RandChain: A Scalable and Fair Decentralised Randomness Beacon

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

Haoyu Lin
FluiDEX & ZenGo X
chris.haoyul@gmail.com

Jiangshan Yu
Monash University
jiangshan.yu@monash.edu

***Abstract*—We propose RandChain, a Decentralised Randomness Beacon (DRB) that is the first to achieve both *scalability* (i.e., a large number of participants can join) and *fairness* (i.e., each participant controls comparable power on deciding random outputs). Unlike existing DRBs where participants are *collaborative*, i.e., aggregating their local entropy into a single output, participants in RandChain are *competitive*, i.e., competing with each other to generate the next output. The competitive design reduces the communication complexity from at least $O(n^2)$ to $O(n)$ without trusted party, breaking the scalability limit in existing DRBs.**

To build RandChain, we introduce *Sequential Proof-of-Work (SeqPoW)*, a cryptographic puzzle that takes a random and unpredictable number of sequential steps to solve. We implement RandChain and evaluate its performance on up to 1024 nodes, demonstrating its superiority (1.3 seconds per output with a constant bandwidth of 200KB/s per node) compared to state-of-the-art DRBs RandHerd (S&P'18) and HydRand (S&P'20).

I. Introduction

Randomness is a key building block for various protocols and applications. Decentralised Randomness Beacon (DRB) allows a set of participants to jointly generates random outputs periodically. It has been a promising approach to provide secure randomness. To support security-critical protocols and applications with high financial stake such as public blockchains [1]–[4] and voting protocols [5]–[7], DRBs have to be 1) *scalable*: even with a large number of participants, the DRB produces random outputs with an expected rate, and 2) *fair*: each participant controls comparable power on deciding random outputs. Without scalability, the DRB can be maintained only by a small set of participants. Without fairness, the DRB can be dominated by a small subset of powerful participants out of the entire set. When the DRB is dominated by a small set of participants, they can collude and manipulate the randomness in order to take advantage in protocols and applications supported by the DRB. However, designing a DRB that is both scalable and fair remains an open challenge.

Existing DRBs do not scale. Most DRB protocols are built from periodically executing a Distributed Randomness Generation (DRG) protocol, where participants contribute their local entropy and aggregate them into a single random output. Commonly used DRG protocols are based on threshold cryptosystems [3], [8], [9], Verifiable Random Functions (VRFs) [2], [10], [11], and/or Publicly Verifiable Secret Sharing (PVSS) [1], [12]–[16].

While DRG-based DRBs are fair given their "one-man-one-vote" design, they are not scalable, as they suffer from at least $O(n^2)$ communication complexity. DRG-based DRBs usually involve all-to-all broadcast primitives, leading to at least $O(n^2)$ communication complexity. To overcome the communication complexity bound, DRG-based DRBs have to employ a central point that relays messages. The central point is either a dealer [8], [9], [11], [13], [15] or a leader elected by a leader election protocol [1]–[3], [10], [12]. A dealer is either implemented as a trusted party or in a distributed manner which introduces extra communication overhead [17]. If the elected leader is corrupted, then it can bias random outputs by withholding messages and can compromise the liveness by sending messages to and advancing rounds for only a subset of participants [18], [19]. To tolerate corrupted leaders, the DRB has to employ an extra round synchronisation protocol [19], which allows participants to re-synchronise and replace the corrupted leader with a new leader to start a new round. However, round synchronisation protocols introduce extra communication complexity [18], [19] and/or increase latency [20].

The scalability crux: participants are collaborative. We attribute these limitations to the design that participants are *collaborative*: participants contribute their local inputs and aggregate them into a single output. The collaborative process ensures that no participant can fully control random outputs, making them hard to bias or predict. However, in order to collaborate, participants should continuously broadcast messages to and synchronise with each other. The former incurs at least $O(n^2)$ communication complexity, and the latter requires round synchronisation. All extra designs incorporated with DRG – e.g., using dealers [8], [9], [13], [15], leader election [1]–[3], [10]–[12], sharding [3], [12], cryptographic sortition [10], Byzantine consensus [10], [14], and erasure coding [13], [15] – 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.

Competitive DRBs: a new design space. To address the inherent limitations in the collaborative design, we consider a new design space for DRBs called *competitive DRBs*. Unlike existing DRBs where participants are collaborative, participants in competitive DRBs compete to solve cryptographic puzzles, whose solutions are unpredictable. The participant who first solves the puzzle becomes the leader, and broadcasts the puzzle solution to other participants. Upon a new puzzle solution, participants execute Nakamoto consensus [21] to agree on and append it to the sequence of puzzle solutions, ensuring consistency and liveness. A random output is extracted from each puzzle solution by using a Verifiable Delay Function (VDF) [22] which takes longer time than the puzzle solution becoming irreversible in the sequence. The time delay prevents the adversary from withholding its puzzle solution and biasing the random output to its own advantage.

RandChain: the first scalable and fair DRB. We propose RandChain, the first competitive DRB. RandChain works in permissioned settings identical to all existing DRBs, and is the first to achieve both scalability and fairness: it allows an unbounded number of participants to participate and restricts their voting power to be comparable. To achieve scalability, RandChain employs Nakamoto consensus [21] with linear communication complexity. To achieve fairness, RandChain realises *non-parallelisable mining* [23], where more processors do not give any advantage in solving a puzzle. As no existing primitive can provide *non-parallelisable mining*, we introduce *Sequential Proof-of-Work (SeqPoW)*, a cryptographic puzzle that takes a random and unpredictable number of sequential steps to solve. SeqPoW is also of independent interest in other protocols such as leader election and Proof-of-Stake (PoS)-based consensus. Our contributions are summarised as follows.

- We identity and formalise a new design space for DRBs, namely *competitive DRBs*, which break the scalability limit in existing DRB designs.
- As existing primitives lack the properties desired by the competitive DRBs (given the analysis in §III), we introduce and formalise the concept of SeqPoW that satisfies these properties. We provide two constructions based on VDFs [24], [25] and Sloth [26], and analyse their security and efficiency. We also discuss applications of SeqPoW in leader election and Proof-of-Stake (PoS)-based consensus (§IV).
- We provide RandChain as a concrete instantiation of competitive DRBs, and provide an analysis on its security and performance (§V).
- We provide an implementation of SeqPoW and RandChain and evaluate their performance (§VI). The implementation adds/changes about 4500 Rust lines of code (LoCs) on top of `parity-bitcoin` [27]. The evaluation results show that RandChain is indeed scalable and fair: with 1024 nodes, RandChain can produce a random output every 1.3 seconds (2.3x faster than RandHerd [12], 6.6x faster than HydRand [14] with 128 nodes); utilise constant bandwidth of about 200 KB/s per node (comparable with RandHerd with 1024 nodes and HydRand with 128 nodes); and provide nodes with comparable chance of producing random outputs.
- We establish a unified evaluation framework of DRBs, and compare RandChain with existing DRBs under this framework (§VII). Our comparison shows that RandChain is the only DRB that is secure, scalable and fair, without relying on any trusted third party.

We conclude the paper in §VIII, and provide additional details in the appendix. Appendix A provides preliminary definitions on VDFs. Appendix B provides formal definitions of SeqPoW and security proofs of SeqPoW. Appendix C provides formal security proofs for our SeqPoW constructions. Appendix D provides formal security proofs for RandChain. Appendix E provides the details of existing DRBs. Appendix F discusses three limitations and the corresponding resolutions of RandChain, namely the energy efficiency, churn tolerance and finality.

II. Model of DRBs

In this section, we define the model for DRBs, including the system model, correctness properties and performance metrics.

A. System model

System setting. We consider the system setting common in most DRBs [1]–[3], [8]–[16]. In particular, a DRB contains a set of n participants $\mathcal{P} = \{p_1,...,p_n\}$. Each participant $p_k \in \mathcal{P}$ has a pair of secret key sk_k and public key pk_k, and is uniquely identified by pk_k. Each participant is only directly connected to a subset of peers in the system. Participants jointly maintain a unique sequence of random outputs. Participants continuously execute the DRB protocol to agree on new random outputs and append them to the sequence.

Network model. Network model concerns the timing guarantee of messages delivery between participants. We consider a synchronous network where messages are delivered within a known finite time bound Δ.

Adversary model. The adversary controls $\alpha \in [0,1)$ voting power in the system. We assume $\alpha < \frac{1}{1+e}$ for RandChain, where e is Euler's number. The adversary is adaptive in the sense that it can corrupt any set of participants with at most α voting power at any time. The adversary can coordinate corrupted participants without delay; and can arbitrarily delay, drop, forge and modify messages from its corrupted participants.

B. Correctness properties

Consistency and liveness. Similar to consensus, DRBs should satisfy *consistency* and *liveness*. Consistency ensures that participants agree on a unique sequence of random outputs, and liveness ensures that participants produce new random outputs at an admissible rate. We adapt the *common-prefix* and *chain-growth* definitions from Nakamoto consensus protocols [28]–[31] rather than the *agreement* and *termination* definitions from BFT-style consensus protocols [32], as we consider a streamlined execution rather than a single-shot execution.

For consistency, we adapt the common-prefix definition in Nakamoto-style consensus where correct participants can only have different views on a certain number of last blocks. In DRBs, the consistency ensures that correct participants can only have different views on a certain number of last random outputs. Some randomness-based applications require RB to have *finality* [33], i.e., at any time, correct participants do not have conflicted views on the random output, which is equivalent to 0-consistency or *agreement* in Byzantine consensus [34].

Definition 1 (Υ-Consistency)**.** For any two correct participants at any time, their sequences can differ only in the last $\Upsilon \in \mathbb{N}$ random outputs.

For liveness, we adapt the chain-growth definition in Nakamoto-style consensus where correct participants produce blocks at a certain rate. In DRBs, the liveness ensures that correct participants produce random outputs at a certain rate. If the speed does not reach the lowest speed, then the DRB cannot satisfy the requirement of real-world applications.

Papers formalising a single-shot execution of DRBs refer liveness as *termination* [8], [10], [16] or *Guaranteed Output Delivery (G.O.D.)* [9], [13], [15], [35] where, for every round, a new random output will be produced.

Definition 2 ((t,τ)-Liveness)**.** For any time period of length t, every correct participant learns at least $t \cdot \tau$ new random outputs, where $t,\tau \in \mathbb{R}^+$.

Uniform distribution. Uniform distribution ensures that every random output in the DRB is statistically close to a uniformly random string.

Definition 3 (Uniform distribution)**.** Every random output is indistinguishable from a random string of the same length, except for negligible probability.

Unpredictability. Unpredictability ensures that the adversary cannot predict random outputs that have not been produced yet. Otherwise, if the adversary can predict future random outputs, then it can take advantage in randomness-based applications.

Definition 4 (Unpredictability)**.** Any adversary can only obtain negligible advantage on the following game. Assuming participants in the DRB agree on an ℓ-long sequence of random outputs. Before the $(\ell+1)$-th random output $R_{\ell+1}$ is produced, the adversary makes a guess $R'_{\ell+1}$ on $R_{\ell+1}$. The adversary's advantage is quantified as $\Pr\left[R'_{\ell+1} = R_{\ell+1}\right]$.

Unbiasibility. Unbiasibility ensures that the adversary cannot influence the produced random output to another value to its own advantage [9], [12], [14], [35]. Otherwise, if the adversary can bias random outputs, then it can take advantage in randomness-based applications. Unbiasibility can be achieved by the *output-independent-abort* property [36]: the adversary has to decide to proceed or abort the protocol before learning the protocol's outcome. In the context of an Υ-consistent DRB, output-independent-abort ensures that, participants learn a random output only after it becomes Υ-deep in a correct participant's view.

