 2) whether S_i satisfies the difficulty T.

Unique SeqPoW from Sloth (Figure 2b). $\mathsf{SeqPoW}_{\mathsf{VDF}}$ does not provide *uniqueness*: the prover can keep incrementing the ISF to find as many valid solutions as possible. We construct $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ with *uniqueness* from Sloth [76], a *slow-timed* hash function. In Sloth, the prover square roots (on a cyclic group G) the input t times to get the 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 is faster than computing: on cyclic group G, squaring is $O(\log|G|)$ times faster than square rooting. Similar to $\mathsf{SeqPoW}_{\mathsf{VDF}}$, $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ takes each of $S_i=f(i\cdot\psi,S_0)$ as an intermediate output and checks if $H(pk\|S_i)\leq\frac{2^\kappa}{T}$. To make the solution unique, $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ only treats the first solution satisfying the difficulty as valid. When verifying S_i, if the verifier finds an intermediate output S_j $(j<i)$ satisfying the difficulty, then S_i is considered invalid (line 6 in $\mathsf{Verify}(\cdot)$ of Figure 2b).

Other possible constructions. SeqPoW is mainly based on the succinct proof of ISFs. In addition to VDFs and Sloth, Incremental Verifiable Computation (IVC) [96] 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 update the last step's proof with little effort, rather than computing a new one from scratch.

The advantage of IVC-based SeqPoW is that it supports any ISFs. This means IVC-based SeqPoW can be more egalitarian by using ISFs that are hard to parallelise and optimise. However, IVC is usually constructed from complicated cryptographic primitives, such as SNARKs [33, 37, 38, 81, 96], making it inefficient and challenging to implement. In addition, when generating proofs takes non-negligible time, IVC-based SeqPoW may not be fair, as miners with powerful hardware can execute $\mathsf{SeqPoW.Prove}(\cdot)$ and provide proofs faster.

Non-outsourceable SeqPoW. If $\mathsf{Init}(\cdot)$, $\mathsf{Solve}(\cdot)$ and $\mathsf{Prove}(\cdot)$ take the public key rather than the secret key as input, then the construction cannot prevent *outsourcing*: with others' public keys, the adversary can solve others' SeqPoW puzzles. To prevent outsourcing, one can 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 evaluation [19]. As long as secret keys are kept in secret, the adversary cannot execute $\mathsf{VRFHash}(\cdot)$ for other nodes. This modification introduces negligible overhead to $\mathsf{SeqPoW}_{\mathsf{VDF}}$, but non-negligible overhead to $\mathsf{SeqPoW}_{\mathsf{Sloth}}$, as the verifier should verify all VRF outputs and proofs for assuring no prior solution satisfies the difficulty. More efficient non-outsourceable unique SeqPoW constructions are considered as future work.

2.5 Security and efficiency analysis

Security. Appendix B provides full security proofs for our SeqPoW constructions. *Completeness* and *Soundness* directly follow the completeness, soundness and self-composability of

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

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 $(sk, pk)$

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

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

Prove($pp, pk, i, x, S_i$)
  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: $\pi_{\mathsf{VDF}} \leftarrow \mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, S_0, S_i, i \cdot \psi)$
  5: return $\pi_{\mathsf{VDF}}$

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 $\mathsf{VDF.Verify}(pp_{\mathsf{VDF}}, S_0, S_i, \pi_i, i \cdot \psi) = 0$
  5:   return 0
  6: if $H(pk \| S_i) > \frac{2^\kappa}{T}$ then return 0
  7: return 1
```

(a) $\mathsf{SeqPoW}_{\mathsf{VDF}}$.

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

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 $(sk, pk)$

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

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

Prove($pp, pk, i, x, S_i$)
  1: return $\perp$

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
```

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

Figure 2: Construction of SeqPoW.

Sloth and VDFs. By *pseudorandomness* of $H_G(\cdot)$ and *sequentiality* of Sloth and VDFs, Solve$(\cdot)$ outputs unpredictable solutions. Then, by $H(\cdot)$'s *pseudorandomness* and 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.

VDFs may use cyclic groups that require trusted setup, e.g., two prevalent VDFs [87, 98] use RSA groups. In practice, trusted setup can be done by a trusted party, or a specialised multi-party protocols [47, 48].

Efficiency (Table 4). $\mathsf{SeqPoW}_{\mathsf{VDF}}$ and $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ employ repeated squaring on an RSA group and repeated square rooting on a prime-order group as ISFs, respectively. Let s be the size (in Bytes) of a group element, and ψ be the step parameter. Each Solve$(\cdot)$ executes ψ steps of the ISF, and the prover attempts Solve$(\cdot)$ for T times on average to find a valid solution. Thus, Prove$(\cdot)$ generates proofs on ψT steps, and Verify$(\cdot)$ verifies proofs of ψT steps.

We analyse $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Wesolowski's VDF (Wes19) [98] and Pietrzak's VDF (Pie19) [87] seaparately, without considering optimisation or parallelilsation techniques [27, 87, 98]. According to the existing analysis [35], the proving complexity, verification complexity and proof size of Wes19 are $O(\psi T)$, $O(\log \psi T)$ and s Bytes, respectively; and those of Pie19 are $O(\sqrt{\psi T} \log \psi T)$, $O(\log \psi T)$ and $s \log_2 \psi T$, respectively. When $\psi T = 2^{40}$ and $s = 32$ Bytes, the proof sizes of $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Wes19 [98] and with Pie19 [87] are 32 and 1280 Bytes, respectively. $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ has the verification complexity of $O(\psi T)$ and does not have proofs.

Table 4: Efficiency of two SeqPoW constructions.

	Solve$(\cdot)$	Prove$(\cdot)$	Verify$(\cdot)$	Proof size (Bytes)
$\mathsf{SeqPoW}_{\mathsf{VDF}}$	$O(\psi)$	$O(\psi T)$	$O(\log \psi T)$	s
	$O(\psi)$	$O(\sqrt{\psi T} \log \psi T)$	$O(\log \psi T)$	$s \log_2 \psi T$
$\mathsf{SeqPoW}_{\mathsf{Sloth}}$	$O(\psi)$	0	$O(\psi T)$	0

3 RandChain: DRB from SeqPoW

Figure 3 describes the full construction of RandChain.

