Good things come to those who wait: Dishonest-Majority Coin-Flipping Requires Delay Functions

Joseph Bonneau[1][3], Benedikt Bünz[1], Miranda Christ[2], and Yuval Efron[2]

[1] New York University
jcb@cs.nyu.edu,bb@nyu.edu
[2] Columbia University
mchrist@cs.columbia.edu,ye2210@columbia.edu
[3] a16z crypto

Abstract. We reconsider Cleve's famous 1986 impossibility result on coin-flipping without an honest majority. Recently proposed constructions have circumvented this limit by using cryptographic delay functions. We show that this is necessary: a (weak) notion of delay functions is in fact *implied* by the existence of a protocol circumventing Cleve's impossibility. However, such delay functions are weaker than those used in existing constructions. We complete our result by showing an equivalence, that these weaker delay functions are also sufficient to construct not just fair dishonest-majority coin-flipping protocols, but also the stronger notion of a distributed randomness beacon. We also show that this is possible in a weaker communication model than previously considered, without the assumption of reliable broadcast or a public bulletin board.

1 Introduction

Distributed *randomness beacon* (DRB) protocols [CMB23; KWJ23; RG22] have seen a resurgence of interest in recent years, with many new constructions in the literature and several now deployed in practice. In a DRB, a group of n participants collectively generate a public random string Ω that no party can predict or bias. The output Ω can be used as a seed for applications requiring public randomness such as consensus protocols, games and lotteries. Consensus protocols are a particularly important application, as leader election should be both unpredictable (to resist adaptive corruption) and unbiasable (to ensure fairness between all participants). Deployments like Ethereum's RANDAO are used to secure systems with billions of dollars in economic value.

To the best of our knowledge, all DRBs in practical use assume a majority of the participating nodes are honest. Some, like Ethereum, incentivize the required honesty by penalizing malicious behavior (such as failing to reveal committed values), but they still require an honest majority to obtain any security guarantees. This assumption is unsurprising, given Cleve's famous 1986 impossibility result [Cle86] showing that distributed coin-flipping (a closely related, but weaker, notion to DRBs) is impossible without a majority of honest participants. In the simplest coin-flipping scenario, Alice and Bob wish to generate a shared random bit that neither can predict nor influence, without trusting the other. Cleve ruled out solutions to this problem where either Alice or Bob is malicious, or more generally where a majority of n participants is malicious.

However, a few protocols in the literature purport to tolerate a *dishonest majority*, typically assuming only one honest node. The first such protocol, Unicorn [LW15a], is simple enough to describe in a single sentence:

> All nodes broadcast a random contribution within a fixed window and the output is the hash of all contributions run through a *delay function*.

Intuitively, a delay function is computable in polynomial time, but is highly *sequential*, in the sense that any polynomial-size circuit computing the function must have large depth. In other words, the advantage of parallel computation in computing the function is limited by this depth. The key ingredient in Unicorn is clearly the delay function, which is assumed to be too slow for any adversary to compute within the contribution window (note that this protocol requires synchrony). The delay function ensures that even an adversary who broadcasts their contribution last, with the benefit of seeing all other nodes' contributions, does not have enough time to compute the eventual output Ω resulting from any potential contribution of their own. Using a modern *verifiable* delay function [Bon+18], this protocol is also efficient as the delay function only need be computed once and all other nodes can quickly check that it is correct. Unicorn has inspired several follow-up protocols designed to work with a dishonest majority [Sch+23; Cho+23; CCB24].

The existence of these protocols raises an obvious question: how can they remain secure under a dishonest majority, when Cleve's result stated that this is impossible? In this work, we tackle this discrepancy. At a high level, we show that delay functions enable the circumvention of Cleve's impossibility, paving the way for the family of Unicorn-style protocols. More interestingly, we show unconditionally that any protocol which circumvents Cleve's impossibility can be used to construct a delay function. Hence, we show a surprising equivalence between dishonest-majority coin-flipping protocols and delay functions. We also show that Unicorn-style protocols can be constructed using a much weaker notion of a delay function than previously considered, which we show can be *unconditionally* boosted to a full-fledged delay function.

2 Related work

2.1 Coin-flipping and distributed randomness beacons

Rabin [Rab83] formalized the abstraction of a *randomness beacon* in 1983, using it as a tool for fair exchange and confidential disclosure. An ideal randomness beacon should broadcast random values at regular intervals that all parties can read, but none can predict or manipulate. Rabin's proposed instantiation was a satellite in space broadcasting signed, randomly chosen integers to Earth at regular intervals. Rabin's satellite stood in for a trusted third party, and there have been several popular beacon services that operate this way, including a beacon operated by NIST [Kel+19] and the widely used `random.org` service [Haa99]. Interestingly, the startup Cryptosat actually launched a satellite-based beacon service in 2022, realizing Rabin's original vision [Mic22].

In the absence of a trusted third party, the beacon abstraction can also be realized by a group of semi-trusted nodes continually running a collective *coin-flipping protocol* and publishing the results. The coin-flipping problem was extensively studied during the early 1980s [Blu83a; LMR83; DB84; BL85]. The original formulation of the problem appears lost to history, but it was presented by Blum in 1983 [Blu83a] as two mutually untrusted parties aiming to collectively agree to one random bit over the telephone. This application is useful on its own, though it was quickly recognized as an important tool for solving other problems including mental poker [GM82] and fair exchange [Blu83b]. While initially conceived as a two-party protocol, it was naturally extended to coin-flipping with multiple parties [BL85]. This line of work motivated Cleve's seminal impossibility result [Cle86] on coin-flipping without an honest majority.

The simplest approach for coin-flipping (dating to Blum [Blu83a]) is *commit-reveal*, in which each party commits to a random value and then all reveal and the results are combined (for example, by hashing or computing the exclusive-or). The use a of binding commitment locks each party into their contribution before seeing those of others. As long as all parties reveal, the result is uniformly random if any party's contribution is. However, the protocol is not secure against a malicious participant selectively dropping out after seeing others' revealed contribution (sometimes called a *last-revealer* attack).

Chor et al. [Cho+85] first proposed addressing this issue by having each participant used *verifiable secret sharing* (VSS) to share their contribution with the other nodes. Using VSS, a majority of nodes can reconstruct any aborting party's contribution, leading to the *commit-reveal-recover* paradigm. Observe that, given a t-of-n VSS scheme, a dishonest coalition of t nodes can learn honest nodes' contributions early and adjust their own contributions in response, biasing the output. However, a coalition of $n - t + 1$ nodes can bias the results by aborting, forcing a protocol restart. Hence, the best the protocol can achieve is setting $t = \frac{n}{2}$, requiring an honest majority.[4] This paradigm was later strengthened using publicly verifiable secret sharing (PVSS) [Sta96; Sch99], removing the need for interaction during the contribution phase.

[4] In practice, the two types of attack (prediction vs. bias) may have different implications, motivating a different parameter choice than $t = \frac{n}{2}$. The attacks are also distinct in that prediction leaves no public evidence, while honest nodes will be aware if the protocol is restarted due to withholding and the faulty nodes can be identified.

In the modern context, there has been extensive work on multiparty *distributed randomness beacon* protocols (DRBs), combining the earlier cryptographic formulations with performance and robustness insights from the distributed systems literature. Syta et al. introduced the notion of a DRB in 2017 [Syt+17], proposing protocols aiming to scale to hundreds of users, as well as protocols which utilize a more expensive one-time setup amortized across many rounds. There are now many DRB protocols; we refer the reader to several comprehensive surveys [CMB23; KWJ23; RG22]. Following Choi et al. [CMB23], we can divide approaches into several broad families:

– *commit-reveal-punish* protocols which keep the basic commit-reveal paradigm but incorporate financial penalties to discourage aborting the protocol [And+14; BK14; Qia17; Yak+20].
– *commit-reveal-recover* protocols which utilize some form of secret-sharing to recover from aborting nodes [Syt+17; CD17; CD20].
– *share-recover* protocols which skip the optimistic case and always use secret-sharing to recover each party's contribution [Syt+17; CSS19; GSX20]
– *threshold-based* protocols in which the parties construct a shared secret during a setup phase and use it to produce pseudorandom values in each DRB round [Syt+17; Dra24; Gal+21; Bea+23].

All of the approaches above assume an honest majority or supermajority (e.g., 2/3 honest nodes). The exceptions, based on delay functions, are discussed below.

2.2 Time-based cryptography and delay functions

Dwork and Naor introduced the notion of intentionally slow cryptographic computations in 1992 [DN92]. Their goal was requiring a moderately expensive computation before sending an email to raise the cost of spam. Most of their constructions were parallelizable (what would now be called *proof-of-work*), but they also proposed computing square roots modulo p, pointing out that this appears to be inherently sequential and hence requires significant time even with many parallel processors.

Rivest, Shamir and Wagner first proposed *time-lock encryption* and *time-lock puzzles* in 1996 [RSW96], with the goal of encrypting a message that could only be decrypted after significant time had elapsed. They introduced the problem of repeated squaring modulo a composite N as an inherently sequential function, with the possibility of efficient puzzle generation using a trapdoor (the factorization of N).[5]

Franklin and Malkhi [FM97] proposed simpler delay functions based on iterated hashing as a simple proof that some time had elapsed. Goldschlag and Stubblebine [GS98] first proposed using such *delay functions*[6] to construct a fair coin-flipping protocol, using the delay function after combining each participant's contribution to prevent biasing in a simple one-round protocol. They left open how to achieve efficient verification using different delay functions with algebraic properties. Boneh and Naor [BN00] proposed *timed commitments* in 2000, enabling parties to commit to a value in a manner that is temporarily hiding but can be recovered by computing a delay function (in their construction repeated squaring modulo N). Their focus was on fair exchange, though they noted that coin-flipping was a potential application as well.