Definition 5 (Unbiasibility)**.** Assuming a DRB satisfies Υ-consistency, and participants in the DRB agree on an ℓ-long sequence of random outputs. The adversary learns the $(\ell + 1)$-th random output $R_{\ell+1}$ only after $(\ell + \Upsilon + 1)$ consecutive random outputs are recorded in the sequence of at least one correct participant, except for negligible probability.

C. Performance metrics

Communication complexity. Communication complexity is the total amount of communication required to complete a protocol [37]. In the context of DRBs, the communication complexity is quantified as the amount of communication (in bits) all participants take to generate a random output. For example, for a DRB that includes n participants and achieves $O(n)$ (aka linear) communication complexity, each participant handles a constant amount of communication for generating a random output, leading to the total amount of communication proportional to n. A protocol may have different communication complexity in the best-case and worst-case executions.

Latency. Latency is the time required to complete a protocol. In the context of DRBs, the latency is quantified as the time participants take to generate a random output. Similarly, a protocol may have different latencies in the best-case and worst-case executions. If the protocol's latency only depends on the actual network delay δ but not the delay upper bound Δ, then the protocol is *responsive* [38].

III. Design goals and strawman designs

In this section, we describe our two design goals, namely *scalability* and *fairness*, and analyse two strawman designs towards them. The analysis reveals the need for a cryptographic puzzle with two properties, namely *sequentiality* and *hardness*. As no existing puzzle achieves these two properties simultaneously, we are motivated to propose a new primitive named *Sequential Proof-of-Work (SeqPoW, §IV)* that satisfies both properties, allowing us to construct RandChain (§V).

A. Design goals: scalability and fairness

Our goal is to design a DRB that can serve security-critical protocols and applications with high financial stake, such as public blockchains and voting protocols. To ensure that the DRB can be trusted by such protocols and applications, we demand two additional requirements on the DRB atop the model in §II, namely *scalability* and *fairness*.

Scalability. Scalability specifies that the DRB can produce random outputs regularly even in the presence of a large set of n participants. Having a large set of participants reduces the trust needed on each participant, making the DRB more resilient to malicious parties. Otherwise, if the DRB is maintained by a small set of participants, then they can collude to bias and/or predict random outputs and thus take advantage in the randomness-based applications.

To produce random outputs regularly when n is large, the DRB has to minimise the communication complexity and latency. For communication complexity, $O(n)$ is considered scalable as each participant handles a constant amount of communication independent with n, while $O(n^2)$ is not as each node handles overwhelming communication overhead when n is large. For latency, demand it to be as small as possible.

Fairness. Fairness specifies that each participant controls comparable voting power on deciding random outputs, regardless of their financial stake or hardware resource. The voting power of a node is quantified as the amount of its contributed entropy in collaborative DRBs, and as its chance of producing the next block in competitive DRBs. Without fairness, few powerful participants among all participants will control the randomness generation process of the DRB. This is not desirable as the powerful participants can collude to compromise the DRB, similar to the scalability case.

Unlike DRG-based DRBs that satisfy fairness immediately given the "one-man-one-vote" nature, participants in competitive DRBs may have different voting power, leading to weak fairness. We define fairness as the maximum voting power difference among participants in the DRB. In the context of competitive DRBs, fairness is the maximum difference of nodes' chances of producing the next block.

Definition 6 (μ-Fairness)**.** Assuming all messages are delivered instantly and participants in a DRB agree on an ℓ-long sequence of random outputs. Let $X(p_k)$ be the event that participant p_k produces the $(\ell+1)$-th random output earlier than other participants. For any two participants p_i and p_j,

$$\mu = \min_{\forall i,j \in [n]} \frac{\Pr\left[X(p_i)\right]}{\Pr\left[X(p_j)\right]}$$

When $\mu = 1$, the DRB achieves ideal fairness and the network is fully decentralised, and vice versa when $\mu \to 0$. As a design goal, we demand μ to be as close to 1 as possible.

B. Strawman designs

We analyse two strawman designs towards the two goals. The analysis reveals the need for a cryptographic puzzle satisfying two properties, namely *sequentiality* and *hardness*. No existing puzzle satisfies both of them simultaneously.

Strawman#1: Nakamoto-style DRBs. The scalability goal requires the DRB to achieve $O(n)$ communication complexity. We have shown in §I that no existing DRB achieves it without a trusted third party, motivating us to propose the competitive DRB approach. A natural choice is building upon the Nakamoto-style consensus, where each participant solves a PoW puzzle to become the leader, and a random output is extracted from the PoW solution deterministically.

Such design satisfies *scalability* but not *fairness*, as participants with more mining hardware have more chance of mining blocks than others. To achieve fairness, the DRB has to prevent participants from investing more mining resource to take advantage in mining. A possible solution is the *non-parallelisable mining* [23], where a participant can only use a single processor for mining and cannot speed up mining by using multiple parallel processors. To realise non-parallelisable mining, the puzzle has to be *sequential*: it cannot be solved faster by using multiple parallel processors.

Strawman#2: Applying time-sensitive cryptography. Sequentiality has been formalised and achieved in time-sensitive cryptographic primitives. For example, Verifiable Delay Functions (VDFs) [22] enforce a parameterisable time delay on generating outputs and allow to verify outputs fast. Recent proposals [39], [40] apply VDFs to construct Nakamoto-style consensus: each participant derives a random output y from the latest system state, maps y to a random time parameter t in a designated interval, and solves a VDF with time parameter t. The first participant solving the VDF derives the next random output from its VDF output.

However, Nakamoto-style consensus with existing time-sensitive primitives achieves weaker fairness and consistency guarantee. All existing time-sensitive primitives have a fixed time delay. Nakamoto-style consensus with such puzzles is *locally predictable* [41]: given the input x, each participant can learn the time parameter t immediately, and thus can predict when it will propose the next random output. The adversary can apply such prediction to amplify its advantage in selfish mining [42] and double-spending [21], weakening the system's fairness and consistency guarantee, respectively [41].

To make the mining process unpredictable, the puzzle has to take a random and unpredictable number of attempts to solve. PoW satisfies such requirement by providing the *hardness* property [43]: upon each attempt on solving the puzzle, the solver has probability $\frac{1}{T}$ to solve the puzzle, where T is a hardness parameter. However, none of existing primitives satisfies both *sequentiality* and *hardness*.

IV. Sequential Proof-of-Work

In this section, we introduce *Sequential Proof-of-Work (SeqPoW)*, a PoW variant that satisfies both *sequentiality* and *hardness*. We formalise SeqPoW, provide two constructions, and analyse their security and efficiency.

A. Preliminaries on VDFs

Verifiable Delay Function (VDF) [22], [24], [25] allows a prover to evaluate an input, and produce a unique output deterministically with a succinct proof attesting the output's correctness. The evaluation process takes non-negligible and parameterisable time to execute, even with parallelism. Appendix A provides its formal definition. 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.
- $\mathsf{Eval}(pp, x, t) \to y$**:** On input pp, input x and time parameter $t \in \mathbb{N}^+$, produces output y.
- $\mathsf{Prove}(pp, x, y, t) \to \pi$**:** On input pp, x, y and t, outputs proof π.
- $\mathsf{Verify}(pp, x, y, \pi, t) \to \{0,1\}$**:** On input pp, x, y, π and t, outputs 1 if y is a correct evaluation, otherwise 0.

VDF satisfies three properties, namely *completeness* that all outputs from honest evaluations can pass the verification, *soundness* that all outputs from malicious evaluations cannot pass the verification, and σ-*sequentiality* that $\mathsf{Eval}(\cdot,\cdot,t)$ cannot be evaluated in less than time $\sigma(t)$ even with an unbounded amount of parallel processors. Sequentiality also implies *unpredictability*: before finishing $\mathsf{Eval}(\cdot)$, the probability of making a correct guess on its output is negligible.

VDFs are usually constructed from an iteratively sequential function (ISF) and a succinct proof attesting the ISF's execution results [24], [25]. ISF $f(t, x) = g^t(x)$ is a function that composes a sequential function $g(x)$ for t times. The fastest way of computing ISF $f(t, x)$ is to iterate $g(x)$ for t times, as $g(\cdot)$ is sequential. Squaring and squaring root over cyclic groups are two sequential functions with proven sequentiality [26], [44], [45]. Their repeated versions – repeated squaring [24], [25] and repeated squaring root [26] over cyclic groups – are two widely used ISFs.

ISF $f(\cdot)$ usually provides the *self-composability* property: 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* [46].

B. Basic idea of SeqPoW

SeqPoW is a cryptographic puzzle that takes a random and unpredictable number of sequential steps to solve. 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

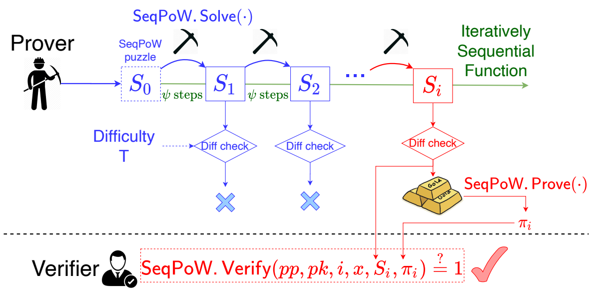


Figure 1: Sequential Proof-of-Work.

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.

Comparisons with relevant primitives (Table I). SeqPoW is the first primitive that satisfies both sequentiality and hardness, and therefore can be used for constructing RANDCHAIN. SeqPoW differs from VDFs and other time-sensitive cryptographic primitives, e.g., Timelock Puzzle (TLP) [47] and Proofs of Sequential Work (PoSW) [48], [49] in that, the SeqPoW prover iterates an ISF for a *randomised* (rather than given) number of times. In addition, compared to TLP, SeqPoW provides publicly verifiable outputs. Compared to PoSW, SeqPoW allows outputs to be unique. SeqPoW differs from PoW in that SeqPoW is sequential. SeqPoW differs from *memory-hard functions* (MHFs) [50]–[52] in that, SeqPoW is bottlenecked by the processor's frequency, whereas MHF is bottlenecked by the memory bandwidth.

Two concurrent works [39], [53] propose ways to randomise the number of iterations in VDFs, without formal treatment. 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 both of them cannot achieve.

Applications. Given the unpredictability and hardness properties, SeqPoW is of independent interest for other protocols. First, SeqPoW can improve the fairness of leader election protocols. Mining in PoW-based consensus can be seen as a way of electing leaders: given a set of participants, the first participant 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. In §V-C, we show that compared to parallelisable PoW, SeqPoW-based leader election achieves better fairness.

Second, SeqPoW can improve the fault tolerance capacity of Proof-of-Stake (PoS)-based consensus. In Proof-of-Stake (PoS)-based consensus [54], each participant's chance of mining a block is in proportion to its *stake*, e.g, the participant's balance. Most PoS-based consensus protocols [1], [2], [55]–[57] select block proposers in a *predictable* [41], [58] way, thus are vulnerable to various prediction-based attacks and tolerate less Byzantine mining power [41], [58] than PoW-based consensus, as analysed in §III. To make PoS-based consensus unpredictable, one can randomise the process of selecting block proposers. SeqPoW can provide such functionality: each participant solves a SeqPoW with its identity, the last block, and the difficulty parameter inversely proportional to its stake as input, and the first participant solving its SeqPoW becomes the block proposer. A concurrent and independent work [53] provides a concrete protocol following the similar idea.

Table I: 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.	✓	✓

C. Definition

We provide formal definitions of SeqPoW in Appendix B. The syntax of SeqPoW is as follows.

$\mathsf{Setup}(\lambda,\psi,T)\to pp$**:** On input security parameter λ, step $\psi\in\mathbb{N}^+$ and difficulty $T\in[1,\infty)$, outputs public parameter pp.