System setting. Similar to other DRBs, RandChain works in a permissioned network. Nodes $\mathcal{P} = \{p_1, \dots, p_n\}$ register themselves into the system through a Public Key Infrastructure (PKI). The PKI 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 uniquely identified by its public key in the system. Each node is only

```
mainChain($\mathcal{C}_k$)
1: $\mathcal{MC}_k \leftarrow \varepsilon$
2: foreach $\mathcal{C}_k^t \in \mathcal{C}_k$
3:     $\mathcal{MC}_k \leftarrow max(\mathcal{C}_k^t, \mathcal{MC}_k)$
4: return $\mathcal{MC}_k$

MainProcedure($pp, sk_k, pk_k$)
1: Synchronise chain as $\mathcal{C}_k$
2: MineRoutine($pp, sk_k, pk_k, \mathcal{C}_k$)
       in a thread
3: SyncRoutine($pp, \mathcal{C}_k$)
       in a thread

SyncRoutine($pp, \mathcal{C}_k$)
1: while $True$
2:     Wait for a new block as $B$
3:     $(h^-, h, i, S, pk, \pi) \leftarrow B$
4:     if $h^- \notin \mathcal{C}_k$ 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 $\mathcal{C}_k$ after block with hash $h^-$
9:     Propagate $B$

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

Figure 3: Construction of RANDCHAIN.

directly connected to a subset of peers, and does not know the exact number of nodes in the system.

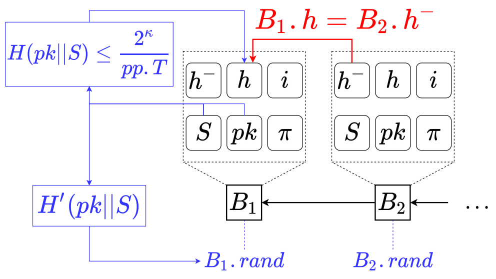
Blockchain structure (Figure 4a). Each block B is of the format (h^-, h, i, S, pk, π), where h^- is the previous block ID, h is the current block ID, i is the SeqPoW solution index, S is the SeqPoW solution, pk is the public key of this block's creator, and π is the proof that S is a valid SeqPoW solution on input h^-. Let $H, H' : \{0,1\}^* \rightarrow \{0,1\}^\kappa$ be two different hash functions. The ID $B.h$ and random output $B.rand$ of block B are calculated as $B.h = H(pk\|S)$ and $B.rand = H'(pk\|S)$, respectively.

Each node p_k has its local view $\mathcal{C}_k$ of the blockchain. View $\mathcal{C}_k$ may have *forks*, i.e., multiple blocks are following the same block. RANDCHAIN applies Nakamoto consensus that, nodes always mine on the longest chain, as specified in mainChain$(\cdot)$.

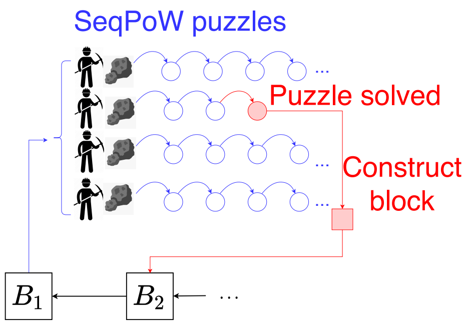
Protocol execution (Figure 4b). Node p_k runs two routines: the synchronisation routine SyncRoutine$(\cdot)$ and the mining routine MineRoutine$(\cdot)$. In SyncRoutine$(\cdot)$, node p_k synchronises its local blockchain $\mathcal{C}_k$ with other nodes. The synchronisation process is the same as in 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 appending new blocks to the main chain $\mathcal{MC}_k$. In particular, node p_k solves a SeqPoW puzzle S derived from SeqPoW.Init$(pp, sk_k, B^-.h)$, where pp is the public parameter, sk_k is its secret key, and $B^-.h$ is the hash of $\mathcal{MC}_k$'s last block. To solve puzzle S, node p_k keeps executing SeqPoW.Solve$(\cdot)$ until finding a solution that satisfies the difficulty. With a valid solution S, node p_k constructs a block B consisting of a random output, and appends B to $\mathcal{MC}_k$.

Non-parallelisable mining. PoW mining is usually centralised and energy-inefficient. Nodes with more computational power have more advantage, leading to an arms race of mining hardware between nodes. Eventually, few nodes with massive mining hardware dominate the mining, which leads to various security issues, e.g., 51% attacks, and cost a great amount of electricity [4]. A promising solution to keep mining decentralised and energy-efficient is *non-parallelisable mining* [2], where each node with its unique key pair can use at most a single processor to mine. Compared to PoW mining that is parallelisable, non-parallelisable mining is more energy-efficient as a fully operating processor costs less energy than massively parallel mining hardware, and more decentralised as powerful nodes cannot obtain much speedup on mining. RANDCHAIN, for the first time, achieves mining *non-parallelisable*. This is because 1) each node has a unique input of the PoW puzzle, and 2) the PoW puzzle is sequential.



(a) Beacon structure. A block includes the last block hash h^-, current block hash h, node's public key pk, SeqPoW solution S and its index i and proof π. The block hash h should satisfy the difficulty parameter T. The random output is calculated as $H(pk||S)$.



(b) Process of mining. Each node keeps solving its own SeqPoW puzzle. The node who first solves its SeqPoW puzzle (the red one) proposes the next block.

Figure 4: Structure and process of RANDCHAIN.

4 Security analysis of RANDCHAIN

In this section, we analyse the security of RANDCHAIN. We formally define DRBs with security properties, and prove that RANDCHAIN implements a DRB satisfying all properties.

4.1 Security model

We consider a *synchronous* network: all messages are delivered within a known time bound Δ. There is no round or *lock-step synchrony*. The network consists of n nodes, each of which controls *an unlimited number of processors*. Processors of the same node execute at the same speed, but processors of different nodes may have different execution speed.

Let α and β be the total mining power of the adversary and the honest nodes *on a single block*, where $\alpha + \beta = 1$. The adversary is *adaptive*: it can corrupt any set of nodes with at most α mining rate at any time; can coordinate corrupted nodes without delay; and can arbitrarily delay, drop, forge and modify messages from its nodes. Honest nodes only mine on the latest block, and the adversary can mine on any block.

DRBs are usually used for supporting other applications (e.g., jackpot) that rely on randomness. We make the following assumptions for the external environment for DRBs: 1) randomness-based applications can securely access the RANDCHAIN blockchain; 2) users (e.g., gamblers) participate in randomness-based applications, and can also securely access the RANDCHAIN blockchain; and 3) nodes, users and applications are controlled by different sets of people who do not collude with each other.

4.2 Defining DRBs

DRBs should satisfy five properties, namely *consistency*, *liveness*, *fairness*, *uniform distribution* and *unpredictability*.

Consistency and liveness. Similar to Nakamoto consensus, DRB should satisfy *consistency* and *liveness* [45, 65, 85, 88]. *Consistency* specifies that nodes can only have different views on the latest ℓ blocks, where ℓ is the degree of consistency guarantee. Some randomness-based applications require RB to have *finality* [97], i.e., at any time, correct nodes do not have conflicted views on the blockchain, which is equivalent to 0-consistency or *agreement* in Byzantine consensus [40]. In Appendix D.2 we discuss how to add *finality* to RANDCHAIN.

Definition 9 (ℓ-Consistency)**.** A DRB satisfies ℓ-consistency if for any two correct nodes at any time, their chains can differ only in the last $\ell \in \mathbb{N}$ blocks.

Liveness specifies the lower and upper bound of the mining speed, i.e., the number of random outputs produced per time unit. If the speed does not reach the lower bound, then the DRB cannot satisfy the requirement of real-world applications. If the speed exceeds the upper bound, then this means an adversary is producing random outputs faster than all honest nodes. Such adversary can take advantage in time-sensitive applications to gain extra profit.

Some papers define *termination* [41, 67, 74] or *Guaranteed Output Delivery (G.O.D.)* [42, 43, 49, 92] for DRBs that, for every round, a new random output will be produced. We do not follow their definitions, as 1) competitive DRBs do not have the concept of rounds, and 2) these definitions do not specify the upper bound for mining speed.

Definition 10 ((t, τ^-, τ^+)-Liveness)**.** A DRB satisfies (t, τ^-, τ^+)-liveness if for any time period t, every correct node receives $[t \cdot \tau^-, t \cdot \tau^+]$ new outputs, where $t, \tau^-, \tau^+ \in \mathbb{R}^+$.

Fairness. Unlike collaborative DRBs that are usually "one-man-one-vote", nodes in competitive DRBs may have different mining power. We define *fairness* to specify the maximum mining power difference among nodes in the network. A node's mining power is quantified by the probability that it mines the next block before other nodes. *Fairness* implies the decentralisation level of DRBs. If few powerful nodes control mining power much greater than other nodes, then the DRB is dominated by these powerful nodes, making the network highly centralised.