Lenstra and Wesolowski [LW15a] formalized the use of delay functions for randomness generation in 2015, introducing the basic Unicorn protocol and proving it secure. They also proposed Sloth, a delay function based on computing a chain of modular square roots which allows for more efficient (though still linear in t) verification.

In 2018, Boneh et al. [Bon+18], citing randomness generation as a motivating application, introduced the modern notion of a *verifiable* delay function, requiring t steps to compute but enabling verification in polylog(t) steps. A number of VDF constructions have since been proposed based on repeated squaring [Pie18; Wes19], isogenies [De +19], hyperelliptic curves [CZ23] and lattices [LM23; AMZ24]. This line of work has also inspired many new constructions in time-based cryptography, including homomorphic timed commitments [MT19; Fre+21] and delay encryption [BD21].

[5] Interestingly, the authors posted a time-lock encrypted message as a challenge intended to take 35 years to open. In actuality it was opened 20 years later, though after only 3 years of computation on an FPGA [Obe19].

[6] Goldschlag and Stubblebine called them "delaying" functions.

While useful for constructing DRBs, Abram et al. [ARS24] actually showed that weaker primitives than VDFs (e.g. time-lock puzzles) are in fact sufficient. Abram et al. [ARS24] also constructed delay functions from weaker assumptions than were previously known, specifically one-way functions and the existence of a function that is hard for sequentially-bounded adversaries to predict.

Finally, several DRBs constructions embellish on the basic Unicorn framework. RandRunner [Sch+23] uses trapdoor VDFs in a leader-based DRB, enabling efficient computation by an honest leader and recovery by computing the sequential VDF in case of an absent leader. Bicorn [Cho+23] uses special homomorphic time-lock puzzles to construct a commit-reveal protocol which is highly efficient in the optimistic case (if all nodes open their commitment) but can recover in case any nodes drop out by combining all commitments into a single timed commitment to open. Cornucopia [CCB24] reduces the linear broadcast costs in Unicorn by employing a semi-trusted coordinator (trusted for liveness but not security) to combine participant contributions via a cryptographic accumulator, which is then fed into a delay function.

3 Technical Overview

3.1 Dishonest majority coin-tossing protocol implies a delay function (Section 5)

Cleve's result (Theorem 2). Our starting point is Cleve's [Cle86] result for a two-party (Alice and Bob) coin tossing protocol with a single dishonest party. Specifically, Cleve shows that any coin tossing protocol in such a setting that has high *consistency* (i.e., if both parties behave honestly, the probability that they agree on their output is bounded above 1/2) can be *biased* by a dishonest adversary. In other words, one dishonest party in the coin tossing protocol can significantly skew the distribution over the outputs towards a particular value. One strategy that can be carried out by a dishonest party is that of *aborting*. We say that that a party aborts at round i if all its messages from round i onwards are simply strings of fixed length of all zeroes. Given an r-round coin tossing protocol, Cleve lists $4r+1$ adversary strategies, and then proves that *at least one of them* has to realize a bias of about $\Omega(\frac{1}{r})$ in the resulting output. In fact, the behaviours of these adversaries are quite simple to describe. Apart from one, which is the adversary that halts immediately at the beginning of the protocol, any adversary from Cleve's list is characterized by the following.

- A choice of player, i.e., either Alice or Bob.
- A choice of round, i.e. an index $i \in [r]$.
- A choice of output bit $b \in \{0,1\}$.

This leads to $4r+1$ strategies: one for each combination of player/round/output bit, plus the strategy of halting immediately before round 1 without doing anything. Assume w.l.o.g. that Alice is chosen, with index i, and bit b, the adversary strategy is as follows: Run the protocol as Alice, behaving honestly up to round i, and compute the *default* output value at round i. The default value is the value that Alice would output if Bob were to behave honestly up to round i, then abort. If this default value is b, Alice *aborts*. Otherwise, run honestly for one more round, and then abort.

It is precisely these computations of default values that are of interest to us in showing the existence of a delay function. Specifically, each of these computations can be modeled as a function from $\{0,1\}^*$ to $\{0,1\}$. Note that in particular, these functions are efficiently computable, as they are computed by the honest party as well. Similarly to the list of $4r+1$ adversaries, there are $4r+1$ such functions, one for each adversary. We will show that if an adversary cannot carry out Cleve's attack, it must be unable to reliably compute one of these functions.

Circumventing Cleve implies a weak delay function (Theorem 3). Suppose that we have a protocol that circumvents Cleve's lower bound. This adversary cannot reliably compute all of the default value functions; otherwise, it could carry out Cleve's attack by computing the functions to obtain the default values and deciding whether to abort. Therefore, *at least one* of the $4r+1$ default value functions must be at least somewhat unpredictable by the adversary.

Now, recall our motivation in revisiting Cleve's bound: Recent DRB proposals from the applied cryptography community claim to achieve security against a dishonest majority, *if the adversary is sequentially* *bounded.* These protocols make use of delay functions, and we observe that this is no accident. Using the above argument that the adversary must be unable to reliably compute the default value functions, and taking the adversary to be sequentially bounded, we immediately obtain that some default value function must be a *weak delay function* (Definition 3). That is, there exists an efficiently samplable distribution of inputs over which the function's output cannot be predicted with greater than $1 - 1/\mathsf{poly}(\lambda)$ probability.

Bootstrapping from weak to strong delay functions (Theorem 4). Finally, with a weak delay function in hand, we amplify its hardness into being a proper delay function (Definition 2). Our approach emulates that of a parallel repetition theorem. Intuitively, if a low parallel time adversary can predict the outcome of a function f on input x w.p. at most p, then a low parallel time adversary should be able to predict f^n on inputs $x_1, \ldots, x_n$ w.p. about the order of magnitude of p^n. Translating this intuition into a rigorous proof however involves several challenges:

- **Extracting a strong predictor from a weak predictor**. The main technical hurdle in all parallel repetition proofs is extracting a *local* adversary from a *global* adversary. In other words, given an adversary that can predict the output of f^n on inputs $x_1, \ldots, x_n$ w.p. at least $\frac{1}{p(\lambda)}$ for a given polynomial $p(\cdot)$, one needs to extract from it an adversary that can predict the output of f on an input x with probability $1 - \frac{1}{q(\lambda)}$ for any given polynomial $q(\cdot)$. We emulate the classical proof of amplifying a weak one-way function to a strong one-way function (e.g. [PS10, Theorem 35.1]) and adapt it to delay functions. This adaptation, however, leads to another challenge.
- **Verifiability.** The approach of weak-to-strong one-way function amplification has roughly the following structure: Sample random inputs $x_1, \ldots, x_n$, feed them to the *global* adversary to obtain predictions $y_1, \ldots y_n$, and check locally whether $f(x_i) = y_i$ for some coordinate i, and if so, output y_i as the prediction. When one discusses delay functions, the last step becomes a problem, as the *local* adversary we construct cannot afford to compute $f(x_i)$, since it also has to run in low parallel time. In fact, it was shown that for a different but related notion of puzzles, such an amplification does not hold if the puzzle is not verifiable [CHS05].
  To circumvent this challenge, we show that any function f can *unconditionally* be augmented into a function f' for which one can perform output *verification* in constant parallel time. The key idea is that we define the output of $f'(x_i)$ to include both y_i and the wire assignments to a circuit computing f. One can now verify that f' is correct by checking the consistency of each gate in this circuit, and checking that its output wires match y_i. Crucially, this is a local computation that can be highly parallelized. This allows the adversary, which has arbitrary polynomial parallelism, to check whether $f'(x_i) = y_i$ without needing to *compute* $f'(x_i)$ *from scratch* (Section 5.3).

3.2 Delay functions imply n-party distributed randomness beacons (Section 6)

We complete our equivalence by showing that, in the random oracle model, our notion of a delay function implies an n-party distributed randomness beacon that is secure against a sequentially-bounded adversary corrupting all but one party. Therefore, if there exists a protocol circumventing Cleve's lower bound, there exists a delay function, and there exists an n-party DRB. This result closes the gap between two-party protocols that circumvent Cleve's lower bound, and DRB protocols with (seemingly) far stronger security.

Cleve's lower bound shows that coin-tossing protocols with even just constant-probability agreement and certain inverse-polynomial bias are impossible. In contrast, a distributed randomness beacons requires agreement with probability 1, and *negligible* bias. A priori, it is possible that there exist delay-based protocols that circumvent Cleve's protocol by achieving weak agreement and weak unbiasability, but full agreement and unbiasasbility are impossible. Somewhat surprisingly, our result (Theorem 7) shows that two-party coin-tossing protocols with weak agreement and certain inverse-polynomial bias *imply* n-party randomenss beacons that are fully unbiasable by an adversary corrupting even $n - 1$ parties. Of course, this adversary must be sequentially bounded.

While existing works (described in Section 2) already construct distributed randomness beacons using time-based cryptography, they use stronger notions of delay functions (e.g., verifiable delay functions) and assume a public bulletin board. Fortunately, their approach can be adapted with a few modifications, which we detail below. We use Unicorn [LW15b] as our starting point.

The Unicorn protocol. In the simple and elegant Unicorn protocol [LW15b], n parties use a public bulletin board to generate shared randomness. The protocol starts at some time T_0. Before a contribution deadline T_1, each participant P_i posts a uniform randomness contribution r_i to the bulletin board. After time T_1, a delay function Eval is computed on the hash of the concatenation of all parties' contributions: $y = \mathsf{Eval}(H(r_1||\dots||r_n))$. If Eval requires more than $(t_1 - t_0)$ time for the adversary to predict on a random input, y is unpredictable in the random oracle model: If any participant P_i is honest and contributes a random r_i, $H(r_1||\dots||r_n)$ is freshly drawn from the uniform distribution after time t_0. Therefore, the beacon output $\Omega = H(y)$ is indistinguishable from random by such an adversary.