$\mathsf{Gen}(pp)\to(sk,pk)$**:** A probabilistic function, which on input pp, produces a secret key sk and a public key pk.

$\mathsf{Init}(pp,sk,x)\to(S_0,\pi_0)$**:** On input pp, sk and input x, outputs initial solution S_0 and proof π_0. Some constructions may use pk 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 pp, sk and i-th solution S_i, outputs $(i+1)$-th solution S_{i+1} and result $b_{i+1}\in\{0,1\}$.

$\mathsf{Prove}(pp,sk,i,x,S_i)\to\pi_i$**:** On input pp, sk, i, x and S_i, outputs proof π_i.

$\mathsf{Verify}(pp,pk,i,x,S_i,\pi_i)\to\{0,1\}$**:** On input pp, pk, i, x, S_i and π_i, outputs 1 if S_i is a valid solution, otherwise 0.

A tuple (pp,sk,i,x,S_i,π_i) is honest if (S_i,π_i) is generated from evaluating $\mathsf{Solve}(\mathsf{p}, sk, S_{i-1})$ and $\mathsf{Prove}(pp, sk, i, x, S_i)$ honestly, and is valid if it is honest and b_i associated to S_i is 1. SeqPoW satisfies *completeness*, *soundness*, *hardness* and *sequentiality*. Completeness ensures that for every honest tuple $(pp, sk, i, x, S_i, \pi_i)$, $\mathsf{Verify}(pp, pk, i, x, S_i, \pi_i) = 1$. Soundness ensures that for every non-honest tuple $(pp, sk, i, x, S_i, \pi_i)$, $\mathsf{Verify}(pp,pk,i,x,S_i,\pi_i)=0$. Hardness ensures that given difficulty parameter T, each $\mathsf{Solve}(\cdot)$ attempt has the success rate of $\frac{1}{T}$, implying that the number of sequential steps towards solving the puzzle is *uniformly random with a mathematical expectation of* T. Sequentiality ensures that even with parallel processors, the fastest way of computing S_i is incrementing $\mathsf{Solve}(\cdot)$ for i times, which takes time $\sigma(i\cdot\psi)$. Similar to VDFs [22], sequentiality in SeqPoW also implies unpredictability: without i sequential $\mathsf{Solve}(\cdot)$ invocations towards S_i, the probability of making a correct guess on S_i is negligible.

SeqPoW also has an optional property *uniqueness*, by which each SeqPoW puzzle only has a single valid solution S_i. Before finding a valid solution S_i each $\mathsf{Solve}(\cdot)$ attempt follows the *hardness* definition, but after finding S_i no further $\mathsf{Solve}(\cdot)$ attempt returns a valid solution.

D. Constructions

Let $H:\{0,1\}^*\to\{0,1\}^\kappa$ be a cryptographic hash function; G be a cyclic group; $H_G:\{0,1\}^*\to G$ be a function mapping

Setup(λ, ψ, T)

$pp_{\mathsf{VDF}} = (G, g) \leftarrow \mathsf{VDF.Setup}(\lambda)$
$pp \leftarrow (pp_{\mathsf{VDF}}, \psi, T)$
return pp

Gen(pp)

$(G, g, \psi, T) \leftarrow pp$
Sample random $sk \in \mathbb{N}$
$pk \leftarrow g^{sk} \in G$
return (sk, pk)

Init(pp, pk, x)

$(G, g, \psi, T) \leftarrow pp$
$S_0 \leftarrow H_G(pk \| x)$
return S_0

Solve(pp, pk, S_i)

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

Prove(pp, pk, i, x, S_i)

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

Verify$(pp, pk, i, x, S_i, \pi_i)$

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

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

Setup(λ, ψ, T)

$pp \leftarrow (G, g, \psi, T)$
return pp

Gen(pp)

$(G, g, \psi, T) \leftarrow pp$
Sample random $sk \in \mathbb{N}$
$pk \leftarrow g^{sk} \in G$
return (sk, pk)

Init(pp, pk, x)

$(G, g, \psi, T) \leftarrow pp$
$S_0 \leftarrow H_G(pk \| x)$
return S_0

Solve(pp, pk, S_i)

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

Prove(pp, pk, i, x, S_i)

return $\perp$

Verify$(pp, pk, i, x, S_i, \pi_i)$

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

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

Figure 2: Construction of SeqPoW.

an arbitrary string to an element on G; g be a generator of G; 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 $S_i = \mathsf{VDF.Eval}(pp, S_{i-1}, \psi)$ 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 $S_i = \mathsf{VDF.Eval}(pp, S_{i-1}, \psi) = \mathsf{VDF.Eval}^i(pp, S_0, \psi) = \mathsf{VDF.Eval}(pp, S_0, i \cdot \psi)$ when VDF is *self-composable*. 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 [26], a *slow-timed* hash function. In Sloth, the prover calculates the square root (on a cyclic group G) over the input for t times to get the output. The verifier calculates the square over the output for t times to recover the input and checks if the input is same as the one from the prover. Although the verification is linear (and thus do not meet the VDF definition [22]), 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.

E. Security and efficiency analysis

Security. Appendix C provides full security proofs of the SeqPoW constructions. The completeness and soundness are immediate from Sloth and VDFs' completeness, soundness and self-composability. By *pseudorandomness* of $H_G(\cdot)$ and *sequentiality* of Sloth and VDFs, $\mathsf{Solve}(\cdot)$ outputs unpredictable solutions. As $H(\cdot)$ is modelled as a random oracle and $\mathsf{Solve}(\cdot)$ produces an unpredictable solution, the probability that the solution satisfies the difficulty is $\frac{1}{T}$, leading to *hardness*. The sequentiality and self-composability of Sloth and VDFs guarantee the sequentiality of the SeqPoW constructions.

VDFs can be instantiated with any cyclic group, including the RSA group that requires a trusted setup and the class group without such requirement. The trusted setup is usually conducted by a trusted party or a multi-party protocol [59], [60].

Efficiency (Table II). $\mathsf{SeqPoW}_{\mathsf{VDF}}$ 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 $\mathsf{Solve}(\cdot)$ executes ψ steps of the ISF, and the prover attempts $\mathsf{Solve}(\cdot)$ for T times on average to find a valid solution. $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ generate and verify proofs of ψT consecutive modular squaring operations, respectively.

We analyse $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with both Wesolowski's VDF (Wes19) [25] and Pietrzak's VDF (Pie19) [24] without optimisation/parallelisation techniques [24], [25], [61]. According to the existing analysis [62], 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 [25] and with Pie19 [24] are 32 and 1280 Bytes,

Table II: Efficiency of two SeqPoW constructions.

	Solve(·)	Prove(·)	Verify(·)	Proof size (Bytes)
$\mathsf{SeqPoW_{VDF}}$ + Wes19	$O(\psi)$	$O(\psi T)$	$O(\log \psi T)$	s
$\mathsf{SeqPoW_{VDF}}$ + Pie19	$O(\psi)$	$O(\sqrt{\psi T} \log \psi T)$	$O(\log \psi T)$	$s \log_2 \psi T$
$\mathsf{SeqPoW_{Sloth}}$	$O(\psi)$	0	$O(\psi T)$	0

respectively. $\mathsf{SeqPoW_{Sloth}}$ has the verification complexity of $O(\psi T)$ and uses the solution itself to represent the proof.

V. RandChain: DRB from SeqPoW

In this section, we build the RandChain protocol. Figure 3 and 4 provides the intuition and full specification of RandChain, respectively. In RandChain, participants jointly maintain a sequence of random outputs as a blockchain, where each random output is derived from a block (§V-A). Specifically, participants agree on a unique blockchain by executing the Nakamoto consensus, which ensures consistency, liveness, and scalability in synchronous networks (§V-B). RandChain composes Nakamoto consensus with our proposed SeqPoW puzzle to achieve *non-parallelisable mining*, guaranteeing the fairness (§V-C). Each random output is extracted from a block by using a Verifiable Delay Function (VDF) so that the random output is learned only after the block becomes irreversible in the blockchain, guaranteeing the uniform distribution, unpredictability and unbiasibility (§V-D). Appendix D provides the proofs of all correctness properties for RandChain.

A. DRB structure

Each participant p_k locally maintains a ledger $\mathcal{C}_k$ formed as a directed acyclic graph (DAG) of blocks. Following Nakamoto consensus $\mathsf{mainChain}(\cdot)$, p_k selects the longest fork in $\mathcal{C}_k$ as the main chain $\mathcal{MC}_k$. If there are multiple longest forks at the same length, p_k chooses the one it receives first. $\mathcal{MC}_k$ is formed as a blockchain, i.e., a totally ordered sequence of blocks. We denote $|\mathcal{MC}_k|$ as the length of $\mathcal{MC}_k$.

Each block B is of the format $B = (h^-, h, i, S, pk, \pi)$, 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^-. Each block B is identified by its ID $B.h = H(B.pk \| B.S)$, and points to a previous block B^- by setting $B.h^- = B^-.h$. One can extract a random output $B.R$ from each block B by using a deterministic function $\mathsf{randomOutput}(\cdot)$, which we will describe later in (§V-D).

B. Synchronising and agreeing on blocks

Each participant p_k keeps running $\mathsf{SyncRoutine}(\cdot)$ to synchronise its local ledger $\mathcal{C}_k$ with other participants. Specifically, participant p_k keeps receiving blocks from other participants, verifying them, and adding valid blocks to its local ledger $\mathcal{C}_k$. Participant p_k keeps tracking the main chain $\mathcal{MC}_k$ following Nakamoto consensus $\mathsf{mainChain}(\cdot)$, and executes the mining routine $\mathsf{MineRoutine}(\cdot)$ on $\mathcal{MC}_k$.

RandChain achieves consistency and liveness in synchronous networks, and can tolerate an adversary with mining power $\alpha < \frac{1}{1+e}$, where e is Euler's number. The intuition of the proof is to deduct the consistency and liveness analysis of RandChain to that of Proof-of-Stake (PoS)-based Nakamoto consensus, which is proven [58], [63] to tolerate an adversary with $\frac{1}{1+e}$ mining power, rather than the well-known "honest majority" result.

RandChain does not achieve 0-consistency (aka *finality*). One can deploy existing finality layer mechanisms [64]–[66] into RandChain. In Appendix F3 we analyse two approaches of adding *finality* to RandChain.

RandChain inherits communication complexity and latency guarantees from Nakamoto consensus. The communication complexity is $O(n)$ as the only communication is the leader broadcasting blocks. The latency is $t_{\mathsf{block}} + \delta$, where t_{block} is the parameterised block interval and δ is the actual network delay. Thus, RandChain achieves scalability.

C. Non-parallelisable mining

RandChain employs the SeqPoW puzzle for the mining routine $\mathsf{MineRoutine}(\cdot)$. Specifically, participant p_k keeps solving the latest SeqPoW puzzle S derived from $\mathsf{SeqPoW.Init}(pp, sk_k, B^-.h)$, 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, participant p_k keeps executing $\mathsf{SeqPoW.Solve}(\cdot)$ until finding a valid solution. With a valid solution, participant p_k constructs a block B, and appends B to $\mathcal{MC}_k$.

RandChain achieves non-parallelisable mining, leading to μ-fairness with $\mu > \frac{1}{5}$ in practice where every node at least preserves a commodity processor with 2∼3 GHz frequency. Each participant has a unique input deriving a unique SeqPoW puzzle, so can only use a single processor for mining. By SeqPoW's sequentiality, to accelerate solving SeqPoW puzzles, one can only increase the processor's frequency. While commodity processors usually achieve 2∼3 GHz frequency, the most advanced processor achieves the frequency of 8.723 GHz [67], which is hard to improve further due to the voltage limit [68]. Hence, the fastest processor can mine at most five times faster than normal processors, leading to $\mu > \frac{1}{5}$. The limited speedup is evidenced by the recent *VDF Alliance FPGA Contest* [69]–[71], where optimised VDF implementations are about four times faster than the baseline implementation.