Definition 11 (μ-Fairness)**.** The DRB satisfies μ-fairness if the following holds for the largest possible $\mu \in (0,1]$. Assuming all messages are delivered instantly and nodes in a DRB agree on a blockchain of length ℓ. Let $X(p_k)$ be the event that node p_k mines the $(\ell+1)$-th block earlier than other nodes. For any two nodes p_i and p_j,

$$\mu \leq \frac{\Pr[X(p_i)]}{\Pr[X(p_j)]}$$

When $\mu = 1$, the DRB achieves ideal fairness and the network is fully decentralised, and vice versa when $\mu \to 0$.

Uniform distribution. As a DRB, each output should be *uniformly distributed*, i.e., pseudorandom. Some papers [36] refer this property as *unbiasability*.

Definition 12 (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.

Unpredictability. Each output should be *unpredictable*: given the current blockchain, no node can predict the next random output before it is produced. If the adversary can predict future random outputs, then it may take advantage in randomness-based applications. Some papers [49, 92, 93, 95] define *unbiasibility* that, the adversary cannot manipulate the random output to its preferred value. This is a special case of *unpredictability*: the adversary guesses its preferred value, then attempts to choosing an input which makes the random output equal to its preferred value.

Definition 13 (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 in the DRB 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 $\Pr[r = r']$.

4.3 Security analysis

Consistency and liveness. RandChain achieves the same consistency and liveness bound $\alpha \leq \frac{1}{1+e}$ as Proof-of-Space (PoSpace)-based Nakamoto consensus [56]. Similar to PoSpace-based Nakamoto consensus, the optimal attack on RandChain is the *grinding attack*: the adversary allocates a processor to mine on each of existing blocks. Compared to 51% attacks on PoW-based consensus, grinding attack amplifies the adversary's mining power by up to e, and the consensus protocol tolerates fewer faulty nodes, i.e., $\alpha \leq \frac{1}{1+e}$. We refer readers to the papers [29, 56] for detailed analysis.

Fairness. RandChain achieves μ-fairness where $\mu > \frac{1}{5}$. With *non-parallellisable mining*, mining can only be accelerated by using processors with higher clock rate. While laptops' clock rate – as the baseline – is usually 2$\sim$3 GHz, the highest clock rate achieved by processors is 8.723 GHz [1]. Given the voltage limit, the clock rate is hard to improve further [20]. Hence, one can accelerate solving SeqPoW for less than five times, leading to $\mu > \frac{1}{5}$. The limited speedup is also evidenced by the recent *VDF Alliance FPGA Contest* [16–18], where optimised VDF implementations are approximately four times faster than the baseline implementation.

When network delay exists, the adversary can launch *selfish mining* attacks, i.e., adaptively withhold blocks to mine the portion of blocks more than its portion of mining power [62], compromising the fairness guarantee of Nakamoto consensus. We refer to existing countermeasures [69, 84, 105] to defend against selfish mining attacks.

Uniform distribution. For every block B, $B.rand$ is produced by the hash function $H'(\cdot)$. By pseudorandomness of hash functions, $B.rand$ is indistinguishable with a uniformly random κ-bit string.

Unpredictability. Appendix C provides formal proofs of unpredictability. 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*. When the $(\ell+1)$-th block is produced by correct nodes, the adversary's best strategy is guessing, leading to negligible advantage. When the $(\ell+1)$-th block is produced by the adversary's nodes, the adversary's best strategy is to produce as many blocks as possible before receiving a new block from honest nodes. By SeqPoW's *hardness*, the adversary can only produce a limited number of blocks in this time period. By SeqPoW's *sequentiality*, the adversary can make accurate guesses on its SeqPoW solutions only with negligible probability. Therefore, the adversary's advantage on the prediction game is negligible.

If RandChain employs SeqPoW with *uniqueness*, then the advantage is even smaller, as each of the corrupted nodes can only mine a single block at height $(\ell+1)$.

5 Implementation and evaluation

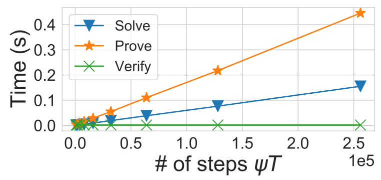
We implement SeqPoW and RandChain, and evaluate their performance. For SeqPoW, the evaluation shows that all SeqPoW constructions are practical, and SeqPoW$_{\text{VDF}}$ with Pie19 [87] is most suitable for instantiating RandChain, given its trade-off on generating/verifying proofs and the proof size. For RandChain, the evaluation shows that on a cluster with up to 1024 nodes, a random output can be propagated to the majority of nodes within 1.3 seconds; nodes have comparable chance of producing blocks; and the average bandwidth utilisation is less than 200KB/s even when the interval between two random outputs is only one second. We will make all code and experimental data publicly accessible after the paper is published.

5.1 SeqPoW: microbenchmarks

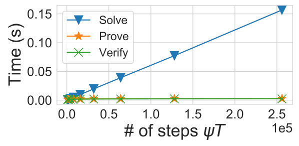
We microbenchmark all SeqPoW constructions, including $\mathsf{Solve}(\cdot)$, $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ for $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with two VDFs, and $\mathsf{Solve}(\cdot)$ and $\mathsf{Verify}(\cdot)$ for $\mathsf{SeqPoW}_{\mathsf{Sloth}}$.

Implementation. We implement the the SeqPoW constructions in Rust. We use the `rug` [6] crate for big integer arithmetic, and implement the RSA group with 1024-bit keys and the group of prime order based on `rug`. We implement the two $\mathsf{SeqPoW}_{\mathsf{VDF}}$ constructions based on the RSA group, and $\mathsf{SeqPoW}_{\mathsf{Sloth}}$ based on the group of prime order. Our implementations strictly follow their original papers [76, 87, 98].

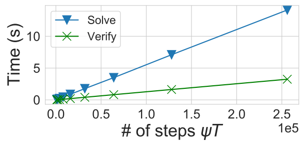
Experimental setting. For each function, we test $\psi T \in [1000, 2000, 4000, 8000, 16000, 32000, 64000, 128000, 256000]$, where ψ is the step parameter and T is the difficulty. The code for microbenchmarking is based on Rust's native benchmarking suite `cargo-bench` [5] and `criterion` [7]. We sample ten executions for each group (i.e., a function with a unique set of parameters) of the experiments, and specify O3-level optimisation for compilation. 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.



(a) $\mathsf{SeqPoW}_{\mathsf{VDF}}$ + Wes19 [98].



(b) $\mathsf{SeqPoW}_{\mathsf{VDF}}$ + Pie19 [87].



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

Figure 5: Evaluation of SeqPoW constructions.

Performance (Figure 5). For all SeqPoW constructions, the running time of $\mathsf{Solve}(\cdot)$ increases linearly with ψT. This is as expected as $\mathsf{Solve}(\cdot)$ is dominated by the ISF. For $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Wes19, $\mathsf{Prove}(\cdot)$ takes more time than $\mathsf{Solve}(\cdot)$, as our implementation does not apply optimisations (such as [27]). For $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Pie19, $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ take negligible time compared to $\mathsf{Solve}(\cdot)$. 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 the theoretically optimal value, i.e, $\log_2 |G| = 1024$ in our case [22], the verification overhead is acceptable when random outputs are not generated frequently.

5.2 RandChain: end-to-end evaluation

The evaluation focuses on three metrics as follows:

- **Latency** is the time taken for propagating a block to the network. It is quantified by the block propagation delay (BPD), i.e., the time taken for the majority of nodes to receive a block.
- **Decentralisation** is the evenness of nodes' chance of producing blocks. It is quantified by the distribution of nodes in terms of the number of blocks they produce on the main chain.