Unicorn with a weaker delay function. Observe that security of Unicorn hinges on the fact that Eval is unpredictable on a *uniform* input—whereas the notion of a delay function we consider is only guaranteed to be unpredictable on an input generated by an algorithm $\mathsf{SampleInput}$ that is specified as part of the delay function. In our protocol, each party P_i independently generates a value x_i from $\mathsf{SampleInput}$, and we let y be the concatenation of the *delay function evaluations* of the x_i's. That is, $y = (\mathsf{Eval}(x_1)||\dots||\mathsf{Eval}(x_n))$. y is unpredictable if Eval is unpredictable and any x_i was indeed sampled from $\mathsf{SampleInput}$; therefore, taking the beacon output to be $\Omega = H(y)$ yields a secure DRB. Observe that the random oracle here is used to convert an unpredictable value (y) to a random value (Ω). We leave removing the random oracle as an interesting direction for future work.

It also is worth noting that this modified protocol requires evaluating the delay function n times, compared to just once in Unicorn. If desired, one could have the parties jointly sample an input by each P_i sampling a uniform r_i and using $H(r_1||\dots||r_n)$ as the random coins to generate an x from $\mathsf{SampleInput}$. However, as our protocol is theoretical in nature, we use the simpler though less efficient approach of each party sampling its own input.

Obviating reliance on a public bulletin board. Unicorn and similar DRBs assume a public bulletin board available to all parties. We observe that we can avoid this assumption by relying instead on a protocol realizing Byzantine Broadcast. Such protocols are known to exist even in the setting where $n-1$ out of n participants are corrupted [DS83].

n-party DRB from *weak* delay functions. Finally, we apply our amplification theorem from weak to full delay functions (Theorem 4) to obtain a corollary that even weak delay functions are sufficient to construct an n-party DRB.

4 Preliminaries

Let λ denote the security parameter. We write $\mathsf{poly}(\lambda)$ to mean a polynomial function in λ. A function f of λ is *negligible* if $f(\lambda) = O(\frac{1}{\mathsf{poly}(\lambda)})$ for every polynomial $\mathsf{poly}(\cdot)$. We write $f(\lambda) \leq \mathsf{negl}(\lambda)$ to mean that f is negligible.

Throughout the paper we consider a network of players, with communication over a peer-to-peer synchronous network. Namely, all players have access to a global clock progressing in *timeslots* that is perfectly synchronized between all players. Furthermore, each message sent between honest players takes at most 1 time slot to arrive at its recipient.

4.1 Delay functions

We follow a combination of [Bon+18; ARS24], with some modifications for our setting, in defining delay functions. Our main modification is the introduction of a $\mathsf{SampleInput}$ algorithm that defines the input distribution over which the function's output should be unpredictable. In contrast, existing definitions from [Bon+18; ARS24] consider only unpredictability over the uniform distribution. Our new definition is necessary for our result—we show that honest-majority coin-tossing protocols imply delay functions that are unpredictable over an input distribution which is not necessarily uniform and depends on the protocol.

```
$\underline{\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,\mathsf{DF}}(\lambda)}$
$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$
$\alpha \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{pp})$
$x \leftarrow \mathsf{SampleInput}(1^\lambda, \mathsf{pp})$
$\tilde{y} \leftarrow \mathcal{A}_1(1^\lambda, \mathsf{pp}, x, \alpha)$
$y \leftarrow \mathsf{Eval}(1^\lambda, \mathsf{pp}, x)$

return $\tilde{y} \overset{?}{=} y$
```

Fig. 1. Delay function sequentiality game

Sequential and parallel time. Following [Bon+18], we say that an algorithm runs in *parallel time* t on p processors if a PRAM machine with p processors can run the algorithm in time t. The *total*, or *sequential* time of an algorithm is the time it takes to run on a machine with a *single* processor.

Definition 1 (Delay function). *A delay function is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ *such that:*

$\mathsf{Setup}(1^\lambda) \rightarrow \{\mathsf{pp}\}$**:** Setup *takes as input the security parameter and outputs public parameters* pp.
$\mathsf{SampleInput}(1^\lambda, \mathsf{pp}) \rightarrow x$**:** $\mathsf{SampleInput}$ *is a randomized algorithm that takes as input the security parameter and outputs a random value x from the input distribution.*[7]
$\mathsf{Eval}(1^\lambda, x, \mathsf{pp}) \rightarrow y$**:** Eval *is a function that takes an input x and outputs a corresponding evaluation y.*

For brevity's sake, we sometimes omit 1^λ as an input to $\mathsf{SampleInput}$ and Eval.

Efficiency. The sampling algorithm and evaluation function are computable in polynomial total time.

Sequentiality. Security of a delay function states that no efficient adversary should be able to compute Eval much faster than the function's delay parameter. We allow the adversary to run two algorithms $\mathcal{A}_0$ and $\mathcal{A}_1$. $\mathcal{A}_0$ (the adversary's precomputation algorithm) takes as input the public parameters and can run in arbitrary polynomial total and parallel time to produce an advice string α. $\mathcal{A}_1$ (the adversary's online algorithm) takes as input α, the public parameters, and the challenge input. It runs in at most $t(\lambda)$ parallel time and attempts to compute the corresponding output.

Definition 2 ($t(\cdot)$-sequentiality ([Bon+18]).). *A delay function* $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ *is $t(\cdot)$-sequential if for all randomized algorithms $\mathcal{A}_0, \mathcal{A}_1$ running in total time* $\mathsf{poly}(\lambda)$*, and $\mathcal{A}_1$ running in parallel time at most $t(\lambda)$,*

$$\Pr\left[\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,\mathsf{DF}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

Definition 3 (ϵ-weak $t(\cdot)$-sequentiality.). *An* ϵ-weak *delay function* $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ *is* ϵ-weakly $t(\cdot)$-sequential *if for all randomized algorithms $\mathcal{A}_0, \mathcal{A}_1$ running in total time* $\mathsf{poly}(\lambda)$*, and $\mathcal{A}_1$ running in parallel time at most $t(\lambda)$,*

$$\Pr\left[\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,\mathsf{DF}}(\lambda) = 1\right] \leq 1 - \epsilon.$$

Comparison to VDFs ([Bon+18]) VDFs [Bon+18] feature an additional Verify function (optionally using a proof π output by Eval) to efficiently verify the results of the delay function. Though in the real world efficient verification is highly desirable, we (and [ARS24]) consider verifiability to be a bonus property of a delay function and therefore do not include it in the specification. We later show that one can transform any delay function into one with a verifier than can be implemented in few sequential steps on many parallel processors. However, this verifier is not as efficient as a VDF requires.

[7] We emphasize that this distribution may not be uniform.

Another important difference is that VDFs allow one to specify the delay parameter $t(\cdot)$ as an input to the Setup algorithm. In contrast, our delay function is defined with respect to a specific function $t(\cdot)$.

Finally, we depart slightly from the original sequentiality definition of VDFs [Bon+18], which considers adversaries that have a bounded number p of parallel processors. We consider a stronger class of adversaries, with an arbitrary polynomial number of parallel processors at their disposal.

Comparison to complexity-theoretic hard functions. At this point the curious reader might wonder whether delay functions exist unconditionally, as their definition strikes a resemblance to known results about unconditionally "hard" functions that cannot be computed by circuits of a fixed size (by Shannon's counting argument) [Sha49], or by bounded-time Turing Machines (by the Time Hierarchy Theorem [HS66]). A natural question is whether delay functions similarly exist unconditionally. We stress that that the answer to this question is unknown, and in particular time hierarchy theorems imply a (seemingly) significantly weaker notion of hardness than the one required from delay functions. In the former, hardness requires only that no circuit or Turing machine from the given class can compute the function *exactly.* However, the function may be computable on a significant fraction of inputs. In contrast, our notion of a delay function requires that no sequentially bounded algorithm can compute the function on a *non-negligible fraction of inputs.*

Furthermore, to our knowledge the class of circuits that we consider (i.e., bounded parallel time) does not have unconditionally hard but still polynomial-time computable functions. Recall that the class of circuits we consider have bounded depth but with arbitrary polynomial width.

4.2 Coin-tossing protocols

Coin-tossing protocols and DRBs are both interactive protocols designed to output a random string agreed on by all parties. Although they are variants of the same notion, we present and use both definitions. The notion of coin-tossing protocols is significantly weaker, requiring only that the bias of the random string is upper bounded by some constant and that parties agree with only constant probability. In contrast, DRBs allow only negligible bias and require agreement among all honest parties. We prove our lower bound for the weaker notion of coin-tossing protocols, and our upper bound for the stronger notion of DRBs.

We recall the notion of a two-party coin-tossing protocol considered by Cleve [Cle86].