The adversary can weaken μ to $\geq \frac{\mu}{2}$ by *selfish mining*, i.e., withholding and publishing blocks adaptively w.r.t. blocks from honest miners [42]. To defend against selfish mining attacks, one can deploy existing countermeasures [72]–[74].

D. Extracting a random output from a block

Given block B, $\mathsf{randomOutput}(\cdot)$ extracts the random output $B.R$ via $\mathsf{VDF.Eval}(pp, B.pk \| B.S, t_{\mathsf{VDF}})$ and computes proof $B.\pi_R$ via $\mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, B.pk \| B.S, B.R, t_{\mathsf{VDF}})$, where pp_{VDF} and t_{VDF} are VDF's public parameter and time parameter known to all participants, respectively. The time parameter t_{VDF} is chosen so that finishing $\mathsf{Eval}(\cdot)$ takes longer than participants extending $(\Upsilon + 1)$ blocks for a Υ-consistent RandChain.

The time delay in $\mathsf{randomOutput}(\cdot)$ ensures the unbiasibility of RandChain. If the random output is extracted from a block instantly, then the adversary can withhold its block if it does not like the extracted random output, compromising

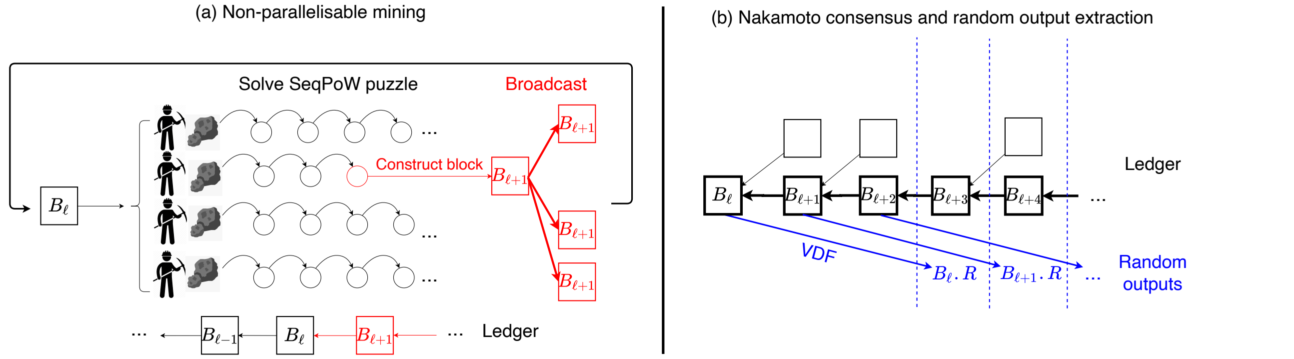


Figure 3: The RandChain protocol. **(a)** Upon block B_ℓ, each participant keeps solving its own SeqPoW puzzle. The participant who first solves its SeqPoW puzzle (the red one) proposes the next block $B_{\ell+1}$ (in red). $B_{\ell+1}$ piggybacks B_ℓ by including B_ℓ's ID, i.e., $B_{\ell+1}.h^- = B_\ell.h$. **(b)** Each participant maintains a local ledger formed as a DAG of blocks. It considers the longest fork of the DAG as the main chain and mines over it. For each block B_ℓ, the random output $B_\ell.R$ is extracted by a VDF that takes longer than nodes extending $(\Upsilon+1)$ blocks (in this case $\Upsilon=1$) so that $B_\ell.R$ is learned only after B_ℓ becomes irreversible.

$\mathsf{mainChain}(\mathcal{C}_k)$

1: $\mathcal{MC}_k \leftarrow \perp$
2: **foreach** fork $\mathcal{C}_k^*$ of $\mathcal{C}_k$ **do**
3:   **if** $|\mathcal{C}_k^*| > |\mathcal{MC}_k|$ **then** $\mathcal{MC}_k \leftarrow \mathcal{C}_k^*$
4: **return** $\mathcal{MC}_k$

$\mathsf{randomOutput}(pp, B, t_{\mathsf{VDF}})$

1: $(pp_{\mathsf{VDF}}, \cdot, \cdot) \leftarrow pp$
2: $B.R \leftarrow \mathsf{VDF.Eval}(pp_{\mathsf{VDF}}, B.pk \| B.S, t_{\mathsf{VDF}})$
3: $B.\pi_R \leftarrow \mathsf{VDF.Prove}(pp_{\mathsf{VDF}}, B.pk \| B.S,$
4: $B.R, t_{\mathsf{VDF}})$
5: **return** $B.R, B.\pi_R$

$\mathsf{MainProcedure}(pp, sk_k, pk_k)$

1: Synchronise ledger as $\mathcal{C}_k$
// The following two lines read/modify C_k concurrently
2: $\mathsf{MineRoutine}(pp, sk_k, pk_k, \mathcal{C}_k)$ **in a thread**
3: $\mathsf{SyncRoutine}(pp, \mathcal{C}_k)$ **in a thread**

$\mathsf{MineRoutine}(pp, sk_k, pk_k, \mathcal{C}_k)$

1: **repeat**
2:   $\mathcal{MC}_k \leftarrow \mathsf{mainChain}(\mathcal{C}_k)$
3:   $B^- \leftarrow \mathcal{MC}_k[-1]$
4:   $i \leftarrow 0$
5:   $S \leftarrow \mathsf{SeqPoW.Init}(pp, sk_k, B^-.h)$
6:   **repeat**
7:     **if** $\mathcal{C}_k$ is updated by $\mathsf{SyncRoutine}(\cdot)$ **then**
8:       Repeat line 2-5
9:     $S, b \leftarrow \mathsf{SeqPoW.Solve}(pp, sk_k, S)$
10:     $i += 1$
11:     **if** $b = 1$ **then break**
12:   $h \leftarrow H(pk_k \| S)$
13:   $\pi \leftarrow \mathsf{SeqPoW.Prove}(pp, sk, i, B^-.h, S)$
14:   $B \leftarrow (B^-.h, h, i, S, pk_k, \pi)$
15:   Append B to $\mathcal{MC}_k$ after B^-
16:   Propagate B

$\mathsf{SyncRoutine}(pp, \mathcal{C}_k)$

1: **repeat**
2:   Wait for a new block as B
3:   $(h^-, h, i, S, pk, \pi) \leftarrow B$
4:   // B should point to an existing block
5:   **if** $\nexists B^- \in \mathcal{C}_k : B^-.h = h^-$ **then** discard B
6:   // B should have a valid ID h
7:   **if** $h \neq H(pk \| S)$ **then** discard B
8:   // B should include a valid puzzle solution
9:   **if** $\mathsf{SeqPoW.Verify}(pp, pk, i, h^-, S, \pi) = 0$ **then**
10:     Discard B
11:   Append B to $\mathcal{C}_k$ after block B^- where $B^-.h = h^-$
12:   Propagate B

Figure 4: Full specification of RandChain.

the unbiasibility. With the time delay of extending $(\Upsilon+1)$ blocks, the adversary has to decide whether to broadcast or withhold its mined block before learning the random output. After learning the random output, the block either becomes irreversible (if the adversary broadcasts the block) or cannot be accepted anymore (if the adversary withholds the block).

RandChain satisfies uniform distribution: a λ-bit random string can be extracted from a block, where λ is SeqPoW and VDF's security parameter. RandChain satisfies unpredictability, as the sequentiality of SeqPoW and VDF implies their outputs are unpredictable as analysed in §IV-A.

VI. Implementation and Evaluation

We implement SeqPoW and RandChain, and evaluate their performance. The evaluation shows that all SeqPoW constructions are practical and RandChain is indeed scalable and fair. Specifically, on a cluster of 1024 nodes (each as a participant), RandChain can produce a random output every 1.3 seconds (2.3x faster than RandHerd [12] with 1024 nodes, 6.6x faster than HydRand [14] with 128 nodes); utilise constant bandwidth of about 200 KB/s per node (comparable with RandHerd with 1024 nodes and HydRand with 128 nodes); and provide nodes with comparable chance of producing random outputs. We will make all code and experimental data publicly accessible after the paper is published.

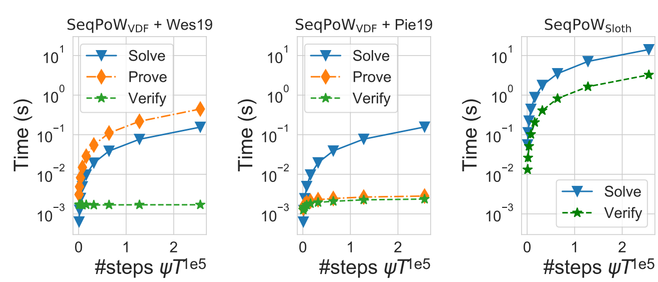


Figure 5: Evaluation of SeqPoW constructions.

A. SeqPoW: benchmarks

Implementation. We implement the SeqPoW constructions in Rust. We use `rug` [75] 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 [24]–[26].

Experimental setting. For each function, we test ψT up to 256000, where ψ is the step parameter and T is the difficulty. The code for benchmarking is based on the `cargo-bench` [76] and `criterion` [77] benchmarking suites. We specify O3-level optimisation for compilation, and sample ten executions for each benchmarked function with a unique set of parameters. All experiments were conducted on a machine with a 2.2 GHz 6-Core Intel Core i7 Processor and a 16 GB 2400 MHz DDR4 RAM.

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 $\mathsf{Solve}(\cdot)$, making it less suitable for instantiating RANDCHAIN. For $\mathsf{SeqPoW}_{\mathsf{VDF}}$ with Pie19, $\mathsf{Prove}(\cdot)$ and $\mathsf{Verify}(\cdot)$ take negligible time compared to $\mathsf{Solve}(\cdot)$. For $\mathsf{SeqPoW}_{\mathsf{Sloth}}$, $\mathsf{Solve}(\cdot)$ is about five times slower than $\mathsf{Verify}(\cdot)$. Although this is far from the theoretically optimal value, i.e., $\log_2 |G| = 1024$ in our setting [78], the verification overhead is acceptable for the use case where random outputs are not generated frequently.

B. RANDCHAIN: *end-to-end evaluation*

We implement RANDCHAIN and evaluate it on computer clusters regarding the following metrics:

- **Block propagation delay (BPD)** is the time taken for the majority of nodes to receive a block (§VI-B2).
- **Block size** is the size of a block. It varies w.r.t. *blocktime* (i.e., the average time interval between two blocks) as the VDF proof size increases with the time parameter. We also estimate the network overhead of propagating blocks amortised by time (§VI-B3).
- **Network overhead** is the average bandwidth utilisation, i.e., the average amount of data transferred in a time unit, of a node (§VI-B4).
- **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 (§VI-B5).

Among the metrics, the former three are the empirical results of scalability (where BPD infers latency and the rest two infer network overhead); and decentralisation is the empirical result of fairness. We also compare RANDCHAIN with state-of-the-art DRBs that have experimental results, including RandHerd [12] and HydRand [14]. Table III summarises the evaluation results and comparison with RandHerd and HydRand.

1) **Implementation and experimental settings:** We implement RANDCHAIN based on `Parity-bitcoin` [27], a Bitcoin implementation in Rust. Each node plays as a participant of RANDCHAIN. It uses `RocksDB` [79] for storage, and Bitcoin's `Wire` protocol [80] for the P2P protocol stack. We adapt the ledger structure, SeqPoW and relevant message types to RANDCHAIN's setting specified in §V. Given the evaluation result in §VI-A, we use Pie19 for instantiating SeqPoW and extracting random outputs from blocks. 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` [81] for monitoring system resource utilisation.