- **Network overhead** is the overhead of data transfer during the protocol execution. It is quantified by the average bandwidth utilisation, i.e., the average amount of data transferred in a time unit, of a node.

We compare these three metrics with state-of-the-art results mentioned in HydRand [93] and RandHerd [95].

Implementation. We implement RandChain based on `Parity-bitcoin` [12], a Bitcoin implementation in Rust. It uses `RocksDB` [11] for storage, and Bitcoin's `Wire` protocol [21] for the P2P protocol stack. We adapt the blockchain structure, SeqPoW and relevant message types to RandChain's setting specified in §3. Given the evaluation result in §5.1, we use SeqPoW with Pie19 for RandChain. The entire project takes approximately 23000 lines of code (LoC), where the RandChain implementation adds/changes approximately 4500 LoC over `Parity-bitcoin`. We use `dstat` [10] for monitoring system resource utilisation.

Experimental setting. We specify O3-level optimisation for compilation, and deploy the project to clusters with $\{128, 256, 512, 1024\}$ nodes on Amazon's EC2 instances. Specifically, we rent $\{16, 32, 64, 128\}$ `t2.micro` EC2 instances (1 GB RAM, one CPU core and 60-80 Mbit/s network bandwidth) in 13 cities around the globe (North Virginia, North California, Oregon, Ohio, Canada, Mumbai, Seoul, Sydney, Tokyo, Singapore, Ireland, Sao Paulo, London, and Frankfurt), and each instance runs 8 RandChain nodes. Each node maintains up to 8 outbound connections and 125 inbound connections, which is same as Bitcoin's setting [21]. When a node starts, it randomly connects to 8 peers, accepts connections from other peers, and starts gossiping messages with them. We test blocktime (i.e., the average time interval between two blocks) of $\{1, 5, 10\}$ seconds by adjusting the SeqPoW difficulty. For each group of the experiments, we sample 30 minutes of the execution, collect logs from all nodes, parse the logs and calculate the three metrics.

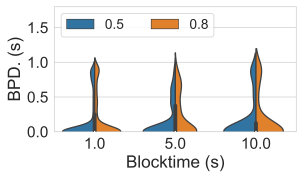
Block size. The major part of a block is the SeqPoW proof that takes $s \cdot \log_2(\psi T)$ Bytes, where ψT depends on 1) the time taken to find a solution and 2) the number of iterations executed in a time unit. For our SeqPoW implementation, the `t2.micro` EC2 instance can increment $\mathsf{Solve}(\cdot)$ for 233868 iterations per second on average. Given blocktime t, the SeqPoW proof size is $s \cdot \log_2(233868 \cdot t) \approx s \cdot (18 + \log_2 t)$. When blocktime is $\{1, 5, 10\}$ seconds and $s = 32$ Bytes, the block size is about

{576,1336,1912} Bytes. When blocktime is 60 seconds (the setting of Drand [9] and the NIST randomness beacon [71]), the block size is about 3402 Bytes.

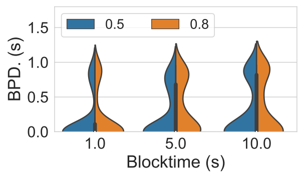
Latency. Figure 6 shows the distribution of BPD for different sizes of clusters. The BPD never exceeds 1.3 seconds, meaning that the system can produce more than 45 random outputs per minute. The result is promising compared to HydRand ($\sim$7 random outputs per minute on a 128-node cluster) and RandHerd ($\sim$20 random outputs per minute on a 1024-node cluster). The reason is that RANDCHAIN achieves linear communication complexity, and a block can reach most nodes within 2 hops. In Figure 6, BPD is usually either less than 0.4s or more than 0.6s, but is hardly in-between values. The two peaks around the saddle of 0.4$\sim$0.6s indicate the average delays for 1-hop and 2-hop block propagation, respectively.

In addition, the average BPD increases slowly with increasing number of nodes, which is consistent with other linear blockchain protocols [104]. In linear protocols, the average BPD is in proportion to the average number of intermediate nodes of two random nodes. With Bitcoin's random peer selection mechanism, the network is structured as an Erdos-Renyi random graph [61], where the average number of intermediate nodes is $O(\frac{\log n}{\log k})$, where n and k is the number of nodes and the average degree of nodes, respectively.

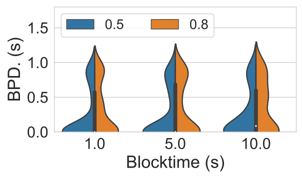
A less important result is that BPD increases when blocks are produced more frequently. This is because a `t2.micro` instance only has a single processor and limited network capacity, making the overhead of verifying and propagating blocks non-negligible.



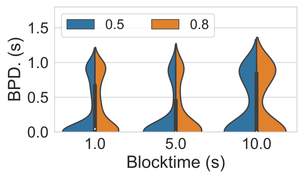
(a) 128 nodes.



(b) 256 nodes.



(c) 512 nodes.



(d) 1024 nodes.

Figure 6: Distribution of block propagation delay. The blue and orange parts indicate the distribution of BPD when blocks are propagated to 50% and 80% of nodes, respectively. Such figures are known as *violin plots* [15].

Decentralisation. Figure 7 shows the distribution of nodes in terms of the number of blocks they produce on the main chain. The data is collected from the group of experiment with 1024 nodes and the blocktime of 1 second. The kernel estimated distribution is close to the normal distribution, meaning that nodes have roughly equal chance of producing blocks. The result is consistent with our experimental setting where nodes' processors share the same set of parameters, and comparable with HydRand and RandHerd that are "one-man-one-vote".

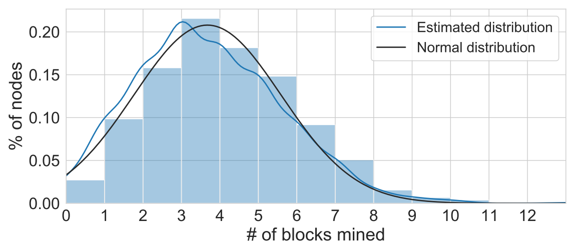


Figure 7: Distribution of the number of blocks produced by distinct nodes. The blue and black lines are the kernel density estimation and the closest normal distribution, respectively.

Network overhead. Figure 8 shows the bandwidth utilisation result. It shows that RANDCHAIN utilises less bandwidth compared to HydRand and RandHerd: even with blocktime of 1 second, each node utilises $\sim$200KB/s bandwidth per second, which is comparable with HydRand (180$\sim$310KB/s on a 128-node cluster) and RandHerd ($\sim$200KB/s on a 1024-node cluster). The bandwidth utilisation remains stable with increasing number of nodes, as RANDCHAIN is linear. These two results are as expected since RANDCHAIN is linear. The inbound and outbound bandwidth are comparable, as the input (i.e., the last block) and the output (i.e., the new block) are comparable in terms of the size, leading to identical bandwidth utilisation. With longer blocktime, the node requires less bandwidth, as nodes send and receive blocks less frequently.

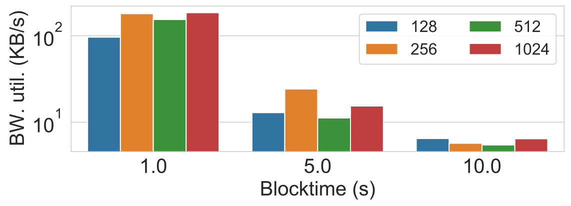


Figure 8: Bandwidth utilisation.

6 Comparison with existing DRBs

In this section, we develop a unified evaluation framework for DRBs, and compare RANDCHAIN with existing DRBs. Our evaluation shows that RANDCHAIN is the only protocol that is leaderless, secure and efficient simultaneously, without relying on lock-step synchrony or trusted parties.

Table 5: Comparison of RANDCHAIN with existing DRBs. Red indicates strong assumptions, unsatisfied security properties, and poor efficiency. Yellow indicates moderate assumptions, partly satisfied security properties, and moderate efficiency. Green indicates weak assumptions, satisfied security properties, and satisfactory efficiency.

	Protocol		**System setting**			**Security properties**						**Efficiency**	
	Name	**Primitives**	**Net. model**	**Trust**	**Fault tol. cap.**	**Consistency**	**Liveness**	**Fairness**	**Uniform dist.**	**Unpred.**	**Pub. ver.**	**Comm. compl.**	**Energy**
DRG-based DRBs	CKS00 [41]	Thr. Sig.	LS sync.$^{\odot}$	Leader	1/3	✓	✓	1	✓	✓	✓	$O(n^3)$	✓
	HERB [49]	Homo. Thr. Enc.	LS sync.	Leader	1/3	✓	✓	1	✓	✓	✓	$O(n)$	✓
	Dfinity [68]	VRF + Thr. Sig.	LS sync.	Leader	1/3	✓	✓	1	✓	✓	✓	$O(cn)^{\diamond}$	✓
	Ouro. Praos [54]	VRF	LS sync.	Leader	1/2	✓	✓	1	✓	✗†	✓	$O(n)$	✓
	GLOW20 [64]	VRF	LS sync.	Leader	1/3	✓	✓	1	✓	✓	✓	$O(n)$	✓
	Algorand [67]	VRF	LS sync.	Leader	1/3	✓	✓	1	✓	✗†	✓	$O(cn)^{\diamond}$	✓
	Ouroboros [72]	PVSS	LS sync.	Leader	1/2	✓	✓	1	✓	✓	✓	$O(n^3)$	✓
	SCRAPE [42]	PVSS	LS sync.	Leader	1/2	✓	✓	1	✓	✓	✓	$O(n^3)$	✓
	RandShare [95]	PVSS	LS sync.	-	1/3	✓	✓	1	✓	✓	✓	$O(n^3)$	✓
	RandHound [95]	PVSS	LS sync.	Leader	1/3	✓	✓	1	✓	✓	✓	$O(c^2n)^{\diamond}$	✓
	RandHerd [95]	PVSS	LS sync.	Leader	1/3	✓	✓	1	✓	✓	✓	$O(c^2 \log n)^{\diamond}$	✓
	HydRand [93]	PVSS	LS sync.	-	1/3	✓	✓	1	✓	✓	✓	$O(n^2)$	✓
	Albatross [43]	PVSS	LS sync.	Leader	1/2	✓	✓	1	✓	✓	✓	$O(n)$	✓
	KMS20 [74]	HAVSS	LS sync.	-	1/3	✓	✓	1	✓	✓	✓	$O(n^4)$	✓