Definition 4 (Coin-tossing protocol). *A* coin-tossing protocol *is a series of processors* $\{A^\lambda, B^\lambda\}_{\lambda=0}^\infty$ *each running in* $\mathsf{poly}(\lambda)$ *time.* A^λ *and* B^λ *are deterministic algorithms that take as input private strings of random bits* r_a *and* r_b *respectively. In each round,* A^λ *and* B^λ *communicate with each other; without loss of generality* A^λ *sends a message to* B^λ*, which sends a message back to* A^λ*. At the end of the protocol,* A^λ *outputs a bit* a*, and* B^λ *outputs a bit* b*.*

Consistency. *A coin-tossing protocol is* ϵ-consistent *if* $\Pr_{r_a, r_b}[a = b] \geq \frac{1}{2} + \epsilon$*. Note that in this consistency definition,* A^λ *and* B^λ *behave according to the protocol.*

Bias. *A corrupted party (without loss of generality)* A^λ *can* bias *the protocol by* δ *if* $\left|\Pr[b = 1] - \frac{1}{2}\right| \geq \delta$*. We say that the* bias of the protocol *is at least* δ *if there exists such an efficient adversary* A^λ *(or* B^λ*).*

4.3 Distributed randomness beacons

Definition 5 (Distributed randomness beacon (DRB)). *A* distributed randomness beacon *is a tuple of polynomial-time protocols* $\Pi = (\mathsf{Setup}, \mathsf{PreProcess}, \mathsf{Post}, \mathsf{Finalize})$ *run by participants* $\mathcal{P} = \{P_1, \ldots, P_n\}$ *such that:*

$\mathsf{Setup}(1^\lambda) \to \mathsf{CRS}$: Setup *is a randomized algorithm that takes as input the security parameter and outputs a common reference string. We assume that* Setup *is run by a trusted party and* CRS *is made available to all participants.*

$\mathsf{PreProcess}_i(1^\lambda) \to \beta_i$: $\mathsf{PreProcess}$ *is a randomized algorithm run locally by each participant* P_i *to produce a string* β_i*.*

```
$\underline{\mathcal{G}^{\text{indist}}_{\mathcal{A},\mathcal{C},t,\mathsf{DRB}}(1^\lambda)}$
$\mathsf{CRS} \leftarrow \mathsf{Setup}(1^\lambda)$
For all $i \in [n]$, $\beta_i \leftarrow \mathsf{PreProcess}(1^\lambda)$
$\alpha_0 \leftarrow \mathcal{A}_0(1^\lambda, \mathsf{CRS}, \{\beta_i\}_{P_i \in \mathcal{C}})$
$\{\mathsf{tr}^{\mathsf{post}}_1, \ldots, \mathsf{tr}^{\mathsf{post}}_n\} \leftarrow \mathsf{Post}^{\mathcal{A}_1(1^\lambda, \alpha_0)}(\mathsf{CRS}, \beta_i)$
$i, \alpha_1 \leftarrow \mathcal{A}_1(1^\lambda, \alpha_0)$

$\Omega_0 \leftarrow \mathsf{Finalize}(1^\lambda, \mathsf{CRS}, \mathsf{tr}^{\mathsf{post}}_i)$
$\Omega_1 \leftarrow \{0,1\}^\lambda$
$b \leftarrow \{0,1\}$
$b' \leftarrow \mathcal{A}_1(1^\lambda, \alpha_1, \Omega_b)$
return $b \overset{?}{=} b' \wedge i \in \mathcal{P} \setminus \mathcal{C}$
```

Fig. 2. Security game for $t(\cdot)$-indistinguishability

$\mathsf{Post}(1^\lambda, \beta_i)$: Post *is an interactive protocol run by all participants, in which they exchange randomness contributions. Each participant P_i gets as input its string β_i at the start of the protocol.*

$\mathsf{Finalize}(1^\lambda, \mathsf{CRS}, \mathsf{tr}^{\mathsf{post}}_i) \to \Omega_i$: $\mathsf{Finalize}$ *is a (possibly randomized)* algorithm *run by each participant P_i on input the security parameter, the CRS, and their local transcript of the* Post *protocol* $\mathsf{tr}^{\mathsf{post}}_i$. *It produces Ω_i, the output of the beacon according to party P_i.*

While $\mathsf{Finalize}$ is computable in polynomial time, in the case of delay-function-based constructions security relies on this computation requiring more parallel time than is available to the adversary.

We use $\Pi^{\mathcal{A}(\alpha),\mathcal{C}}(1^\lambda) \to \{\Omega_1, \ldots, \Omega_n\}$ to denote the outputs of parties $P_1, \ldots, P_n$ generated by running Π with an adversary $\mathcal{A}$ controlling a set of corrupted participants $\mathcal{C} \subseteq \mathcal{P}$. Here, the adversary may take as input an advice string α. In particular, the adversary can see all computations performed by these corrupted parties, and it can direct these parties to deviate from the protocol arbitrarily. Similarly, we use $\mathsf{Post}^{\mathcal{A}(\alpha),\mathcal{C}}(\mathsf{CRS}) \to \{\mathsf{tr}^{\mathsf{post}}_1, \ldots, \mathsf{tr}^{\mathsf{post}}_n\}$ to denote the transcripts of all parties produced by running the Post protocol given CRS, with the adversary controlling the parties in $\mathcal{C}$. Here, the adversary may take as input an advice string α. We may specify that $\mathcal{A}$ is computationally constrained during the Post protocol; for example, we will often require $\mathcal{A}$ to run in parallel time at most some $t(\lambda)$.

As mentioned before, existing DRBs in the literature typically assume that the DRB output is posted on a public bulletin board, trivially resulting in agreement among all parties. Since we study DRBs without assuming a bulletin board, we must define agreement:

Definition 6 (Agreement). *A DRB satisfies* agreement *if for any p.p.t. adversary $\mathcal{A}$ corrupting a subset of parties $\mathcal{C} \subseteq \{P_1, \ldots, P_n\}$:*

$$\Pr[\exists P_i, P_j \notin \mathcal{C} : r_i \neq r_j \mid \{r_1, \ldots, r_n\}] \leq \mathsf{negl}(\lambda).$$

We note that agreement is vacuously achieved when all parties but one are corrupted. However, with at least two honest parties it becomes nontrivial.

We closely follow [Cho+23] in defining indistinguishability of a DRB against a sequentially-bounded adversary.

Definition 7 ($t(\cdot)$-indistinguishability). *The $t(\cdot)$-indistinguishability game tasks a $t(\cdot)$-sequentially bounded adversary with distinguishing between a truly random string and the output of the given protocol. The adversary's computation is broken down into two phases. In the precomputation phase, the adversary may run a polynomial-time algorithm $\mathcal{A}_0$ with arbitrary (polynomial) parallel time to produce an advice string.*

In the protocol phase, the adversary $\mathcal{A}_1$ takes as input the advice string and is constrained to run in parallel time at most $t(\lambda)$. $\mathcal{A}_1$ engages in the Post *protocol, controlling the corrupted parties. This game, denoted $\mathcal{G}^{\text{indist}}$, is given in Figure 2.*

A DRB Π satisfies indistinguishability against a $t(\cdot)$-sequentially bounded adversary corrupting m parties *if for all $\mathcal{C} \subseteq \mathcal{P}$ such that $|\mathcal{C}| = m$, and $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$,*

$$\Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},\mathcal{C},t,\mathsf{DRB}}(1^\lambda) = 1\right] \leq \mathsf{negl}(\lambda).$$

4.4 Byzantine Broadcast

To remove the assumption of a public bulletin board and design a DRB secure in the dishonest majority setting, we rely on the well-known primitive of Byzantine Broadcast (BB). Specifically, in the BB problem there is an identified sender s, whose identity is known to all players, holding an input v. The goal is to design a protocol Π that satisfies the following in the presence of f byzantine failures (which could potentially include s) amongst the n players. We refer to the players experiencing such failures as *corrupt* players. We refer to non corrupt players as *honest.*

1. **Termination.** Every honest player produces an output and halts after a finite number of rounds.
2. **Agreement.** All honest parties output the same value.
3. **Validity.** If s is honest, then all honest parties output v.

As we are mainly concerned with a feasibility result, we leave aside complexity considerations, and we relay on the following well known result of Dolev-Strong [DS83] whenever we want to instantiate BB:

Theorem 1 (Dolev-Strong [DS83]). *Assuming PKI, there exists a deterministic Byzantine Broadcast protocol $\mathsf{BB}_{\mathsf{DS}}$ tolerating f byzantine faults for any $f < n$. $\mathsf{BB}_{\mathsf{DS}}$ has round complexity of n.*

5 Dishonest majority coin-tossing protocol implies a delay function

We will show that a two-party coin-tossing protocol that tolerates a single corrupted party implies a delay function, if this adversary is constrained to run in $t(\lambda)$ parallel time, for some function $t(\cdot)$. We first show that such a protocol implies a much weaker notion of a delay function with the same delay parameter (Definition 3): that is, a function that cannot be predicted with greater than $1 - 1/\mathsf{poly}(\lambda)$ probability in parallel time $t(\lambda)$. We then show that any weak delay function can *unconditionally* be boosted to construct a delay function with full unpredictability (i.e., with negligible probability of successful prediction) with the same delay parameter.

5.1 Cleve's impossibility

We first recall Cleve's impossibility result [Cle86]:

Theorem 2 (Cleve). *Any ϵ-consistent two-party coin-tossing protocol running in r rounds can be biased by at least $\frac{\epsilon}{4r(\lambda)+1}$ by a malicious party.*

Proof. Cleve defines $4r(\lambda) + 1$ faulty processors $\tilde{A}^\lambda, A^\lambda_{ij}, B^\lambda_{ij}$ for all $i \in [r(\lambda)], j \in \{0, 1\}$. To describe these processors, Cleve introduces some vocabulary: *quitting* and a *default value.* A processor *quits* in round i if all messages that it sends during and after that round are simply strings consisting of all zeros. The default value a_i is the value that (honestly behaving) A^λ would output at the end of the protocol if (faulty) B^λ were to quit in round i. a_i depends implicitly on the inputs of A^λ and B^λ. Similarly, the default value b_i is the value that B^λ would output at the end of the protocol if A^λ were to quit in round i.