Table III: Experimental settings and results.

	Experimental setting				Experimental results	
	#nodes	#machines	Deployment	Network	Latency	Net. overhead
RandHerd [12]	1024	32	Datacenter	Simulated	3 sec	200 KB/s
HydRand [14]	128	128	Worldwide	Real	8.6 sec	180∼310 KB/s
RANDCHAIN	1024	128	Worldwide	Real	1.3 sec	200 KB/s

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 deploy $\{16,32,64,128\}$ `t2.micro` EC2 instances (1 GB RAM, one CPU core and 60-80 Mbit/s network bandwidth) in 13 regions around the globe[1], 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 [80]. When a node starts, it randomly connects to 8 peers, accepts connections from other peers, and starts gossiping messages with them. As mining is not allowed in cloud computing platforms, we simulate $\mathsf{SeqPoW.Solve}(\cdot)$ rather than actually executing it. For our SeqPoW implementation, the `t2.micro` EC2 instance can do squaring operations in $\mathsf{SeqPoW.Solve}(\cdot)$ for 233868 times per second on average. We test blocktime 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 metrics. The total size of logs is 1.74 GB.

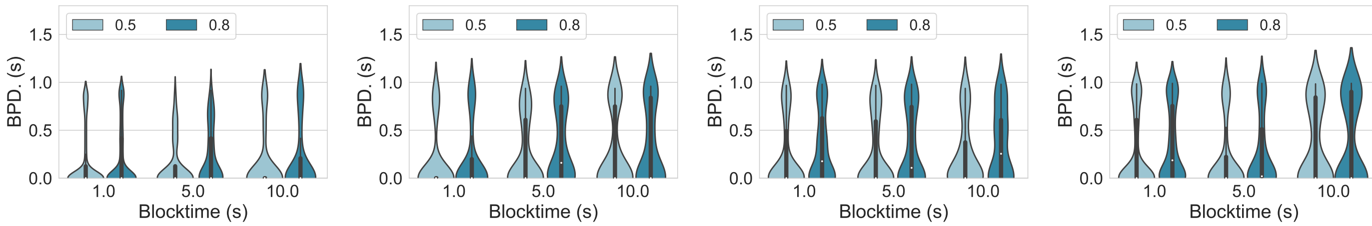
2) **Block propagation delay (BPD):** Figure 6 shows the distribution of BPD for different sizes of clusters. First, with the increasing number of nodes (from 128 to 1024), the BPD never exceeds 1.3 seconds. In other words, the system can produce a random output every 1.3 seconds, which is 2.3x faster than RandHerd ($\sim$3 seconds on a 1024-node cluster) and 6.6x faster than HydRand ($\sim$8.6 seconds on a 128-node cluster). This is expected given the linear communication complexity.

Second, BPD is usually either less than 0.4 second or more than 0.6 second, but is hardly in-between values. This implies that a block can reach most nodes within 2 hops: 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.

Third, the average BPD increases slowly with more nodes. This is consistent with other linear protocols [82]. In linear protocols, the average BPD is proportional to the average number of intermediate nodes of two random nodes. In Bitcoin's setting where each node connects to k random peers, the network is structured as an Erdos-Renyi random graph [83], in which two random nodes have $O(\frac{\log n}{\log k})$ intermediate nodes on average.

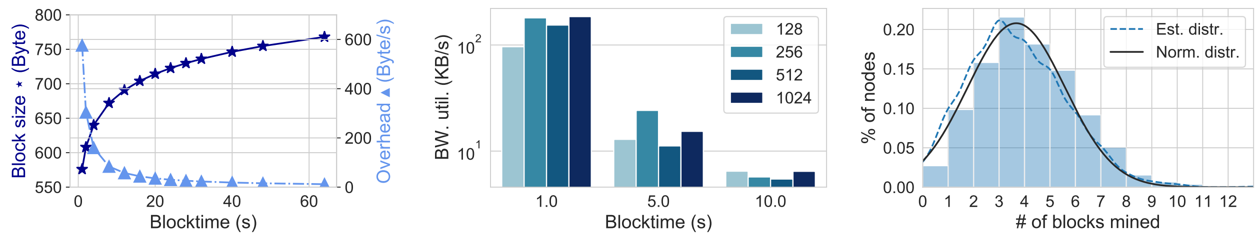
Last, BPD increases when blocks are produced more

[1]The regions include North Virginia, North California, Oregon, Ohio, Canada, Mumbai, Seoul, Sydney, Tokyo, Singapore, Ireland, Sao Paulo, London, and Frankfurt.



(a) 128 nodes. (b) 256 nodes. (c) 512 nodes. (d) 1024 nodes.

Figure 6: Distribution of block propagation delay (BPD), represented as *violin plots*. The light blue and dark blue parts indicate the distribution of BPD when blocks are propagated to 50% and 80% of nodes, respectively.



(a) Block size and its estimated overhead. (b) Network overhead. (c) Decentralisation.

Figure 7: Block size, network overhead and decentralisation. **(a)** Block size and estimated network overhead between two nodes amortised by time v.s. blocktime. The dark blue increasing line is on the block size and the light blue decreasing line is on the overhead. **(b)** Network overhead, quantified as the bandwidth utilisation of each node with different blocktimes. **(c)** Decentralisation level, visualised as the number of blocks produced by distinct nodes. The blue and black lines are the kernel density estimation and the closest normal distribution, respectively.

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.

3) **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 the time taken to find a solution and the number of iterations executed in a time unit. Recall that the computer can do squaring operations for 233868 times per second. Given blocktime t, the SeqPoW proof size is $s \cdot \log_2(233868 \cdot t) \approx s \cdot (18 + \log_2 t)$, and the network overhead between two nodes amortised by time is $\frac{s \cdot (18+\log_2 t)}{t}$. Figure 7a shows the relationship between blocktime, block size and network overhead. When blocktime is $\{1,5,10\}$ seconds and $s = 32$ Bytes, the block size is about $\{576,1336,1912\}$ Bytes, and the amortised network overhead is about $\{576, 267, 191\}$ Bytes/s. When blocktime is 60 seconds (the setting of Drand [84] and the NIST randomness beacon [85]), the block size is about 3402 Bytes, and the amortised network overhead is about 57 Bytes/s.

4) **Network overhead:** Figure 7b shows the bandwidth utilisation result. It shows that RANDCHAIN utilises less bandwidth compared to RandHerd and HydRand: even with blocktime of 1 second, each node utilises ∼200KB/s bandwidth per second, which is comparable with RandHerd (∼200KB/s on a 1024-node cluster) and HydRand (180∼310KB/s on a 128-node cluster). The bandwidth utilisation remains stable with more nodes, as RANDCHAIN is linear. These two results are as expected since RANDCHAIN is linear. The inbound and outbound bandwidths are identical, as the input (i.e., the last block) and the output (i.e., the new block) are identical in terms of size, leading to identical bandwidth utilisation. With longer blocktime, the node requires less bandwidth, as nodes send and receive blocks less frequently.

5) **Decentralisation:** Figure 7c shows the distribution of nodes w.r.t. the number of blocks they produce on the main chain, in the experiment with 1024 nodes and the blocktime of 1 second. The kernel estimated distribution is close to the normal distribution, meaning that nodes have comparable chance of producing blocks, similar to RandHerd and HydRand that are "one-man-one-vote". The result is consistent with our experimental setting where nodes use the same processors.

VII. 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 secure, scalable and fair simultaneously, without relying on any trusted party.

A. Overview of existing DRBs

DRG-based DRBs. Participants execute the single-shot Distributed Randomness Generation (DRG) protocol periodically. DRG can be constructed from various cryptographic primitives, such as threshold cryptosystems [3], [8], [9], Verifiable Random Functions (VRFs) [2], [10], [11], and Publicly Verifiable Secret Sharing (PVSS) [1], [12]–[16]. To relax the network model assumptions, reduce the communication complexity and/or improve the fault tolerance capacity,

Table IV: Comparison of RANDCHAIN with existing DRBs.

	Protocol		System model			Correctness							Performance	
	Name	Primitives	Net. model	Trust	Fault tol. cap.	Consistency	Liveness	Fairness	Uniform dist.	Unpredictability	Unbiasibility	Pub. ver.	Comm. compl.	Latency
DRG-based DRBs	Cachin et al. [8]	Thr. Sig.	Async.	Dealer‡	1/3	✓	✓	1	✓	✓	✓	✓	$O(n^3)$	$O(\delta)$
	HERB [9]	Homo. Thr. Enc.	Part. sync.	Dealer‡	1/3	✓	✓	1	✓	✓	✓	✓	$O(n)$	$O(\delta)$
	Dfinity [3]	VRF + Thr. Sig.	Sync.	-	1/3	✓	✓	1	✓	✓	✗†	✓	$O(cn)^{\diamond¶}$	$O(\Delta)\sim\infty^{¶}$
	Ouro. Praos [2]	VRF	Part. sync.	-	1/2	✓	✓	1	✓	✓	✗†	✓	$O(n)^{¶}$	$O(\Delta)\sim\infty^{¶}$
	GLOW [11]	VRF	Sync.	-	1/3	✓	✓	1	✓	✓	✗†	✓	$O(n)^{¶}$	$O(\delta)\sim\infty^{¶}$
	Algorand [10]	VRF	Sync.	-	1/3	✓	✓	1	✓	✓	✗†	✓	$O(cn)^{\diamond¶}$	$O(\Delta)\sim\infty^{¶}$
	Ouroboros [1]	PVSS	Sync.	-	1/2	✓	✓	1	✓	✓	✓	✓	$O(n^3)$	$O(\Delta)$
	SCRAPE [13]	PVSS	Part. sync.	Dealer‡	1/2	✓	✓	1	✓	✓	✓	✓	$O(n^3)$	$O(\delta)$
	RandShare [12]	PVSS	Async.	-	1/3	✓	✓	1	✓	✓	✓	✓	$O(n^3)$	$O(\delta)$
	RandHound [12]	PVSS	Sync.	-	1/3	✓	✓	1	✓	✓	✗†	✓	$O(c^2n)^{\diamond¶}$	$O(\Delta)\sim\infty^{¶}$
	RandHerd [12]	PVSS	Sync.	Dealer‡	1/3	✓	✓	1	✓	✓	✓	✓	$O(c^2 \log n)^{\diamond}$	$O(\delta)$
	HydRand [14]	PVSS	Sync.	-	1/3	✓	✓	1	✓	✓	✓	✓	$O(n^2)$	$O(\Delta)$
	Albatross [15]	PVSS	Part. sync.	Dealer‡	1/2	✓	✓	1	✓	✓	✓	✓	$O(n)$	$O(\delta)$
	Kogias et al. [16]	HAVSS	Async.	-	1/3	✓	✓	1	✓	✓	✓	✓	$O(n^4)$	$O(\delta)$
SC-based DRBs	RanDAO [86]	VDF	Part. sync.^{x}	Blockchain^x	1/2^x	✓	✓	1	✓	✓	✓	✓	$O(n)$	$t_{\text{block}}+\delta$
	Yakira et al. [87]	Escrow-DKG	Part. sync.^{x}	Blockchain^x	1/3^x	✓	✓	1	✓	✓	✓	✓	$O(n)$	$t_{\text{block}}+\delta$
ISF-based DRBs	Unicorn [26]	Sloth	Async.	Setup	(n-1)/n	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✓	✓	$O(n)$	Any $+\delta$
	Ephraim et al. [88]	Continuous VDF	Async.	Setup	(n-1)/n	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✓	✓	$O(n)$	Any $+\delta$
	RandRunner [35]	Trapdoor VDF	Async.	Setup	1/2	✓	✓	1	✓	✗⋆	✓	✓	$O(n)\sim O(n^2)$	Any $+\delta$
DRBs from ext. entr.	Clark et al. [89]	Rand. extractors	Async.	Data src.	(n-1)/n	✓	✓	-	✓	✓	✓	✗II	$O(n)$	Any $+\delta$
	Bonneau et al. [90]	Rand. extractors	Async.^{x}	Blockchain^x	(n-1)/n^x	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✓	✗II	$O(n)$	$t_{\text{block}}+\delta$
	Bünz et al. [91]	Proof-of-Delay	Async.^{x}	Blockchain^x	(n-1)/n^x	✓	✓	$\rightarrow 0^{\otimes}$	✓	✓	✓	✗II	$O(n)$	$t_{\text{block}}+\delta$
This work	RANDCHAIN	SeqPoW + Nak. consensus	Sync.	-	1/(1+e)	✓	✓	$>\frac{1}{5}$	✓	✓	✓	✓	$O(n)$	$t_{\text{block}}+\delta$