SC-based DRBs	RanDAO [13]	VDF	Async.	Blockchain	1/2	✓	✓	1	✓	✗‡	✓	$O(n)$	✗$^{\circ}$
	YAGK20 [100]	Escrow-DKG	Async.	Blockchain	1/3	✓	✓	1	✓	✓	✓	$O(n)$	✗$^{\circ}$
ISF-based DRBs	Unicorn [76]	Sloth	Async.	Setup	(n-1)/n	✓	✗*	-	✓	✓	✓	$O(1)$	✓
	EFKS20 [60]	Continuous VDF	Async.	Setup	(n-1)/n	✓	✗*	-	✓	✓	✓	$O(1)$	✓
	RandRunner [92]	Trapdoor VDF	Async.	Setup	(n-1)/n	✓	✗*	1	✓	✓	✓	$O(n)$	✓
DRBs from ext. entr.	CH10 [50]	Rand. extractors	Async.	Data src.	(n-1)/n	✓	✓	-	✓	✓	✗$^{\blacksquare}$	$O(1)$	✓
	BCG15 [36]	Rand. extractors	Async.	Blockchain	(n-1)/n	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✗$^{\blacksquare}$	$O(1)$	✗$^{\circ}$
	BGB17 [39]	Proof-of-Delay	Async.	Blockchain	(n-1)/n	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✗$^{\blacksquare}$	$O(1)$	✗$^{\circ}$
This work	RANDCHAIN	SeqPoW + Nak. consensus	Sync.	-	1/(1+e)	✓	✓	$> \frac{1}{5}$	✓	✓	✓	$O(n)$	✓

$\odot$ For ALL DRG-based DRBs, without lock-step synchrony, nodes should either wait for a sufficient long time at every view when the network is synchronous, employ a trusted *pacemaker* [103] or employ *round synchronisation* protocols [82, 83] for synchronising rounds.
† As analysed in [43], the leader can manipulate the random outputs.
$\diamond$ We use c to denote the size of groups in Dfinity [68], RandHound and RandHerd [95], and the size of the committee in Algorand [67].
‡ As analysed in [14, 23].
$\star$ In [60, 76], the fastest node always learns random outputs earlier than others. In [92], the adversary can keep corrupting leaders and producing random outputs through the trapdoor VDF.
$\circ$ If based on PoW-based blockchains.
$\blacksquare$ Entropy generated by external source is not verifiable.
$\otimes$ In PoW-based blockchains, mining can be accelerated by using multiple processors in parallel. Nodes with massive mining machines overwhelmingly outperform normal nodes.

6.1 Overview of existing DRBs

RANDCHAIN, as a new family of DRBs, does not belong to any existing types of DRBs. We categorise existing DRBs to four types:

DRG-based DRBs. Nodes to execute the single-shot DRG protocol periodically. DRG can be constructed from various cryptographic primitives, such as threshold cryptosystems [41, 49, 68], Verifiable Random Functions (VRFs) [54, 64, 67], and Publicly Verifiable Secret Sharing (PVSS) [42, 43, 72, 74, 93, 95].

Cachin et al. [41] construct DRG protocols from threshold signatures. Dfinity [68] reduces communication complexity by sharding. HERB [49] employs the homomorphic threshold encryption, and employs a leader relaying messages to reduce communication complexity.

Ouroboros Praos [54], Algorand [67] and Galindo et al. [64] construct DRG protocols from VRFs. In Ouroboros Praos [54] and Algorand [67], a leader evaluates the current blockchain state with VRF, and uses the VRF output as the random output. Galindo et al. [64] employ Distributed VRF (DVRF), where nodes evaluate a common input to produce the random output. It also employs a leader for reducing communication complexity.

In PVSS-based DRG protocols [42, 43, 72, 74, 93, 95], each node chooses a local random input and uses PVSS to share it to other nodes, aggregates all received shares on different random inputs to a single one, broadcasts aggregated shares, and aggregating received shares again to recover the final random output. To tolerate Byzantine nodes, HydRand [93], RandHound and RandHerd [95] enforce nodes to run an extra Byzantine consensus to agree on a subset of shares, and SCRAPE [42] and Albatross [43] use erasure codes for tolerating Byzantine shares. To reduce communication complexity, RandHound and RandHerd [95] apply sharding techniques, and Albatross [43] employs a leader for relaying messages. To tolerate network asynchrony, Kogias et al. [74] employs High-threshold Asynchronous Verifiable Secret

Sharing (HAVSS), an asynchronous PVSS variant.

Smart contract (SC)-based DRBs. Nodes participate in a smart contract dedicated for generating random outputs. RANDAO [13] allows anyone to submit their random inputs to the smart contract, and the smart contract combines submitted inputs to a single random output. Yakira et al. [100] construct SC-based DRBs from Escrow Distributed Key Generation (DKG) [101], a DKG variant with game-theoretical security against rational adversaries.

DRBs from external entropy. Nodes periodically extract randomness from real-world entropy, e.g., real-time financial data [50] and public blockchains [26, 36, 39].

Iteratively sequential function (ISF)-based DRBs. Nodes use intermediate outputs of an ISF as random outputs, and succinct proofs for the ISF to make outputs verifiable. Such ISFs include Sloth [76] and Continuous VDFs [60]. RandRunner [92] extends this paradigm by allowing nodes to execute the ISF in turn.

6.2 Evaluation framework for DRBs

Existing DRBs work in permissioned networks where each node has a unique identity. The framework consists of three aspects, namely system setting, security properties and efficiency. System setting considers network model, trust assumption and fault tolerance capacity. Network model specifies the timing guarantee of messages. A network is lock-step synchronous if the protocol executes in rounds and all messages are delivered before the end of each round; is synchronous if messages are delivered within a known finite time-bound; is asynchronous if messages are delivered without a known time-bound; or is partially synchronous [58] if messages are delivered within a known finite time-bound with some clock drift. Trust means the trustworthy components that the DRB relies on in order to guarantee all security properties. Fault tolerance capacity means the threshold of Byzantine nodes that the DRB can tolerate.

Security properties are defined in §4. We also consider *public verifiability*: whether a random output is publicly verifiable. Efficiency considers communication complexity [102] and energy efficiency, i.e., the amount of communication and energy needed for running the DRB, respectively.

6.3 Evaluation

Table 5 summarises the comparison.

System setting. All DRG-based DRBs have to assume lock-step synchrony, *regardless of their underlying DRGs' network models*. As nodes execute in rounds, DRG-based DRBs require *round synchronisation* [82], i.e., nodes are always executing the same round of the protocol. If nodes do not agree on the round number, then the system may lose consistency or liveness forever [83]. To guarantee *round synchronisation* without assuming *lock-step synchrony*, DRG-based DRBs should either assume synchronous networks while enforcing nodes to stay in every view for more than the network delay; employ a trusted *pacemaker* [103] for announcing round numbers; or enforce nodes to additionally run a *round synchronisation* protocol [82, 83]. All of these approaches introduce extra trust or overhead. The other three types of DRBs work in asynchronous networks: SC-based DRBs rely on periodical execution of smart contracts, and ISF-based DRBs and DRBs from external entropy allow nodes to compute random outputs locally.

A large number of DRG-based DRBs rely on leaders. SC-based DRBs and DRBs from external entropy should rely on trustworthy smart contracts and the entropy source, respectively. ISF-based DRBs rely on trusted setup (e.g., [48, 70]) of the random seed, otherwise the adversary who previously knows the seed can learn random outptus earlier than honest nodes. All of the DRBs achieve satisfactory fault tolerance capacity. Notably, for ISF-based DRBs and DRBs from external entropy, as long as there is a node is honest, the system works as expected, as nodes can compute random outputs locally without interacting with other nodes.