Now, we describe these faulty processors. $\tilde{A}^\lambda$ is the processor that always quits in round 1. A^λ_{i0} behaves honestly up to (at least) round i, then attempts to bias the output toward 0 by quitting in either round i or round $i + 1$. That is, A^λ_{i0} checks if $a_i = 0$ and quits in round i if so; otherwise it quits in round $i + 1$. A^λ_{i1} is defined analogously except that it checks if $a_i = 1$ instead. B^λ_{i0} and B^λ_{i1} are defined analogously except that they check for the value of b_i.

Letting Δ be the average of the biases of these $4r(\lambda) + 1$ faulty processors, it follows from ϵ-consistency and some algebra that $\Delta \geq \frac{\epsilon}{4r(\lambda)+1}$. Therefore, at least one of these processors must have at least this bias, completing the proof. □

Observe that all processors $A_{ij}^{\lambda}, B_{ij}^{\lambda}$ must be able to compute their default values a_i, b_i before deciding how to behave in round i. Intuitively, if a two-party coin-tossing protocol is secure with a faulty processor, there must be at least one of these default values that cannot be computed reliably in a short amount of time. The function computing this default value will be our delay function.

5.2 Circumventing Cleve's impossibility implies a weak delay function

We will consider corrupted parties that are constrained to run in $t(\lambda)$ parallel time in each round of interaction, and polynomial total time. That is, in each round of interaction the corrupted party takes as input the transcript of the protocol thus far, and its computation tape from the end of the prior round. It may then perform a computation taking at most $t(\lambda)$ parallel time. At the end of the protocol, once all interaction has ceased, the adversary may run in arbitrary polynomial parallel time.

Remark 1. While not necessary to understand or prove our lower bound, one may wonder whether it is possible to successfully run a coin-tossing protocol with an adversary constrained in this way. We give a concrete example of such a protocol in Section 6. The key is to accept messages sent in the protocol only if they arrive within some set amount of time Δ. In practice, Δ is chosen so that it is infeasible to execute a $t(\lambda)$-sequential computation on real-world hardware.

Theorem 3. *Let Π be a an $r(\lambda)$-round $\epsilon(\lambda)$-consistent two-party coin-tossing protocol with bias $< \frac{\epsilon}{10r(\lambda)}$ under the corruption of a single party whose computation in each round of the protocol runs in parallel time at most $t(\lambda)$. Then there exists a $\frac{1}{q(\lambda)}$-weak $t(\lambda)$-sequential delay function* DF *for some polynomial $q(\lambda)$.*

Proof. Let Π be an $r(\lambda)$-round $\epsilon(\lambda)$-consistent two-party coin-tossing protocol, and let $C(=\tilde{A}), A_{i,j}, B_{i,j}, i \in [r(\lambda)], j \in \{0,1\}$ be the faulty processors defined in Cleve's argument [Cle86], such that there exists an adversary from these $4r(\lambda)+1$ that biases the protocol by at least $\frac{\epsilon}{4r(\lambda)+1}$. We note that all of these faulty processors' outputs can be computed in polynomial total time, as the protocol is efficient. Denote by $\mathcal{A}$ be the adversary controlling the corrupted party that executes Cleve's attack. For $Z \in \{A,B,C\}, i \in [r(\lambda)]$, denote by $f_{Z,i,j}(s_Z, \mathsf{Tr}_i)$ (for $Z \in \{A,B\}$) or f_C (for $Z = C, i = j = 0$) the function employed by $Z_{i,j}/C$ to compute its default value v, based on its own internal state, and the transcript of the protocol Π up to round $i (i = 0$ for $C)$, denoted Tr_i. Formally stating these functions in the notation of delay functions as Definition 1, we have the following for each $i \in [r(\lambda)], j \in \{0,1\}, Z \in \{A,B,C\}$:

- $\mathsf{Setup}(1^\lambda) \to \perp$. In particular, $\mathsf{pp} = \perp$ and we thus omit it from further notation in this section. Note that this is a virtue of our result, i.e. the delay function we extract from Cleve's argument requires no public parameters.
- $x \leftarrow \mathsf{SampleInput}(1^\lambda)$ simulates the honest behaviour of the protocol of both parties up to round i on random inputs and then outputs the transcript (Tr_i) of the protocol up to round i concatenated with the internal state of party Z at the beginning of round i. This output is denoted by x and is the input to $f_{Z,i,j}$.
- $\mathsf{Eval}(x) = f_{Z,i.j}(x)$.

Let $q(\lambda) = 10r(\lambda)^4$, and assume towards a contradiction that *all* of these functions are *not* $(1/q(\lambda))$-weakly $t(\cdot)$-sequential. Because $\mathsf{pp} = \perp$, the preprocessing adversary $\mathcal{A}_0$ from the t-sequentiality game (Figure 1) can be implemented by simply hard-coding its advice in the sequentially bounded adversary $\mathcal{A}_1$. That is, there exist algorithms $\mathcal{A}_{Z,i,j}^q$ running in parallel time at most $t(\lambda)$ and polynomial total time, such that the following holds:

$$\Pr_{x \leftarrow \mathsf{SampleInput}(1^\lambda), \Pi_i}[y \leftarrow \mathcal{A}_{Z,i,j}^q(1^\lambda, x) \wedge y = f_{Z,i,j}(x)] \geq 1 - \frac{1}{q(\lambda)}.$$

That being the case, it means that w.p. at least $(1 - \frac{1}{q(\lambda)})^{4r(\lambda)+1} > e^{-\frac{1}{r(\lambda)}}$, the adversary can successfully carry out Cleve's attack as all the default values are computed in time for the proceedings actions to be performed in the same time slot. Denote by S_1 the event that all the functions $f_{Z,i,j}$ were computed correctly by the adversary. Thus we have that the total bias resulting by the adversary's strategy is:

$$\begin{aligned}
\mathsf{bias}_{\mathcal{A}} &= \Pr[S_1]\frac{\epsilon}{4r(\lambda)+1} - \Pr[\neg S_1] \\
&= e^{-\frac{1}{r(\lambda)}} \cdot \frac{\epsilon}{4r(\lambda)+1} - \Pr[\exists Z,i,j \text{ s.t. } f_{Z,i,j}(x) \neq y] \\
&\geq e^{-\frac{1}{r(\lambda)}} \cdot \frac{\epsilon}{4r(\lambda)+1} - \sum_{Z,i,j} \frac{1}{q(\lambda)} \qquad (1) \\
&\geq e^{-\frac{1}{r(\lambda)}} \cdot \frac{\epsilon}{4r(\lambda)+1} - \frac{1}{r(\lambda)^3} \\
&\geq \frac{\epsilon}{10r(\lambda)},
\end{aligned}$$

where Equation (1) follows from an application of the union bound.

We have arrived at a contradiction. Thus there exists Z, i, j such that $f_{Z,i,j}$ is a $(1/q(\lambda))$-weakly $t(\cdot)$-sequential delay function. □

We observe that this result extends to n-party coin-tossing protocols tolerating a dishonest majority, applying another result of Cleve [Cle86]:

Corollary 1. *Let Π be a an $r(\lambda)$-round $\epsilon(\lambda)$-consistent n-party coin-tossing protocol with bias $< \frac{\epsilon}{10r(\lambda)}$ under the corruption of a dishonest majority, where each corrupted party's computation in each round of the protocol runs in parallel time at most $t(\lambda)$. Then there exists a $\frac{1}{q(\lambda)}$-weak $t(\lambda)$-sequential delay function* DF *for some polynomial $q(\lambda)$.*

Proof. Cleve remarks that any n-party dishonest-majority r-round protocol yields a secure two-party protocol with $n \cdot r \cdot c$ rounds, where c is the number of communication channels in the network. c is upper bounded by n^2.

Suppose that the n-party protocol is secure against an adversary controlling a dishonest majority of the parties, each of which is constrained to run in $t(\lambda)$ parallel time. Note that any such adversary can be simulated in the corresponding two-party protocol in $t(\lambda)$ parallel time, by running these algorithms in parallel. Therefore, if there exists an n-party protocol secure against this $t(\lambda)$-bounded adversary controlling a dishonest majority, there exists a two-party protocol secure against a single dishonest adversary running in $t(\lambda)$ parallel time.

Therefore, any n-party dishonest-majority coin-tossing protocol yields a secure two-party protocol with a polynomial number of rounds, which yields a weak delay function by Theorem 2. □

5.3 Boosting a weak delay function to a full delay function

In this section our goal is to bootstrap the weak delay function we obtained from Cleve's construction (see Definition 3) into a proper delay function (see Definition 2). Our approach follows the intuition of parallel repetition arguments: If predicting the outcome of a function f on an input x can be done with probability at most p, then one can hope that predicting the outcome of f^n on inputs $x_1, \ldots, x_n$ can be done with probability on the order of magnitude of p^n. The usual challenge in such parallel repetition arguments is extracting an adversary $\mathcal{A}'$ that can predict the output of $f(x)$ with high confidence (e.g., $1 - 1/\mathsf{poly}(\lambda)$) from a given adversary $\mathcal{A}$ that has only $1/\mathsf{poly}(\lambda)$ success in predicting $f(x_1), \ldots, f(x_n)$. The key to many such arguments is that $\mathcal{A}'$ runs many instances of $\mathcal{A}$, observes if any is correct, and outputs its prediction if so. To apply this paradigm to delay functions, the function must be verifiable.

Therefore, as discussed in Section 3.1, we show that any weak delay function f can be turned into a weak delay function $\overline{f}$ that in addition is verifiable in $O(1)$ parallel time. It is exactly with this notion of verifiability that we begin to pour rigor into the above intuition for the result.