‡ The analysis assumes the dealer is a trusted third party. While the dealer can be implemented in a distributed manner [17], it introduces extra communication overhead.
† The corrupted leader can withhold the random output and enforce participants to start a new round, as analysed in [14], [15].
⋄ We use c to denote the size of shards in Dfinity [3], RandHound and RandHerd [12], and the size of the committee in Algorand [10].
¶ The corrupted leader can send the random output and advance the round for a subset of participants, so that participants are executing different rounds. The DRB requires an extra round synchronisation protocol that suffers from either exponential latency [20] or worst-case communication complexity of $\geq O(n^2)$ [18], [19].
⋆ The adversary can always corrupt leaders and produce random outputs efficiently via the trapdoor.
II Entropy generated by the external source is not verifiable.
⊗ In Unicorn and Ephraim et al., the participant with the fastest processor can always propose random outputs earlier than other participants. In DRBs with PoW-based blockchains as external entropy, mining can be accelerated by using parallelism. Both cases weaken the fairness degree to near zero.
x These DRBs are usually built upon public blockchains. When considering the public blockchain as a part of the DRB, the system model will also respect that of the public blockchain. For example, the DRB may be built upon Ethereum, which requires synchronous networks and fault tolerance capacity $\alpha < \frac{1}{2}$.

these DRBs usually rely on a centralised dealer [8], [9], [13], [15] and/or combine techniques such as leader election [1]–[3], [10]–[12], sharding [3], [12], cryptographic sortition [10], Byzantine consensus [10], [14] and erasure coding [13], [15].

Other types. In Smart contract (SC)-based DRBs [86], [87], [92], participants submit their inputs to an external smart contract, which combines them to a single random output. In DRBs from external entropy, participants periodically extract randomness from real-world entropy, e.g., real-time financial data [89] and public blockchains [90], [91], [93]. In Iteratively sequential function (ISF)-based DRBs [26], [35], [88], participants use intermediate outputs of an ISF as random outputs, and use succinct proofs for the ISF to make outputs verifiable.

B. Evaluation framework for DRBs

We extend our model in §II to build an evaluation framework for DRBs. Apart from synchronous networks in §II-A, the framework additionally considers *partially synchrony* [94] where messages are delivered within a known finite time-bound Δ after an unknown Global Stabilisation Time (GST) and *asynchrony* where messages are delivered without a known time bound. Apart from system model aspects in §II-A, the framework also concerns trust assumptions that some proposals assume in order to guarantee correctness properties. Apart from the correctness properties in §II-B, the framework also concerns *fairness* and *public verifiability*: whether a random output is publicly verifiable.

C. Evaluation

Table IV summarises the evaluation results. Let Δ be the network latency bound in the synchrony period, δ be the actual network delay, and GST be the global stabilisation time.

System model. Most DRG-based DRBs employ synchronous leader election protocols, except for the following proposals. Cachin et al., RandShare and Kogias et al. employ randomised common coin techniques to achieve asynchrony. Ouroboros Praos allows "empty slots" (where participants produce no block) when no leader is elected before GST, and guarantees an elected leader after GST, leading to partial synchrony. HERB, SCRAPE, and Albatross employ a dealer who relays all messages and proceeds the protocol whenever receiving enough shares, which is guaranteed after GST, leading to partial synchrony. These DRBs have to trust the dealer, otherwise a corrupted dealer can selectively multicast messages to allow a subset of nodes to predict random outputs, or withhold messages to bias random outputs. While the dealer can be implemented in a distributed manner [17], it introduces extra communication overhead. SC-based DRBs rely on a permissionless blockchain to achieve partial synchrony. The blockchain is assumed to be trusted, otherwise a corrupted blockchain can censor transactions to bias random outputs, which is known as the Miner Extractable Value (MEV) issue [95]. ISF-based DRBs and DRBs from external entropy proceed as long as a single participant is honestly executing the ISF or sampling the entropy, except for RandRunner which requires a reliable broadcast with fault tolerance degree $\alpha < \frac{1}{2}$.

ISFs require a trusted setup, otherwise the adversary who previously knows the seed can learn random outputs earlier than other participants. The entropy source has to be trusted, otherwise the adversary can manipulate the entropy and bias random outputs. In DRBs based on permissionless blockchains, the blockchains usually employ Nakamoto-style consensus and thus assume synchronous networks. If the blockchain-based DRBs allow nodes to run a blockchain protocol on their own, then it incurs more communication overhead.

Correctness properties. All DRG-based DRBs achieve consistency and liveness. Note that DRG-based DRBs define liveness as termination (where correct participants eventually learn the random output at the end of each round), and our evaluation of DRG-based DRBs follows such definition. All DRBs achieve the ideal fairness, i.e., $\mu = 1$, except for DRBs from PoW-based blockchains [90], [91], Unicorn [26], Ephraim et al. [88] and RANDCHAIN. DRBs from PoW-based blockchains allow accelerating mining by parallelism. For Unicorn and Ephraim et al., the participant with the fastest processor can always propose random outputs earlier than other participants. Both cases weaken the fairness degree to near zero. RANDCHAIN achieves $\mu > \frac{1}{5}$ by making the mining process unpredictable [41], [58] and non-parallelisable, as analysed in §V-B-V-C. All DRBs satisfy uniform distribution and unpredictability, except for RandRunner [35] where the adversary can keep corrupting leaders and computing random outputs efficiently via the trapdoor, breaking unpredictability. In Dfinity, Ouroboros Praos, GLOW, Algorand and RandHound, the corrupted leader can withhold the random output and enforce participants to start a new round, breaking the *unbiasibility*, as analysed in [14], [15]. DRBs from external entropy do not satisfy public verifiability, as the external entropy is not publicly verifiable.

Performance metrics. In all dealer-less DRG-based DRBs, either the leader election, view change or PVSS protocol requires the all-to-all broadcast operations, leading to at least $O(n^2)$ communication complexity. To reduce communication complexity, HERB, RandHerd and Albatross employ a dealer to relay messages; GLOW allows participants to determine a unique leader locally given the last random output; Dfinity, RandHound and RandHerd apply sharding techniques to divide participants into different shards; Algorand samples a subset of participants to execute the protocol; and SC-based DRBs rely on a smart contract that relays all messages. RandRunner is linear in the best case, but requires reliable broadcasts with $O(n^2)$ communication complexity in the worst case. The other two ISF-based DRBs and DRBs from external entropy achieve $O(n)$ communication complexity.

Asynchronous DRG-based DRBs terminate within $O(\delta)$, as asynchronous networks do not have Δ. In HERB, SCRAPE, RandHerd and Albatross, the random output is produced once the dealer receives enough shares, leading to the latency of $O(\delta)$. In Ouroboros and HydRand, the leader election terminates in $O(\Delta)$. In GLOW, when the leader is correct, the latency is $O(\delta)$. When the leader is corrupted, then it can deliver random outputs and advance the round for a subset of participants, so that participants will execute different rounds. To re-synchronise the round, nodes have to execute an extra round synchronisation protocol with either exponential latency (by using the time doubling mechanisms [20]) or at least $O(n^2)$ worst-case communication complexity (by using the broadcast-based mechanisms [18], [19]). In Dfinity, Ouroboros Praos, Algorand, and RandHound, the leader election terminates within $O(\Delta)$, and a corrupted leader can cause the round synchronisation issue similar in GLOW. SC-based DRBs and DRBs from blockchain entropy achieve the latency of the parameterised block interval t_{block} plus δ. ISF-based DRBs and DRBs from other entropy can achieve any latency plus δ, according to the frequency of sampling intermediate outputs and entropy, respectively.

VIII. CONCLUSION

In this paper, we identified a new design space of Decentralised Randomness Beacon (DRB) protocols where participants are competitive, and constructed the first DRB protocol RANDCHAIN that belongs to this class. RANDCHAIN overcomes the scalability limit in the existing DRB design where participants are collaborative. The theoretical analysis and experimental evaluation show that RANDCHAIN is secure, scalable and fair without any trusted party.

REFERENCES

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

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

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

[4] J. Benet and N. Greco, “Filecoin: A decentralized storage network”, *Protocol Labs*, pp. 1–36, 2018.

[5] E. Rescorla, “On the security of election audits with low entropy randomness.”, *EVT/WOTE*, vol. 89, no. 92, p. 114, 2009.

[6] B. Adida, “Helios: Web-based open-audit voting.”, in *USENIX security symposium*, vol. 17, 2008, pp. 335–348.

[7] B. Adida, O. de Marneffe, O. Pereira, and J.-J. Quisquater, “Election a university president using open-audit voting”, in *3rd Int. Workshop on Security and Electronic Voting. Grenoble, France*, 2009.

[8] C. Cachin, K. Kursawe, and V. Shoup, “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography”, *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[9] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, “Homomorphic Encryption Random Beacon.”, *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 1320, 2019.

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

[11] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong, “Fully distributed verifiable random functions and their application to decentralised random beacons”, in *European Symposium on Security and Privacy*, 2021.

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

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

[14] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, “HydRand: Efficient Continuous Distributed Randomness”, in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 32–48.

[15] I. Cascudo and B. David, “ALBATROSS: publicly AttestabLe BATched Randomness based On Secret Sharing”, in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2020, pp. 311–341.