Security properties. All DRG-based DRBs achieve consistency and liveness, except that ISF-based DRBs do not satisfy liveness. For Sloth and Continuous VDF-based DRBs [60, 76], nodes with fastest processors can learn random outputs earlier than other nodes. In RandRunner [92], the adversary can keep corrupting leaders and computing random outputs via the trapdoor. All DRBs achieve the ideal fairness, i.e., $\mu = 1$, except for DRBs from PoW-based blockchains [36, 39] with $\mu \to 0$ as mining can be accelerated by massive parallelism, and RANDCHAIN with $\mu > \frac{1}{5}$. All DRBs satisfy *uniform distribution*. Ouroboros Praos [54] and Algorand [67] do not satisfy unpredictability, as the leader can manipulate random outputs [43]. RanDAO does not satisfy unpredictability as analysed in [14, 23]. All DRBs from external entropy do not satisfy public verifiability, as the external entropy is not publicly verifiable.

Efficiency. All DRBs achieve communication complexity of less than $O(n^2)$, except for leaderless DRG-based DRBs. This is because without leaders, DRG requires nodes to make all-to-all broadcasts to agree on a unique random output. All DRBs are energy-efficient except for SC-based DRBs and DRBs using public blockchains, as public blockchains usually rely on PoW-based consensus that is energy-greedy.

7 Conclusion

In this paper, we identify a new family of Decentralised Randomness Beacon (DRB) protocols where nodes are competitive, and construct the first DRB protocol RANDCHAIN that belongs to this class. The theoretical analysis and experimental evaluation show that RANDCHAIN achieves promising security and performance without relying on strong assumptions.

Acknowledgement

We thank Jieyi Long for discussions on the "winner-takes-all" problem, Jianyu Niu and Sreeram Kannan for referring us to the latest results [29, 55, 56] on analysing Nakamoto consensus, and Omer Shlomovtis for insightful comments.

References

[1] AMD Breaks 8GHz Overclock with Upcoming FX Processor, Sets World Record. http://hothardware.com/News/AMD-Breaks-Frequency-Record-with-Upcoming-FX-Processor/.

[2] [ANSWERED] Why is bitcoin proof of work parallelizable ? https://bitcointalk.org/index.php?topic=46739.0.

[3] BCH Avalanche Transactions Show Finality Speeds 10x Faster Than Ethereum. https://news.bitcoin.com/bch-avalanche-transactions-show-finality-speeds-10x-faster-than-ethereum/.

[4] Bitcoin's energy consumption 'equals that of Switzerland'. https://www.bbc.com/news/technology-48853230.

[5] cargo-bench. https://doc.rust-lang.org/cargo/commands/cargo-bench.html.

[6] crates/rug. https://crates.io/crates/rug.

[7] criterion.rs. https://github.com/bheisler/criterion.rs.

[8] Distributed Randomness Beacon | Cloudflare. https://www.cloudflare.com/leagueofentropy/.

[9] Drand - Distributed Randomness Beacon. https://drand.love/.

[10] dstat-real/dstat: Versatile resource statistics tool. https://github.com/dstat-real/dstat.

[11] facebook/rocksdb: A library that provides an embeddable, persistent key-value store for fast storage. https://github.com/facebook/rocksdb.

[12] paritytech/parity-bitcoin: The Parity Bitcoin client. https://github.com/paritytech/parity-bitcoin.

[13] RANDAO: A DAO working as RNG of Ethereum. https://github.com/randao/randao.

[14] RNG exploitability analysis assuming pure RANDAO-based main chain. https://ethresear.ch/t/rng-exploitability-analysis-assuming-pure-randao-based-main-chain/1825.

[15] seaborn.violinplot — seaborn 0.11.1 documentation. https://seaborn.pydata.org/generated/seaborn.violinplot.html.

[16] supranational/vdf-fpga-round1-results. https://github.com/supranational/vdf-fpga-round1-results.

[17] supranational/vdf-fpga-round2-results. https://github.com/supranational/vdf-fpga-round2-results.

[18] supranational/vdf-fpga-round3-results. https://github.com/supranational/vdf-fpga-round3-results.

[19] VRF-Based Mining: Simple Non-Outsourceable Cryptocurrency Mining. https://github.com/DEX-ware/vrf-mining/blob/master/paper/main.pdf.

[20] Why has CPU frequency ceased to grow? https://software.intel.com/content/www/us/en/develop/blogs/why-has-cpu-frequency-ceased-to-grow.html.

[21] Protocol documentation - Bitcoin Wiki, 2015. https://en.bitcoin.it/wiki/Protocol_documentation.

[22] Hamza Abusalah, Chethan Kamath, Karen Klein, Krzysztof Pietrzak, and Michael Walter. Reversible proofs of sequential work. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 277–291. Springer, 2019.

[23] Musab A Alturki and Grigore Roşu. Statistical Model Checking of RANDAO's Resilience to Pre-computed Reveal Strategies. In *International Symposium on Formal Methods*, pages 337–349. Springer, 2019.

[24] Joël Alwen, Binyi Chen, Krzysztof Pietrzak, Leonid Reyzin, and Stefano Tessaro. Scrypt is maximally memory-hard. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 33–62. Springer, 2017.

[25] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15, 2018.

[26] Marcin Andrychowicz and Stefan Dziembowski. Distributed Cryptography Based on the Proofs of Work. *IACR Cryptol. ePrint Arch.*, 2014:796, 2014.

[27] Vidal Attias, Luigi Vigneri, and Vassil Dimitrov. Implementation Study of Two Verifiable DelayFunctions. *IACR Cryptol. ePrint Arch.*, 2020:332, 2020.

[28] Christian Badertscher, Peter Gaži, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 913–930, 2018.

[29] Vivek Bagaria, Amir Dembo, Sreeram Kannan, Sewoong Oh, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Proof-of-Stake Longest Chain Protocols: Security vs Predictability. *arXiv preprint arXiv:1910.02218*, 2019.

[30] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the Libra Blockchain. *The Libra Assn., Tech. Rep*, 2019.

[31] Juan Benet and Nicola Greco. Filecoin: A decentralized storage network. *Protocol Labs*, pages 1–36, 2018.

[32] Alex Biryukov, Daniel Dinu, and Dmitry Khovratovich. Argon2: new generation of memory-hard functions for password hashing and other applications. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 292–302. IEEE, 2016.

[33] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. Recursive composition and bootstrapping for SNARKs and proof-carrying data. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 111–120, 2013.

[34] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[35] Dan Boneh, Benedikt Bünz, and Ben Fisch. A Survey of Two Verifiable Delay Functions. *IACR Cryptol. ePrint Arch.*, 2018:712, 2018.

[36] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On Bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, 2015:1015, 2015.

[37] Sean Bowe, Jack Grigg, and Daira Hopwood. Halo: Recursive Proof Composition without a Trusted Setup. *IACR Cryptol. ePrint Arch.*, 2019:1021, 2019.

[38] Benedikt Bünz, Alessandro Chiesa, Pratyush Mishra, and Nicholas Spooner. Proof-Carrying Data from Accumulation Schemes. *IACR Cryptol. ePrint Arch.*, 2020:499, 2020.

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

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

[41] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

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

[43] Ignacio Cascudo and Bernardo David. ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 311–341. Springer, 2020.

[44] Miguel Castro, Barbara Liskov, et al. Practical Byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[45] Benjamin Y Chan and Elaine Shi. Streamlet: Textbook Streamlined Blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 1–11. ACM, 2020.

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

[47] Megan Chen, Ran Cohen, Jack Doerner, Yashvanth Kondi, Eysa Lee, Schuyler Rosefield, and Abhi Shelat. Multiparty Generation of an RSA Modulus. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 64–93. Springer, 2020.

[48] Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthuramakrishnan Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. *IACR Cryptol. ePrint Arch.*, 2020:374, 2020.

[49] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic Encryption Random Beacon. *IACR Cryptol. ePrint Arch.*, 2019:1320, 2019.

[50] Jeremy Clark and Urs Hengartner. On the Use of Financial Data as a Random Beacon. *EVT/WOTE*, 89, 2010.

[51] Bram Cohen and Krzysztof Pietrzak. Simple proofs of sequential work. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 451–467. Springer, 2018.

[52] Ronald Cramer, Ivan Bjerre Damgård, et al. *Secure multiparty computation*. Cambridge University Press, 2015.

[53] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security*, pages 23–41. Springer, 2019.

[54] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 66–98. Springer, 2018.