Definition 8 (Verifiability in parallel time). *A delay function* $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ *is* verifiable in parallel time $t(\cdot)$ *if there exists an algorithm* $\mathsf{Verify}(1^\lambda, \mathsf{pp}, x, y)$ *running in parallel time at most* $t(|z|)$ *on at most* $\mathsf{poly}(|z|)$ *parallel processors on input* z*, satisfying both correctness and soundness:*

Correctness.

$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ x \leftarrow \mathsf{SampleInput}(\mathsf{pp})}} [y \leftarrow \mathsf{Eval}(\mathsf{pp}, x) \wedge \mathsf{Verify}(\mathsf{pp}, x, y) \neq \mathsf{true}] \leq \mathsf{negl}(\lambda).$$

Soundness. *For any efficient adversary* $\mathcal{A}$ *(of arbitrary polynomial sequential time),*

$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ x \leftarrow \mathsf{SampleInput}(\mathsf{pp})}} \begin{bmatrix} y \leftarrow \mathcal{A}(1^\lambda, \mathsf{pp}, x) \\ y \neq \mathsf{Eval}(\mathsf{pp}, x) \wedge \\ \mathsf{Verify}(\mathsf{pp}, x, y) = \mathsf{true} \end{bmatrix} \leq \mathsf{negl}(\lambda).$$

For brevity's sake, we sometimes omit 1^λ as input to Verify.

Lemma 1. *Let* $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$ *be a* $(1/q(\lambda))$*-weak delay function that is* $(1/q(\lambda))$*-weakly unpredictable in parallel time* $t(\cdot)$*. One can construct* $\overline{\mathsf{DF}} = (\overline{\mathsf{Setup}}, \overline{\mathsf{SampleInput}}, \overline{\mathsf{Eval}})$ *that is* $(1/q(\lambda))$*-weakly unpredictable* and verifiable *in parallel time* $t(\cdot)$*.*

Proof. Both $\overline{\mathsf{Setup}}$ and $\overline{\mathsf{SampleInput}}$ stay the same as $\mathsf{Setup}, \mathsf{SampleInput}$, respectively. Since Eval is polynomial-time computable, one can efficiently compute a polynomial-sized circuit $C(\cdot, \cdot)$ that implements $\mathsf{Eval}(\cdot, \cdot)$. Let $\mathsf{wires}(C, \mathsf{pp}, x)$ be the $O(|C|)$-length list of wire assignments taken on by C on input (pp, x). Let $\overline{\mathsf{Eval}}$ be defined to take as input pp from Setup and x from the support of the distribution of $\mathsf{SampleInput}$, and output $(\mathsf{Eval}(\mathsf{pp}, x), \mathsf{wires}(C, \mathsf{pp}, x))$.

The corresponding verifier $\overline{\mathsf{Verify}}$ takes as input the public parameters pp, an input x, and an output which consists of y and alleged wire assignments L. $\overline{\mathsf{Verify}}$ is implemented by $O(|C|)$ intermediate parallel processors, each of which takes as input a local segment of L defining the input and output wires of a single gate of C. It checks that these input and output wires are consistent; if not, it outputs $\perp$. A final processor checks if any intermediate processor outputs $\perp$, and outputs false if so. Otherwise, it outputs true if and only if the output input wires and output wires of L match x and y respectively.

Correctness. If y is indeed the output of $\mathsf{Eval}(\mathsf{pp}, x)$, the wire assignments $\mathsf{wires}(C, \mathsf{pp}, x)$ will be consistent. Therefore, $\overline{\mathsf{Verify}}(\mathsf{pp}, x, (y, \mathsf{wires}(C, \mathsf{pp}, x)))$ will output true.

Soundness. $\overline{\mathsf{Eval}}$ is sound with no probability of error. If y' is not the output of $\mathsf{Eval}(\mathsf{pp}, x)$, then C on input (pp, x) does not output y'. Any alleged wire assignment L with input wires (pp, x) and output wires y' must have an inconsistency at some gate of C. The intermediate processor that verifies this gate will output $\perp$, and therefore $\overline{\mathsf{Verify}}(\mathsf{pp}, x, (y', L))$ will output false.

Finally, observe that $\overline{\mathsf{DF}}$ inherits $(1/q(\lambda))$-weak unpredictability from DF, as its output includes $\mathsf{Eval}(\mathsf{pp}, x)$. ◻

Given a delay function $\overline{\mathsf{DF}} = (\overline{\mathsf{Setup}}, \overline{\mathsf{SampleInput}}, \overline{\mathsf{Eval}})$, we define $\mathsf{DF}' = (\mathsf{Setup}', \mathsf{SampleInput}', \mathsf{Eval}')$ as follows. Let $\ell = \lambda q(\lambda)$.

$\underline{\mathsf{Setup}'(1^\lambda)}$:

- For each $i \in [\ell]$, let $\mathsf{pp}_i \leftarrow \overline{\mathsf{Setup}}(1^\lambda)$.
- Output $\mathsf{pp}' = \mathsf{pp}_1 || \ldots || \mathsf{pp}_\ell$.

$\underline{\mathsf{SampleInput}'(1^\lambda, \mathsf{pp}')}$:

- For each $i \in [\ell]$, let $x_i \leftarrow \overline{\mathsf{SampleInput}}(1^\lambda, \mathsf{pp}_i)$.
- Output $x_1 || \ldots || x_\ell$.

$\underline{\mathsf{Eval}'(\mathsf{pp}', x_1 || \ldots || x_\ell)}$:

– For each $i \in [\ell]$, let $(y_i, L_i) \leftarrow \overline{\mathsf{Eval}}(\mathsf{pp}_i, x_i)$.
– Output $(y_1, L_1)||\ldots||(y_\ell, L_\ell)$.

Lemma 2. *Let* $\overline{\mathsf{DF}} = (\overline{\mathsf{Setup}}, \overline{\mathsf{SampleInput}}, \overline{\mathsf{Eval}})$ *be a delay function that is* $(1/q(\lambda))$*-weakly unpredictable in parallel time* $t_1(\cdot)$ *and verifiable in parallel time* $t_2(\cdot)$ *by an algorithm* $\overline{\mathsf{Verify}}$*. Then* DF' *as defined above is a* $(t_1 - t_2)$*-sequential delay function.*

Our proof follows the proof of amplification of a weak one-way function to a full one-way function from, e.g., [PS10, Theorem 35.1].

Proof. Assume towards a contradiction that DF' is not a $(t_1 - t_2)$-sequential delay function. Formally, this means there exist a polynomial $z(\lambda)$ and adversaries $\mathcal{A}'_0$ and $\mathcal{A}'_1$, where:

– $\mathcal{A}'_0$ runs in polynomial parallel time to produce an advice string given the public parameters, and
– $\mathcal{A}'_1$ runs in parallel time at most $(t_1 - t_2)$ to produce a prediction of the delay function output.

such that the following holds:

$$\Pr_{\substack{\mathsf{pp}' \leftarrow \mathsf{Setup}'(1^\lambda) \\ x' \leftarrow \mathsf{SampleInput}'(\mathsf{pp}')}} \begin{bmatrix} \alpha \leftarrow \mathcal{A}'_0(1^\lambda, \mathsf{pp}') \\ y' \leftarrow \mathcal{A}'_1(1^\lambda, \mathsf{pp}', x', \alpha) \wedge y' = \mathsf{Eval}'(\mathsf{pp}', x') \end{bmatrix} \geq \frac{1}{z(\lambda)}.$$

We'll use $(\mathcal{A}'_0, \mathcal{A}'_1)$ to construct an adversary that breaks $\overline{\mathsf{DF}}$. Now, consider the following adversaries $\mathcal{A}_0^{(i)}, \mathcal{A}_1^{(i)}$ for $i \in [\ell]$, acting as follows on inputs $(1^\lambda, \mathsf{pp})$ and $(1^\lambda, \mathsf{pp}, x, \alpha')$ respectively.
$\mathcal{A}_0^{(i)}$ does the following:

– For each $j \neq i$, $j \in [\ell]$, choose $\mathsf{pp}_j \leftarrow \overline{\mathsf{Setup}}(1^\lambda), x_j \leftarrow \overline{\mathsf{SampleInput}}(1^\lambda, \mathsf{pp}_j)$. Denote $\mathsf{pp}' = \mathsf{pp}_1||\ldots||\mathsf{pp}_\ell$.
– Let $\alpha \leftarrow \mathcal{A}'_0(1^\lambda, \mathsf{pp}')$.
– $\mathcal{A}_0^{(i)}$ outputs as advice $\alpha' := (\alpha, \mathsf{pp}', \{x'_j\}_{j \neq i})$.

$\mathcal{A}_1^{(i)}$ now computes:

– $x' := x'_1||\ldots||x'_\ell$, where $x'_i := x$.
– $(y_1, L_1)||\ldots||(y_\ell, L_\ell) \leftarrow \mathcal{A}'_1(1^\lambda, \mathsf{pp}', x', \alpha)$.
– If $\overline{\mathsf{Verify}}(\mathsf{pp}, x, (y_i, L_i)) = 1$ then output y_i. Else, output $\perp$.

With these in mind, consider the following adversaries $\mathcal{A}_0(1^\lambda, \mathsf{pp})$ and $\mathcal{A}_1(1^\lambda, \mathsf{pp}, x)$. $\mathcal{A}_0$ is the preprocessing adversary running in arbitrary polynomial parallel time, and for each $i \in [\ell]$ it runs $\mathcal{A}_0^{(i)}$ to produce advice α_i. It outputs the concatenation of these advice strings $\alpha_1||\ldots||\alpha_\ell$. $\mathcal{A}_1$ is the sequentially bounded adversary which receives the public parameters, challenge input, and advice string from $\mathcal{A}_0$. For each $i \in [\ell]$, $\mathcal{A}_1$ runs *in parallel* $2\ell \cdot \lambda \cdot z(n)$ iterations of $\mathcal{A}_1^{(i)}(1^\lambda, \mathsf{pp}, x, \alpha_i)$ with independent randomness. If any of them outputs a value which is not $\perp$, output it.