[16] E. Kokoris Kogias, D. Malkhi, and A. 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*, 2020, pp. 1751–1767.
[17] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems", in *International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 1999, pp. 295–310.
[18] O. Naor, M. Baudet, D. Malkhi, and A. Spiegelman, "Cogsworth: Byzantine View Synchronization", *arXiv preprint arXiv:1909.05204*, 2019.
[19] O. Naor and I. 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*, ser. LIPIcs, vol. 179, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, 26:1–26:17.
[20] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness", in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.
[21] S. Nakamoto *et al.*, "Bitcoin: A peer-to-peer electronic cash system", 2008.
[22] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions", in *Annual international cryptology conference*, Springer, 2018, pp. 757–788.
[23] "[ANSWERED] Why is bitcoin proof of work parallelizable ?", https://bitcointalk.org/index.php?topic=46739.0.
[24] K. Pietrzak, "Simple verifiable delay functions", in *10th innovations in theoretical computer science conference (itcs 2019)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.
[25] B. Wesolowski, "Efficient verifiable delay functions", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2019, pp. 379–407.
[26] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx.", *IACR Cryptol. ePrint Arch.*, vol. 2015, p. 366, 2015.
[27] "paritytech/parity-bitcoin: The Parity Bitcoin client", https://github.com/paritytech/parity-bitcoin.
[28] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2017, pp. 643–673.
[29] J. A. Garay, A. Kiayias, and N. Leonardos, "Full Analysis of Nakamoto Consensus in Bounded-Delay Networks.", *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 277, 2020.
[30] L. Ren, "Analysis of Nakamoto Consensus.", *IACR Cryptol. ePrint Arch.*, vol. 2019, p. 943, 2019.
[31] B. Y. Chan and E. Shi, "Streamlet: Textbook Streamlined Blockchains.", in *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, ACM, 2020, pp. 1–11.
[32] M. Castro, B. Liskov, *et al.*, "Practical Byzantine fault tolerance", in *OSDI*, vol. 99, 1999, pp. 173–186.
[33] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication", in *International workshop on open problems in network security*, Springer, 2015, pp. 112–125.
[34] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.
[35] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl, "RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness", in *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*, The Internet Society, 2021.
[36] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner, "Craft: Composable randomness beacons and output-independent abort mpc from time", 2020.
[37] A. C.-C. Yao, "Some complexity questions related to distributive computing (preliminary report)", in *Proceedings of the eleventh annual ACM symposium on Theory of computing*, 1979, pp. 209–213.
[38] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model", in *31st International Symposium on Distributed Computing (DISC 2017)*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.
[39] J. Long and R. Wei, "Nakamoto Consensus with Verifiable Delay Puzzle", *arXiv preprint arXiv:1908.06394*, 2019.
[40] B. Cohen and K. Pietrzak, *The chia network blockchain*, 2019.
[41] J. Brown-Cohen, A. Narayanan, A. Psomas, and S. M. Weinberg, "Formal barriers to longest-chain proof-of-stake protocols", in *Proceedings of the 2019 ACM Conference on Economics and Computation*, 2019, pp. 459–473.
[42] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable", in *International conference on financial cryptography and data security*, Springer, 2014, pp. 436–454.
[43] M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan, "Proofs of work from worst-case assumptions", in *Annual International Cryptology Conference*, Springer, 2018, pp. 789–819.
[44] B. Wesolowski and R. Williams, "Lower bounds for the depth of modular squaring", 2020.
[45] J. Katz, J. Loss, and J. Xu, "On the security of time-lock puzzles and timed commitments", in *Theory of Cryptography Conference*, Springer, 2020, pp. 390–413.
[46] N. Döttling, S. Garg, G. Malavolta, and P. N. Vasudevan, "Tight verifiable delay functions", in *International Conference on Security and Cryptography for Networks*, Springer, 2020, pp. 65–84.
[47] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto", 1996.
[48] M. Mahmoody, T. Moran, and S. Vadhan, "Publicly verifiable proofs of sequential work", in *Proceedings of the 4th conference on Innovations in Theoretical Computer Science*, 2013, pp. 373–388.
[49] B. Cohen and K. Pietrzak, "Simple proofs of sequential work", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2018, pp. 451–467.
[50] C. Percival, *Stronger key derivation via sequential memory-hard functions*, 2009.
[51] A. Biryukov, D. Dinu, and D. 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)*, IEEE, 2016, pp. 292–302.
[52] J. Alwen, B. Chen, K. Pietrzak, L. Reyzin, and S. Tessaro, "Scrypt is maximally memory-hard", in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2017, pp. 33–62.
[53] S. Deb, S. Kannan, and D. Tse, *PoSAT: Proof-of-Work Availability and Unpredictability, without the Work*, 2020. arXiv: 2010.08154 [cs.CR].
[54] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake", *self-published paper, August*, vol. 19, p. 1, 2012.
[55] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability", in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 913–930.
[56] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake", in *International Conference on Financial Cryptography and Data Security*, Springer, 2019, pp. 23–41.
[57] L. Fan and H.-S. Zhou, "A scalable proof-of-stake blockchain in the open setting (or, how to mimic nakamoto's design via proof-of-stake)", Cryptology ePrint Archive, Report 2017/656, Tech. Rep., 2017.
[58] V. Bagaria, A. Dembo, S. Kannan, S. Oh, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Proof-of-Stake Longest Chain Protocols: Security vs Predictability", *arXiv preprint arXiv:1910.02218*, 2019.
[59] M. Chen, C. Hazay, Y. Ishai, Y. Kashnikov, D. Micciancio, T. Riviere, A. Shelat, M. Venkitasubramaniam, and R. Wang, "Diogenes: Lightweight scalable rsa modulus generation with a dishonest majority", in *2021 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2021, pp. 590–607.
[60] M. Chen, R. Cohen, J. Doerner, Y. Kondi, E. Lee, S. Rosefield, and A. 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*, ser. Lecture Notes in Computer Science, vol. 12172, Springer, 2020, pp. 64–93.
[61] V. Attias, L. Vigneri, and V. Dimitrov, "Implementation study of two verifiable delay functions", in *2nd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.
[62] D. Boneh, B. Bünz, and B. Fisch, "A Survey of Two Verifiable Delay Functions.", *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 712, 2018.

[63] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. 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*, ACM, 2020, pp. 859–878.

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

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

[66] T. Dinsdale-Young, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Afgjort: A partially synchronous finality layer for blockchains", in *International Conference on Security and Cryptography for Networks*, Springer, 2020, pp. 24–44.

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

[68] "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.

[69] "supranational/vdf-fpga-round1-results", https://github.com/supranational/vdf-fpga-round1-results.

[70] "supranational/vdf-fpga-round2-results", https://github.com/supranational/vdf-fpga-round2-results.

[71] "supranational/vdf-fpga-round3-results", https://github.com/supranational/vdf-fpga-round3-results.

[72] K. A. Negy, P. R. Rizun, and E. G. Sirer, "Selfish mining re-examined", in *International Conference on Financial Cryptography and Data Security*, Springer, 2020, pp. 61–78.

[73] E. 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*, Springer, 2014, pp. 161–162.

[74] R. Zhang and B. Preneel, "Publish or perish: A backward-compatible defense against selfish mining in bitcoin", in *Cryptographers' Track at the RSA Conference*, Springer, 2017, pp. 277–292.

[75] "crates/rug", https://crates.io/crates/rug.

[76] "cargo-bench", https://doc.rust-lang.org/cargo/commands/cargo-bench.html.

[77] "criterion.rs", https://github.com/bheisler/criterion.rs.

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

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

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

[81] "dstat-real/dstat: Versatile resource statistics tool", https://github.com/dstat-real/dstat.

[82] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple", in *2020 IEEE Symposium on Security and Privacy (SP)*, IEEE, 2020, pp. 90–105.

[83] P. Erdős and A. Rényi, "On the evolution of random graphs", *Publ. Math. Inst. Hung. Acad. Sci*, vol. 5, no. 1, pp. 17–60, 1960.

[84] "Drand - Distributed Randomness Beacon.", https://drand.love/.

[85] J. Kelsey, L. T. Brandão, R. Peralta, and H. Booth, "A reference for randomness beacons: Format and protocol version 2", National Institute of Standards and Technology, Tech. Rep., 2019.

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

[87] D. Yakira, A. Asayag, I. Grayevsky, and I. Keidar, "Economically Viable Randomness", *arXiv preprint arXiv:2007.03531*, 2020.

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

[89] J. Clark and U. Hengartner, "On the Use of Financial Data as a Random Beacon.", *EVT/WOTE*, vol. 89, 2010.

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

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

[92] D. Yakira, I. Grayevsky, and A. Asayag, "Rational Threshold Cryptosystems", *arXiv preprint arXiv:1901.01148*, 2019.

[93] M. Andrychowicz and S. Dziembowski, "Distributed Cryptography Based on the Proofs of Work.", *IACR Cryptol. ePrint Arch.*, vol. 2014, p. 796, 2014.

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

[95] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability", in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 566–583.

[96] J. Großschädl, A. Szekely, and S. Tillich, "The energy cost of cryptographic key establishment in wireless sensor networks", in *Proceedings of the 2nd ACM symposium on Information, computer and communications security*, 2007, pp. 380–382.

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

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

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

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

APPENDIX

A. Definition of VDF

We present the formal definition of Verifiable Delay Functions (VDFs) [22], [24], [25].

Definition 7 (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, output y, and time parameter t, outputs proof π.

$\mathsf{Verify}(pp, x, y, \pi, t) \to \{0,1\}$**:** On input pp, x, y, π and t, outputs 1 if y is a correct evaluation, 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 controls any polynomial amount of processors and runs in less than time $\sigma(t)$,

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

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

Definition 8 (Self-Composability)**.** A VDF (Setup, Eval, Prove, Verify) satisfies self-composability if for all λ, x, (t_1,t_2),

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

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

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

B. *Definition of SeqPoW*

We present the formal definition of Sequential Proof-of-Work (SeqPoW).

Definition 9 (Sequential Proof-of-Work (SeqPoW))**.** A Sequential Proof-of-Work SeqPoW is a tuple of algorithms

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

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

$\mathsf{Gen}(pp) \rightarrow (sk,pk)$**:** A probabilistic function, which on input public parameter pp, produces a secret key $sk \in \mathcal{X}$ and a public key $pk \in \mathcal{X}$.

$\mathsf{Init}(pp,sk,x) \rightarrow (S_0,\pi_0)$**:** On input public parameter pp, secret key sk, and input $x \in \mathcal{X}$, outputs initial solution $S_0 \in \mathcal{Y}$ and proof π_0. Some constructions may use public key pk as input rather than sk. This also applies to $\mathsf{Solve}(\cdot)$ and $\mathsf{Prove}(\cdot)$.

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

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

$\mathsf{Verify}(pp,pk,i,x,S_i,\pi_i) \rightarrow \{0,1\}$**:** On input pp, pk, i, x, S_i, and π_i, outputs 1 if S_i is a valid solution, otherwise 0.

We define honest tuples and valid tuples as follows.

Definition 10 (Honest tuple)**.** A tuple (pp,sk,i,x,S_i,π_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 11 (Valid tuple)**.** For all λ, ψ, T, and $pp \leftarrow \mathsf{Setup}(\lambda,\psi,T)$, a tuple (pp,sk,i,x,S_i,π_i) is (λ,ψ,T)-valid if

- (pp,sk,i,x,S_i,π_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*.

Definition 12 (Completeness)**.** A SeqPoW scheme satisfies completeness if for all λ,ψ,T,

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

Definition 13 (Soundness)**.** A SeqPoW scheme satisfies soundness if for all λ,ψ,T,

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

Definition 14 (Hardness)**.** A SeqPoW scheme satisfies hardness if for all (λ,ψ,T)-honest tuple (pp,sk,i,x,S_i,π_i),

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

Definition 15 (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,

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

Definition 16 (Uniqueness (optional))**.** 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.

C. *Security proofs for SeqPoW*

We formally prove the security guarantee of two SeqPoW constructions.

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

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

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

■

Lemma 3. $\mathsf{SeqPoW_{VDF}}$ *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

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

By *soundness* and Lemma 1, if (y,y^+,π^+,ψ) is generated by $\mathcal{A}$, $\mathsf{VDF.Verify}(\cdot)$ will return 0. As hash functions are deterministic, if $S_i > \frac{2^\kappa}{T}$, difficulty check will return 0. Thus, if (pp,sk,i,x,S_i,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

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

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 $(1-\epsilon)$ probability where ϵ is negligible, 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 (λ,ψ,T)-valid tuples $(pp, sk, i, x, S_i, \pi_i)$ and $(pp,\ sk,\ i,\ x,\ S_i,\ \pi_i)$ where $j < i$. According to $\mathsf{SeqPoW_{Sloth}.Solve}(\cdot)$, 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 $\mathsf{SeqPoW_{Sloth}.Verify}(\cdot)$ will pass. However, in the for loop of $\mathsf{SeqPoW_{Sloth}.Verify}(\cdot)$, if S_i is valid, then verification of S_j will fail. Then, $\mathsf{SeqPoW_{Sloth}.Verify}(\cdot)$ returns 0, which contradicts the assumption. ■

D. Security proofs for RandChain

We prove RandChain (denoted as $\Pi_{RandChain}$ throughout the analysis) achieves all correctness properties defined in §II when the network is synchronous and the adversary's voting power $\alpha<\frac{1}{1+e}$. Let $\beta=1-\alpha$ be the voting power of correct nodes.