[55] Soubhik Deb, Sreeram Kannan, and David Tse. PoSAT: Proof-of-Work Availability and Unpredictability, without the Work, 2020.

[56] Amir Dembo, Sreeram Kannan, Ertem Nusret Tas, David Tse, Pramod Viswanath, Xuechao Wang, and Ofer Zeitouni. Everything is a Race and Nakamoto Always Wins. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 859–878. ACM, 2020.

[57] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In *International Conference on Security and Cryptography for Networks*, pages 65–84. Springer, 2020.

[58] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[59] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *Annual International Cryptology Conference*, pages 139–147. Springer, 1992.

[60] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 125–154. Springer, 2020.

[61] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 5(1):17–60, 1960.

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

[63] Lei Fan and Hong-Sheng Zhou. A scalable proof-of-stake blockchain in the open setting (or, how to mimic nakamoto's design via proof-of-stake). Technical report, Cryptology ePrint Archive, Report 2017/656, 2017.

[64] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons. *IACR Cryptol. ePrint Arch.*, 2020:96, 2020.

[65] Juan A Garay, Aggelos Kiayias, and Nikos Leonardos. Full Analysis of Nakamoto Consensus in Bounded-Delay Networks. *IACR Cryptol. ePrint Arch.*, 2020:277, 2020.

[66] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 3–16, 2016.

[67] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[68] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system. *arXiv preprint arXiv:1805.04548*, 2018.

[69] Ethan Heilman. One weird trick to stop selfish miners: Fresh bitcoins, a solution for the honest miner. In *International Conference on Financial Cryptography and Data Security*, pages 161–162. Springer, 2014.

[70] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash protocol specification. *GitHub: San Francisco, CA, USA*, 2016.

[71] John Kelsey, Luís TAN Brandão, Rene Peralta, and Harold Booth. A reference for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology, 2019.

[72] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[73] Sunny King and Scott Nadal. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August*, 19:1, 2012.

[74] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1751–1767, 2020.

[75] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

[76] Arjen K Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptol. ePrint Arch.*, 2015:366, 2015.

[77] Jieyi Long and Ribao Wei. Nakamoto Consensus with Verifiable Delay Puzzle. *arXiv preprint arXiv:1908.06394*, 2019.

[78] 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*, pages 373–388, 2013.

[79] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed computing*, 11(4):203–213, 1998.

[80] Satoshi Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

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

[82] Oded Naor, Mathieu Baudet, Dahlia Malkhi, and Alexander Spiegelman. Cogsworth: Byzantine View Synchronization. *arXiv preprint arXiv:1909.05204*, 2019.

[83] Oded Naor and Idit Keidar. Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[84] Kevin Alarcón Negy, Peter R Rizun, and Emin Gün Sirer. Selfish mining re-examined. In *International Conference on Financial Cryptography and Data Security*, pages 61–78. Springer, 2020.

[85] 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*, pages 643–673. Springer, 2017.

[86] Colin Percival. Stronger key derivation via sequential memory-hard functions, 2009.

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

[88] Ling Ren. Analysis of Nakamoto Consensus. *IACR Cryptol. ePrint Arch.*, 2019:943, 2019.

[89] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[90] Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability. *arXiv preprint arXiv:1906.08936*, 2019.

[91] Meni Rosenfeld. Analysis of hashrate-based double spending. *arXiv preprint arXiv:1402.2009*, 2014.

[92] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. Cryptology ePrint Archive, Report 2020/942, 2020.

[93] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. HydRand: Efficient Continuous Distributed Randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 32–48.

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

[95] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 444–460. Ieee, 2017.

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

[97] Marko Vukolić. The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication. In *International workshop on open problems in network security*, pages 112–125. Springer, 2015.

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

[99] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[100] David Yakira, Avi Asayag, Ido Grayevsky, and Idit Keidar. Economically Viable Randomness. *arXiv preprint arXiv:2007.03531*, 2020.

[101] David Yakira, Ido Grayevsky, and Avi Asayag. Rational Threshold Cryptosystems. *arXiv preprint arXiv:1901.01148*, 2019.

[102] Andrew Chi-Chih Yao. Some complexity questions related to distributive computing (preliminary report). In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213, 1979.

[103] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[104] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 90–105. IEEE, 2020.

[105] Ren Zhang and Bart Preneel. Publish or perish: A backward-compatible defense against selfish mining in bitcoin. In *Cryptographers' Track at the RSA Conference*, pages 277–292. Springer, 2017.

A Preliminaries of Verifiable Delay Functions

Definition 14 (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.

VDF satisfies the following properties

- *Completeness*: For all λ, x and t,
$$\Pr\left[\begin{array}{c|c} \mathsf{Verify}(pp,x,y, & pp \leftarrow \mathsf{Setup}(\lambda) \\ \pi,t) = 1 & y \leftarrow \mathsf{Eval}(pp,x,t) \\ & \pi \leftarrow \mathsf{Prove}(pp,x,y,t) \end{array}\right] = 1$$
- *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-*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 $\sigma(t)$,
$$\Pr\left[\begin{array}{c|c} & pp \leftarrow \mathsf{Setup}(\lambda) \\ \mathsf{Eval}(x,y,t) = y & \mathcal{A}_1 \leftarrow \mathcal{A}_0(\lambda,t,pp) \\ & y \leftarrow \mathcal{A}_1(x) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

We formally define *self-composability* for VDFs as follows.

Definition 15 (Self-Composability)**.** A VDF (Setup, Eval, Prove, Verify) satisfies self-composability if for all λ, x, (t_1,t_2),

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

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

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

B Security proof of SeqPoW

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

Lemma 2. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies completeness.*

Proof. Assuming a (λ, ψ, T)-valid tuple $(pp, sk, i, x, S_i, \pi_i)$, by *completeness* and Lemma 1, $\mathsf{VDF}.\mathsf{Verify}(\cdot)$ 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 3. $\mathsf{SeqPoW}_{\mathsf{VDF}}$ *satisfies soundness.*

Proof. We prove this by contradiction. Assuming a tuple (pp,sk,i,x,S_i,π_i) that is not (λ,ψ,T)-valid and

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

By *soundness* and Lemma 1, if (y, y^+, π^+, ψ) is generated by $\mathcal{A}$, $\mathsf{VDF}.\mathsf{Verify}(\cdot)$ 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,π_i) is not (λ,ψ,T)-valid, then the adversary can break *soundness*. Thus, this assumption contradicts *soundness*. □

Lemma 4. $\mathsf{SeqPoW_{VDF}}$ *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 5. $\mathsf{SeqPoW_{VDF}}$ *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,π_i) such that $H(pk\|S_i) \leq \frac{2^\kappa}{T}$. □

Lemma 6. *If the underlying VDF satisfies σ-sequentiality, then* $\mathsf{SeqPoW_{VDF}}$ *satisfies σ-sequentiality.*

Proof. We prove this by contradiction. Assuming there exists $\mathcal{A}_1$ which runs in less than time $\sigma(i\cdot\psi)$ 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 $\sigma(i\cdot\psi)$. This contradicts the assumption. □