Our goal is to prove that $(\mathcal{A}_0, \mathcal{A}_1)$ succeeds in breaking $(1/q(\lambda))$-weak unpredictability of $\overline{\mathsf{DF}}$, where $\mathcal{A}_1$ runs in parallel time at most $t_1(\lambda)$. Notice first that indeed $\mathcal{A}_1$ can be implemented in $t_1(\lambda)$ parallel time, as each invocation of $\mathcal{A}_1^{(i)}$ for any i requires $(t_1(\lambda) - t_2(\lambda)) + t_2(\lambda)$ parallel time, and all such invocations are run in parallel.

Now, consider the following sets G_i for all $i \in [\ell]$.

$$G_i = \left\{ (\mathsf{pp}, x) \;\middle|\; \Pr_{\alpha \leftarrow \mathcal{A}_0^{(i)}(1^\lambda, \mathsf{pp})} [\mathcal{A}_1^{(i)}(1^\lambda, \mathsf{pp}, x, \alpha) \neq \perp] \geq \frac{1}{2\ell \cdot z(n)} \right\}$$

In the above, the randomness is over the internal randomness of $\mathcal{A}_i$.

Lemma 3. *There exists $i \in [\ell]$ such that*

$$\Pr_{\substack{\mathsf{pp}\leftarrow\overline{\mathsf{Setup}}(1^\lambda)\\ x\leftarrow\overline{\mathsf{SampleInput}}(1^\lambda)}}[(\mathsf{pp},x)\in G_i] \geq 1-\frac{1}{2q(\lambda)}.$$

We call i the guaranteed coordinate.

Proof. Assume towards a contradiction that for all $i \in [\ell]$,

$$\Pr_{\substack{\mathsf{pp}\leftarrow\overline{\mathsf{Setup}}(1^\lambda)\\ x\leftarrow\overline{\mathsf{SampleInput}}(\mathsf{pp})}}[(\mathsf{pp},x)\in G_i] < 1-\frac{1}{2q(\lambda)}.$$

Let S' be the event that $\mathcal{A}_1'$ outputs $(y_1,L_1)||\dots||(y_\ell,L_\ell)$ such that

$$\overline{\mathsf{Verify}}(\mathsf{pp},x,(y_1,L_1)||\dots||(y_\ell,L_\ell))=1.$$

In the following, the randomness is over $\mathcal{A}_0'$'s choice of pp' and x', and the internal randomness of $\mathcal{A}_1'$. We have that

$$\Pr[S'] = \Pr[S' \wedge \forall i\in[\ell],(\mathsf{pp}_i,x_i)\in G_i] + \Pr[S' \wedge \exists i\in[\ell],(\mathsf{pp}_i,x_i)\notin G_i].$$

Focusing on the first term, we have that

$$\begin{aligned}\Pr[S' \wedge \forall i\in[\ell],(\mathsf{pp}_i,x_i)\in G_i] &\leq \prod_{i\in[\ell]}\Pr[(\mathsf{pp}_i,x_i)\in G_i]\\ &< \left(1-\frac{1}{2q(\lambda)}\right)^\ell\\ &< e^{-\lambda},\end{aligned}$$

since $\ell=\lambda q(\lambda)$. For the second term, we have that

$$\begin{aligned}\Pr[S' \wedge \exists i\in[\ell],x_i\notin G_i] &\leq \sum_{i\in[\ell]}\Pr[S' \wedge (\mathsf{pp}_i,x_i)\notin G_i]\\ &\leq \sum_{i\in[\ell]}\Pr[S' \mid (\mathsf{pp}_i,x_i)\notin G_i]\\ &< \frac{\ell}{2\ell z(\lambda)} \qquad (2)\\ &= \frac{1}{2z(\lambda)},\end{aligned}$$

where Equation (2) is by definition of $G_j, j\in[\ell]$, since if (pp_j,x_j) is not in G_j, then $\mathcal{A}_1^{(j)}$ outputs $\perp$ w.p. at least $1-\frac{1}{2\ell z(\lambda)}$. Thus, conditioned on $(\mathsf{pp}_j,x_j)\notin G_j$, S' occurs w.p. at most $\frac{1}{2\ell z(\lambda)}$. In total we have that $\Pr[S'] \leq e^{-\lambda}+\frac{1}{2z(\lambda)} < \frac{1}{z(\lambda)}$. This concludes the proof. □

Claim. It holds that

$$\Pr_{\substack{\mathsf{pp}\leftarrow\overline{\mathsf{Setup}}(1^\lambda)\\ x\leftarrow\overline{\mathsf{SampleInput}}(1^\lambda)\\ \alpha\leftarrow\mathcal{A}_0(1^\lambda,\mathsf{pp})}}\left[\mathcal{A}_1(1^\lambda,\mathsf{pp},x,\alpha)\neq\perp\right] > 1-\frac{1}{q(\lambda)}.$$

Proof. Let S be the event that $\mathcal{A}_1(1^\lambda, \mathsf{pp}, x, \alpha) \neq \perp$. Let i be the guaranteed coordinate from Lemma 3. We have that

$$\begin{aligned}
\Pr[\neg S] &\leq \Pr[\neg S \wedge (\mathsf{pp}, x) \in G_i] + \Pr[\neg S \wedge (\mathsf{pp}, x) \notin G_i] \\
&\leq \Pr[\neg S \wedge (\mathsf{pp}, x) \in G_i] + \Pr[(\mathsf{pp}, x) \notin G_i] \\
&\leq \Pr[\neg S \wedge (\mathsf{pp}, x) \in G_i] + \frac{1}{2q(\lambda)} \\
&\leq \left(1 - \frac{1}{2\ell z(\lambda)}\right)^{2\ell\lambda z(\lambda)} \qquad (3) \\
&\leq e^{-\lambda} + \frac{1}{2q(\lambda)} \\
&< \frac{1}{q(\lambda)},
\end{aligned}$$

where Equation (3) is by definition of G_i and the parallel invocations of $\mathcal{A}_1^{(i)}$. ☐

We have thus proved that with probability at least $1 - \frac{1}{q(\lambda)}$, $\mathcal{A}_1(1^\lambda, \mathsf{pp}, x, \alpha)$ outputs a value $y_i \neq \perp$. Condition on this event and denote by i the adversary $\mathcal{A}_1^{(i)}$ that realized this event. Thus in particular for the string (y_i, L_i) output by $\mathcal{A}'_1$ in the invocation of $\mathcal{A}_i$, $\overline{\mathsf{Verify}}(\mathsf{pp}, x, (y_i, L_i)) = 1$. The soundness of $\overline{\mathsf{Verify}}$ implies that indeed $y_i = \overline{\mathsf{Eval}}(\mathsf{pp}, x)$.

As previously noted, $\mathcal{A}_0$ and $\mathcal{A}_1$ run in polynomial total time, and $\mathcal{A}_1$ runs in at most $t_1(\lambda)$ parallel time. This contradicts the premise of Lemma 2, thus proving the lemma. ☐

Theorem 4 (Delay function hardness amplification). *Let* DF *be a* $(1/q(\lambda))$*-weak delay function that is* $(1/q(\lambda))$*-weakly unpredictable in parallel time* $t(\lambda)$*. Then there exists a delay function* DF' *that is (*fully*)* $(t(\lambda) - O(1))$*-unpredictable.*

Proof. The proof follows from Lemmas 1 and 2. That is, Lemma 1 shows that from DF, one can construct $\overline{\mathsf{DF}}$ which is verifiable and weakly unpredictable with the same delay parameter. Lemma 2 shows that from $\overline{\mathsf{DF}}$, one can construct DF' which is fully unpredictable but whose delay parameter is $t - t'$, where t' is the parallel runtime of the verifier of $\overline{\mathsf{DF}}$. Recall that this verifier checks consistency of the wire assignments of the circuit computing the evaluation function of DF, and it does so in parallel. Therefore, the number of sequential steps it requires is the number of steps to evaluate a single gate of this circuit, which is $O(1)$.

Therefore, DF' is a delay function that is $(t(\lambda) - O(1))$-unpredictable. ☐

5.4 Main theorem

Our main theorem now follows from applying Theorem 3 followed by Theorem 4. It states that a protocol circumventing Cleve's lower bound implies a delay function that is unpredictable except with negligible advantage.

Theorem 5. *Let* Π *be a an* $r(\lambda)$*-round* $\epsilon(\lambda)$*-consistent* n*-party coin-tossing protocol with bias* $< \frac{\epsilon}{10r(\lambda)}$ *under the corruption of a dishonest majority, where each corrupted party's computation in each round of the protocol runs in parallel time at most* $t(\lambda)$*. Then there exists a* $(t(\lambda) - O(1))$*-sequential delay function.*

6 Delay functions imply n-party distributed randomness beacons

In the previous section (Section 5), we showed that any fair coin tossing protocol secure against a dishonest majority implies the existence of a delay function. This delay function is weaker than those previously used to construct distributed randomness beacons, in several senses (see Section 4 for a more comprehensive discussion):

– The distribution over which the function is unpredictable may not be the uniform distribution
– The delay parameter is fixed and cannot be specified
– The output of the delay function is not efficiently verifiable

A natural question is whether one can construct a distributed randomness beacon using the notion of a delay function implied by circumventing Cleve's protocol. In this section, we answer this in the affirmative. In fact, we show that even a delay function that can be predicted with $1 - 1/\mathsf{poly}(\lambda)$ probability can be used to construct a distributed randomness beacon that is computationally indistinguishable from random except with *negligible* advantage.