Consistency and liveness. We prove that $\Pi_{RandChain}$ achieves the same consistency and liveness guarantee as PoS-based Nakamoto consensus (termed as *Nakamoto-PoS*) protocol Π_{Nak}^{PoS} [58].

Lemma 8. $\Pi_{RandChain}$ *satisfies consistency and liveness when the network is synchronous and the adversary's voting power $\alpha<\frac{1}{1+e}$.*

Proof: Π_{Nak}^{PoS} realises Nakamoto consensus Π_{Nak} based on the PoS puzzle Π_{PoS}, and is proven [53], [63] to achieve consistency and liveness when the network is synchronous and the adversary's voting power $\alpha<\frac{1}{1+e}$. When composed with Π_{Nak}, Π_{PoS} and Π_{SeqPoW} provide the same guarantee on generating blocks: the adversary has an mining rate α on every existing block, which is independent with each other. Thus, $\Pi_{RandChain}$ achieves the same security guarantee as Π_{Nak}^{PoS}, i.e., achieves consistency and liveness when the network is synchronous and $\alpha<\frac{1}{1+e}$. ■

Uniform distribution. We prove that each block derives a λ-bit uniformly distributed random string, where λ is the security parameter of SeqPoW and VDF.

Lemma 9. $\Pi_{RandChain}$ *satisfies uniform distribution.*

Proof: Each random output $B.R$ of $\Pi_{RandChain}$ is extracted from a block B via the VDF. By VDF's sequentiality, each VDF output contains non-negligible entropy that is unpredictable. A hash function can be applied to the VDF output to extract a λ-bit uniform random string [22]. ■

Unpredictability. In the prediction game, the $(\ell+1)$-th block is either produced by correct participants or the adversary's participants. If the adversary's advantage is negligible for both cases, then $\Pi_{RandChain}$ satisfies *unpredictability*. When the $(\ell+1)$-th block is produced by correct participants, the adversary's best strategy is guessing, leading to negligible advantage. When the $(\ell+1)$-th block is produced by the adversary's participants, the adversary's best strategy is to produce as many blocks as possible before receiving a new block from the correct participants. First, consider $\Pi_{RandChain}$ using SeqPoW without *uniqueness*.

Lemma 10. *Assuming all messages are delivered instantly and participants agree on a blockchain of length ℓ. If the $(\ell+1)$-th block is produced by a correct participant, then the adversary's advantage on the prediction game is $\frac{1}{2^\kappa}$.*

If the next output is produced by the adversary's participants, the adversary's best strategy is to produce as many blocks as possible before receiving a new block from the correct participants. First, consider $\Pi_{RandChain}$ using SeqPoW without *uniqueness.*

Lemma 11. *Consider $\Pi_{RandChain}$ using SeqPoW without uniqueness. Assuming all messages are delivered instantly* *and participants 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 [58], [63]. Thus, the probability that the adversary and correct participants 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 consistency, and $\alpha+\beta=1$.

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

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

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 participants mine a block, leading to the advantage of $\frac{k}{2^\kappa}$. ■

Then, we analyse $\Pi_{RandChain}$ using SeqPoW with *uniqueness*. Without the loss of generality, we assume all participants share the same mining rate.

Lemma 12. *Consider $\Pi_{RandChain}$ using SeqPoW with uniqueness. Assuming all participants share the same mining rate, all messages are delivered instantly and participants 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 $\Pr[V'_k]$, where*

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

Proof: Similar to Lemma 11, the adversary and the correct participants control mining rate $\frac{e\alpha}{e\alpha+\beta}$ and $\frac{\beta}{e\alpha+\beta}$, respectively. When all participants share the same mining rate, the adversary and the correct participants control αn and βn participants, respectively. Let V'_k be the event that "the adversary mines k blocks at height $(\ell+1)$ before correct participants mine a block at height $(\ell+1)$", where $k \leq \alpha n$. By *uniqueness*, each participant 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

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

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

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

$$\dots \tag{4}$$

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

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

When V'_k happens, the adversary's advantage is $\frac{k}{2^\kappa}$. Therefore, with probability $\Pr[V'_k]$, the adversary mines k blocks before correct participants mine a block, leading to the advantage of $\frac{k}{2^\kappa}$ (where $k \leq \alpha n$). ■

Remark 1. The adversary's advantage in $\Pi_{RandChain}$ with *unique* SeqPoW is always smaller than in $\Pi_{RandChain}$ with *non-unique* SeqPoW. That is, for every k, $\Pr[V'_k] < \Pr[V_k]$. Given k, we have

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

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

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

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

Unbiasibility. $\Pi_{RandChain}$ achieves unbiasibility by realising the *output-independent-abort* notion [36]. With a VDF with time parameter long than a new block becoming irreversible, the adversary has to decide whether to broadcast or withhold a block before learning the random output.

Lemma 13. $\Pi_{RandChain}$ *satisfies unbiasibility.*

Proof: The proof is by contradiction. Assuming participants agree on an ℓ-long blockchain, and the adversary learns the random output $B_{\ell+1}.R$ in the $(\ell+1)$-th block $B_{\ell+1}$ when every correct participant's main chain contains less than $(\ell+\Upsilon+1)$ blocks, where Υ is the consistency degree. Recall that extracting $B_{\ell+1}.R$ from $B_{\ell+1}$ is by evaluating a VDF with a time parameter longer than participants extending $(\Upsilon+1)$ blocks on the blockchain. By VDF's sequentiality, to learn $B_{\ell+1}.R$, the adversary has to learn $B_{\ell+1}$ first. By SeqPoW's sequentiality, the adversary can learn $B_{\ell+1}$ only after learning its previous block B_ℓ, which is already agreed by participants. Thus, the adversary extracts $B_{\ell+1}.R$ from $B_{\ell+1}$ only after a correct participant grows its main chain from ℓ blocks to $(\ell+\Upsilon+1)$ blocks if the adversary withholds $B_{\ell+1}$, and to $(\ell+\Upsilon+2)$ blocks if the adversary publishes

$B_{\ell+1}$, leading to a contradiction to the assumption. Therefore, $\Pi_{RandChain}$ achieves unbiasibility. ■

E. Details of existing DRBs

We categorise existing DRBs into four types, namely Distributed Randomness Generation (DRG)-based DRBs, Smart contract (SC)-based DRBs, DRBs from external entropy, and Iteratively sequential function (ISF)-based DRBs.

DRG-based DRBs. Participants execute the single-shot DRG protocol periodically. DRG can be constructed from various cryptographic primitives, such as threshold cryptosystems [3], [8], [9], Verifiable Random Functions (VRFs) [2], [10], [11], and Publicly Verifiable Secret Sharing (PVSS) [1], [12]–[16].

Cachin et al. [8], Dfinity [3] are constructed from threshold signatures, and HERB [9] is constructed from homomorphic threshold encryption. Cachin et al. and HERB assume a trusted dealer who relays all messages, and Dfinity allows participants to decide a leader locally according to the last random output. To work in asynchronous networks, Cachin et al. employs common coin techniques where participants share a unique input (e.g., the round number). To reduce communication complexity, Dfinity divides participants into different shards and avoids all-to-all broadcast operations.

Ouroboros Praos [2], Algorand [10] and GLOW [11] are constructed from VRFs. In these designs, participants first execute a leader election protocol to determine a leader. In Ouroboros Praos and Algorand, the leader executes VRF over the current blockchain state to produce the random output solely. In GLOW, participants jointly execute the Distributed VRF (DVRF) over the last random output to produce the current random output, and all messages are relayed by the leader.

In PVSS-based DRG protocols (Ouroboros [1], RandHound/RandHerd [12], SCRAPE [12], HydRand [14], Albatross [15], Kogias et al. [16]), each participant chooses a local random input and uses PVSS to share it to other participants, aggregates all received shares on different random inputs into a single one, broadcasts aggregated shares, and aggregating received shares again to recover the final random output. To tolerate corrupted participants, HydRand, RandHound and RandHerd enforce participants to execute consensus to agree on a subset of shares; and SCRAPE and Albatross use erasure codes to encode shares. To reduce communication complexity, RandHound and RandHerd apply sharding techniques similar to Dfinity; and SCRAPE and Albatross employ a trusted dealer relaying messages. To tolerate network asynchrony, Kogias et al. employs an asynchronous PVSS variant.

SC-based DRBs. Participants employ a smart contract as the bulletin board. RANDAO [86] 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. [87] construct SC-based DRBs from Escrow Distributed Key Generation (DKG) [92], a DKG variant with game-theoretical security against rational adversaries.

DRBs from external entropy. Participants periodically extract randomness from real-world entropy, e.g., real-time financial data [89] and public blockchains [90], [91], [93].

ISF-based DRBs. Participants use intermediate outputs of an ISF as random outputs, and succinct proofs for the ISF to make outputs verifiable. Such ISFs include Sloth [26] and Ephraim et al. [88]. RandRunner [35] extends this paradigm by allowing participants to execute the ISF in turn.

F. Limitations and resolutions

We discuss three limitations and the corresponding resolutions for RANDCHAIN, including the energy-efficiency, churn tolerance and finality support. We consider the concrete resolutions and analysis as future work.

1) **Energy efficiency:** As RANDCHAIN requires all nodes to solve SeqPoW puzzles to produce a random output, RANDCHAIN seems to be less energy-efficient than existing DRG-based DRBs. In fact, whether RANDCHAIN is less energy-efficient than existing DRG-based DRBs remains arguable. In terms of communication, RANDCHAIN costs strictly less energy than DRG-based DRBs, which require at least $O(n^2)$ communication complexity. The energy cost of communication is not always less than that of computation, as shown by existing literature [96]. In terms of computation, it remains arguable whether computing a random output through threshold cryptographic primitives (which can involve computationally intensive operations such as pairing, Lagrange interpolation, and Zero Knowledge Proofs) is more energy-efficient than non-parallelisable mining, where every node executes a single SeqPoW instance. In addition, with shorter block intervals, the energy cost by computing a random output in RANDCHAIN reduces linearly, while that in collaborative DRBs remains constant. We consider the energy efficiency analysis and improvement of RANDCHAIN as future work.

2) **Churn tolerance:** Similar to existing DRBs, RANDCHAIN does not tolerate churn, i.e., nodes joining and leaving. However, with little modifictaions, RANDCHAIN can tolerate churn like PoW-based consensus protocols. To tolerate churn [97], PoW-based blockchains adjust difficulty parameters adaptively to the rate of new blocks. In RANDCHAIN, the difficulty adjustment mechanism can use the number i of iterations running $\mathsf{SeqPoW.Solve}(\cdot)$ to 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. We consider a concrete construction and analysis on the difficulty adjustment mechanism as future work.

3) **Finality:** Due to the probabilistic Nakamoto consensus, RANDCHAIN does not achieve finality, and an appended block may be reverted later. A block being reverted does not lead to financial loss, as the random output is revealed only after the block becomes stable, guaranteed by the unbiasibility property. However, when a block is reverted, some randomness-based applications may abort the execution. We consider two approaches to achieve finality, namely the quorum mechanism and herding-based consensus, and consider concrete constructions and analysis as future work.

Quorum mechanism. Quorum [98] 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 [94], [98].

RandChain can employ the quorum mechanism as follows. A node signs a block to vote it. 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. The fault tolerance assumption changes to $n \geq 3f+1$. 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 the choices of other people. Herding-based consensus allows nodes to decide proposals according to neighbour nodes' votes only, rather than a quorum of votes. Existing research [65], [99] 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 [65] as a finality layer for Bitcoin Cash [100].