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

Lemma 7. $\mathsf{SeqPoW_{Sloth}}$ *satisfies uniqueness.*

Proof. We prove this by contradiction. Assuming there exists two $(\lambda,\ \psi,\ T)$-valid tuples $(pp,\ sk,\ i,\ x,\ S_i,\ \pi_i)$ and $(pp,\ sk,\ i,\ x,\ S_i,\ \pi_i)$ where $j < i$. According to $\mathsf{SeqPoW_{Sloth}}.\mathsf{Solve}(\cdot)$, 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 $\mathsf{SeqPoW_{Sloth}}.\mathsf{Verify}(\cdot)$ will pass. However, in the for loop of $\mathsf{SeqPoW_{Sloth}}.\mathsf{Verify}(\cdot)$, if S_i is valid, then verification of S_j will fail. Then, $\mathsf{SeqPoW_{Sloth}}.\mathsf{Verify}(\cdot)$ returns 0, which contradicts the assumption. □

C Proof of unpredictability for RANDCHAIN

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*. When the $(\ell+1)$-th block is produced by correct nodes, the adversary's best strategy is guessing, leading to negligible advantage. When the $(\ell+1)$-th block is produced by the adversary's nodes, the adversary's best strategy is to produce as many blocks as possible before receiving a new block from honest nodes. First, consider RANDCHAIN using SeqPoW without *uniqueness*.

Lemma 8. *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 $\frac{1}{2^\kappa}$.*

If the next output is produced by the adversary's nodes, the adversary's best strategy is to produce as many blocks as possible before receiving a new block from honest nodes. First, consider RANDCHAIN using SeqPoW without *uniqueness*.

Lemma 9. *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 probability $\frac{(e\alpha)^k\beta}{(e\alpha+\beta)^{k+1}}$.*

Proof. With grinding attacks, the adversary amplifies its mining rate by factor e [29, 56]. Thus, the probability that the adversary and honest nodes mine the next block are $\frac{e\alpha}{e\alpha+\beta}$ and $\frac{\beta}{e\alpha+\beta}$, respectively. Note that $\alpha \leq \frac{1}{1+e}$ for satisfying conssitency, and $\alpha+\beta=1$.

Let V_k be the event that "the adversary mines k blocks at height $(\ell+1)$ before correct nodes mine a block at height $(\ell+1)$". When SeqPoW is not unique, a node can mine unlimited number of blocks after a single block. Thus, we have

$$\Pr\left[V_k\right] = \left(\frac{e\alpha}{e\alpha+\beta}\right)^k\cdot\frac{\beta}{e\alpha+\beta} = \frac{(e\alpha)^k\beta}{(e\alpha+\beta)^{k+1}}$$

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

Therefore, with probability $\frac{(e\alpha)^k\beta}{(e\alpha+\beta)^{k+1}}$, the adversary mines k blocks before correct nodes mine a block, leading to the advantage of $\frac{k}{2^\kappa}$. □

Then, we analyse RANDCHAIN using SeqPoW with *uniqueness*. Without the loss of generality, we assume all nodes share the same mining rate.

Lemma 10. *Consider* RANDCHAIN *using SeqPoW with uniqueness. Assuming all nodes share the same mining rate, 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 probability $\Pr\left[V'_k\right]$, where*

$$\Pr\left[V'_k\right] = \prod_{i=0}^{k-1}\frac{(\alpha n-i)e}{(\alpha n-i)e+\beta n}\cdot\frac{\beta}{e\alpha+\beta}$$

Proof. Similar to Lemma 9, the adversary and the honest nodes control mining rate $\frac{e\alpha}{e\alpha+\beta}$ and $\frac{\beta}{e\alpha+\beta}$, respectively. When all nodes share the same mining rate, the adversary and the honest nodes control αn and βn nodes, respectively. Let V'_k be the event that "the adversary mines k blocks at height $(\ell+1)$ before correct nodes mine a block at height $(\ell+1)$", where $k \leq \alpha n$. By *uniqueness*, each node can only mine a single block at height $(\ell+1)$, and the adversary can mine at most αn blocks at height $(\ell+1)$. Then, we have

$$\Pr\left[V'_0\right] = \frac{\beta}{e\alpha+\beta} \tag{1}$$

$$\Pr\left[V'_1\right] = \frac{e\alpha}{e\alpha+\beta}\cdot\frac{\beta}{e\alpha+\beta} \tag{2}$$

$$\Pr\left[V'_2\right] = \frac{\frac{\alpha n-1}{\alpha n}e\alpha}{\frac{\alpha n-1}{\alpha n}e\alpha+\beta}\cdot\frac{e\alpha}{e\alpha+\beta}\cdot\frac{\beta}{e\alpha+\beta} \tag{3}$$

$$\cdots \tag{4}$$

$$\Pr\left[V'_k\right] = \prod_{i=0}^{k-1}\frac{\frac{\alpha n-i}{\alpha n}e\alpha}{\frac{\alpha n-i}{\alpha n}e\alpha+\beta}\cdot\frac{\beta}{e\alpha+\beta} \tag{5}$$

$$= \prod_{i=0}^{k-1}\frac{(\alpha n-i)e}{(\alpha n-i)e+\beta n}\cdot\frac{\beta}{e\alpha+\beta} \tag{6}$$

When V'_k happens, the adversary's advantage is $\frac{k}{2^\kappa}$. Therefore, with probability $\Pr\left[V'_k\right]$, 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. The adversary's advantage in RandChain with *unique* SeqPoW is always smaller than in RandChain with *non-unique* SeqPoW. That is, for every k, $\Pr\left[V'_k\right] < \Pr\left[V_k\right]$. Given k, we have

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

$$= \frac{\prod_{i=0}^{k-1}\frac{(\alpha n-i)e}{(\alpha n-i)e+\beta n}}{\left(\frac{e\alpha}{e\alpha+\beta}\right)^k} \tag{8}$$

$$= \prod_{i=0}^{k-1}\frac{\frac{(\alpha n-i)e}{(\alpha n-i)e+\beta n}}{\frac{e\alpha}{e\alpha+\beta}} \tag{9}$$

As $0 \leq i < \alpha n$, it holds that $\frac{\Pr\left[V'_k\right]}{\Pr\left[V_k\right]} < 1$ for all k.

D Discussion

D.1 Frontrunning attacks

In DRBs, the adversary may launch the *frontrunning* attack, i.e., learn random outputs earlier than honest nodes. For collaborative DRBs, the adversary can collect inputs from all honest nodes, and solely aggregate them with its inputs to obtain the output. Honest nodes cannot learn the output before the adversary reveals its own inputs. Such adversary is known as *rushing* adversary [52]. In competitive DRBs, the adversary may mine a random output before all honest nodes, and withhold it until honest nodes mine another output. Compared to competitive DRBs, the adversary has less advantage in collaborative DRBs, as honest nodes can receive enough inputs to reconstruct the random output in a short time period.

Identifying and preventing frontrunning attacks is not trivial, due to the difficulty of distinguishing whether a message is delayed by the network or withheld by the adversary. We consider concrete study of defences against frontrunning attacks in DRBs as future work.

D.2 Achieving finality

There are various ways to achieve finality in RandChain. First, RandChain can follow the same approach of RANDAO [13] and Proofs-of-Delay [39], namely execute a VDF over each random output with time parameter longer than a block becoming stable. In addition, we consider two approaches in Byzantine consensus research, namely the quorum mechanism and herding-based consensus.

Quorum mechanism. Quorum [79] is the minimum number of votes that a proposal has to obtain for being agreed by nodes. A vote is usually a digital signature with some metadata, and a quorum of votes is called a *quorum certificate*. The quorum size is $n-f$, where n and f be the number of nodes and faulty nodes in the system, respectively. Achieving agreement in synchronous networks and partially synchronous networks require $n \geq 2f+1$ and $n \geq 3f+1$, respectively [58, 79].

RandChain can employ the quorum mechanism as follows. The the system should assume $n \geq 3f+1$. A node signs to vote a block. A node's view is represented as the latest block hash. Nodes proactively propagate their votes, i.e., signatures on blocks. A node finalises a block if collecting a quorum certificate, i.e., $\geq 2f+1$ votes, 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.

Herding-based consensus. *Herding* is a social phenomenon where people make choices according to other people's preference. Herding-based consensus allows nodes to decide proposals according to neighbour nodes' votes only, rather than a quorum of votes. Existing research [46, 90] shows that, herding-based consensus can achieve agreement with overwhelming probability in a short time period.

RandChain can employ 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 [90] as a finality layer for Bitcoin Cash [3].

D.3 Dynamic difficulty

PoW-based blockchains dynamically adjust difficulty parameters for stabilising the rate of generating new blocks. This is necessary when churn [94] is high. 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.