We also remove the reliance on a public bulletin board or consensus protocol present in prior work. Instead, we observe that Byzantine Broadcast is possible even with a dishonest majority, and we use such a protocol as a subroutine for participants to broadcast their randomness contributions. Our protocol largely follows the structure of the Unicorn protocol [LW15b].

```
$n$-party distributed randomness beacon
---------------------------------------
**Setup phase:**
Our protocol requires no setup.

**Preprocessing phase:**
Each party $P_i$ draws $\mathsf{pp}_i \leftarrow \mathsf{Setup}(1^\lambda)$ and $x_i \leftarrow \mathsf{SampleInput}(1^\lambda, \mathsf{pp}_i)$
Let $\beta_i = (\mathsf{pp}_i, x_i)$

**Posting phase:**
For $i \in [n]$, do the following in sequence:
    At round $r = 0, P_i$ initiates, as the sender, an instance of $\mathsf{BB}_{\mathsf{DS}}$ with input $(\mathsf{pp}_i, x_i)$
    At rounds $r \in [1, n], P_i$ runs $n$ parallel instances of $\mathsf{BB}_{\mathsf{DS}}$ where $P_j$ is the sender of instance $j$
The posting phase concludes as soon as either $\Delta_\lambda$ time has elapsed, or $n$ rounds have elapsed.

**Finalization phase:**
$P_i$ computes the following:
    Let $((\mathsf{pp}^i_1, x^i_1), \ldots, (\mathsf{pp}^i_n, x^i_n))$ be $P_i$'s outputs from the $n$ instances of $\mathsf{BB}_{\mathsf{DS}j}$ with sender $j \in [n]$
    $y_j \leftarrow \mathsf{Eval}(\mathsf{pp}^i_j, x^i_j)\ \forall j \in [n]$
    $\Omega_i \leftarrow H(y_1 || \ldots || y_n)$
$P_i$ outputs $\Omega_i$.
```

Fig. 3. n-party randomness beacon instantiated using a delay function $\mathsf{DF} = (\mathsf{Setup}, \mathsf{SampleInput}, \mathsf{Eval})$

Theorem 6. *The protocol depicted in Figure 3 satisfies termination and agreement, i.e., there exists a value r such that all honest players produce r as output and terminate.*

Proof. Denote by $f < n$ the number of corrupt players. Since $\mathsf{BB}_{\mathsf{DS}}$ solves Byzantine Broadcast in the presence of f corruptions, for any $f < n$, we get that for all honest parties i, j, it holds that $(x^i_1, \ldots, x^i_n) = (x^j_1, \ldots, x^j_n)$. This clearly implies, by the behaviour of the algorithm, that all honest parties produce the same output, as required. □

Theorem 7. *Let* DF *be a delay function that is unpredictable by* $t(\cdot)$*-sequentially bounded adversaries. Suppose that* Δ_λ *is an upper bound on the amount of wall-clock time that the adversary takes to compute a* $t(\lambda)$*-sequential computation, and suppose that the posting phase requires less than* Δ_λ *wall-clock time. Then* DRB_n *(Figure 3), when instantiated with* DF*, satisfies indistinguishability against a* $t(\cdot)$*-sequentially bounded adversary corrupting up to* $n-1$ *parties in the random oracle model.*

Proof. Suppose for the sake of contradiction $(\mathcal{A}_0, \mathcal{A}_1)$ corrupts $n-1$ parties and succeeds in breaking indistinguishability of DRB_n, where $\mathcal{A}_1$ runs in parallel time at most $t(\lambda)$.

By agreement, all honest parties output the same value. Let P_j be an honest party. The output of P_j is $\Omega_j = H(y_1||\dots||y_n)$, where $y_j \leftarrow \mathsf{Eval}(\mathsf{pp}_j, x_j)$, and $x_j \leftarrow \mathsf{SampleInput}(\mathsf{pp}_j, x_j)$. Now, observe that if $\mathcal{A}_1$ never queried $y_1||\dots||y_n$ to its random oracle, $H(y_1||\dots||y_n)$ is a random value and $\mathcal{A}_1$ cannot distinguish it from random. Therefore, $\mathcal{A}_1$ must have queried $y_1||\dots||y_n$.

We will now specify $(\mathcal{B}_0, \mathcal{B}_1)$, an adversary that engages in $\mathcal{G}^{\text{sequential}}$ to break sequentiality of DF. Recall that $\mathcal{B}_0$ is given public parameters and produces an advice string in arbitrary polynomial parallel time; $\mathcal{B}_1$ is given this advice string and must predict the output of the delay function in $t(\lambda)$ parallel time. $\mathcal{B}_0$ receives public parameters $\mathsf{pp}_j \leftarrow \mathsf{Setup}(1^\lambda)$. $\mathcal{B}_0$ simulates $\mathcal{A}_0$, except that it uses the challenge public parameters pp_j. It outputs the advice string output by $\mathcal{A}_0$.

$\mathcal{B}_1$ receives a challenge input $x_j \leftarrow \mathsf{SampleInput}(1^n)$ and the advice string from $\mathcal{B}_0$, and $\mathcal{B}_1$ simulates an honest party P_j that contributes x_j to DRB_n. $\mathcal{B}_1$ also simulates $\mathcal{A}_1$ in running DRB_n. At the end of this simulation, $\mathcal{B}_1$ chooses one of $\mathcal{A}_1$'s random oracle queries at random, chooses a random contiguous subsequence of it, and outputs it. We claim that $\mathcal{B}_1$ succeeds in breaking unpredictability of DF.

Since $\mathcal{A}_1$ queried $y_1||\dots||y_n$, and $\mathcal{A}_1$ made polynomially many queries, $\mathcal{B}_1$ succeeds in outputting $y_j = \mathsf{Eval}(\mathsf{pp}_j, x_j)$ with inverse polynomial probability.

Finally, we note that $\mathcal{B}_1$ simply runs $\mathcal{A}_1$, so $\mathcal{B}_1$ runs in $t(\lambda)$ parallel time. □

Remark 2. Note that we can avoid reliance on the random oracle model if we wish to achieve the weaker notion of *unpredictability* of the beacon output, rather than indistinguishability from random.

Corollary 2. *Let* $q(\cdot)$ *be any polynomial, and suppose there exists a* $(1/q(\lambda))$*-weak delay function that is* $(1/q(\lambda))$*-weakly unpredictable by* $t(\cdot)$*-sequentially bounded adversaries. Then there exists an* n*-party distributed randomness beacon satisfying indistinguishability against a* $(t(\cdot) - O(1))$*-sequentially bounded adversary corrupting up to* $n-1$ *parties in the random oracle model.*

Proof. Theorem 4 states that a $(1/q(\lambda))$-weak delay function can be boosted to a fully-fledged delay function that is unpredictable by a $(t(\lambda) - O(1))$-sequentially bounded adversary. Combining this with Theorem 7 gives the result. □

7 Concluding discussion

Our work shows that, at a fundamental level, generating randomness in a distributed manner without relying on an honest majority assumption requires using some notion of a delay function. In contrast to previous work, we also show that a delay function is sufficient for distributed randomness generation under a dishonest majority, even without the use of strong communication assumptions like reliable broadcast or a public bulletin board.

In practice, these stronger communication assumptions may be justifiable, even though they themselves typically require an honest majority to achieve. The justification is that practical protocols realizing reliable broadcast or a public bulletin board are usually accountable, with malicious or faulty nodes being identifiable in the event of a safety violation. By contrast, violations of key security properties for randomness protocols, namely prediction or biasing attacks by a dishonest majority, appear difficult to detect. Hence, it may make sense in some scenarios to build a dishonest-majority randomness protocol on top of an honest-majority consensus protocol (obtaining significant efficiency improvements in the process).

Finally we note that our analysis, and all in the literature using delay functions that we are aware of, assume network synchrony (with a known maximal delay in message delivery). Intuitively, it appears difficult to use delay functions with a partial-synchrony or fully-asynchronous network model, as the adversary can always delay messages for long enough to compute any delay function. Remaining in synchrony, another possible setting is the 'clock drift' model, where the rounds in which honest players initiate the protocol can differ arbitrarily. This mimics some aspects of the asynchronous model while avoiding message delay attacks by the adversary. We leave a complete analysis of randomness generation without network synchrony to future work.

Acknowledgements

Miranda Christ was partially supported by a Google CyberNYC grant, an Amazon Research Award, and NSF grant CCF-2312242. Joseph Bonneau was supported by a16z crypto and NSF grant CNS-2239975. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, a16z, Google or any other supporting organization.

References

[AMZ24] Shweta Agrawalr, Giulio Malavolta, and Tianwei Zhang. "Time-Lock Puzzles from Lattices". In: *CRYPTO*. 2024.

[And+14] Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Łukasz Mazurek. "Fair two-party computations via bitcoin deposits". In: *Financial Crypto*. 2014.

[ARS24] Damiano Abram, Lawrence Roy, and Mark Simkin. "Time-Based Cryptography From Weaker Assumptions: Randomness Beacons, Delay Functions and More". In: *Cryptology ePrint Archive* (2024).

[BD21] Jeffrey Burdges and Luca De Feo. "Delay encryption". In: *Eurocrypt*. 2021.

[Bea+23] Donald Beaver, Konstantinos Chalkias, Mahimna Kelkar, Lefteris Kokoris Kogias, Kevin Lewi, Ladi de Naurois, Valeria Nicolaenko, Arnab Roy, and Alberto Sonnino. "STROBE: Stake-based Threshold Random Beacons". In: *Advances in Financial Technologies*. 2023.

[BK14] Iddo Bentov and Ranjit Kumaresan. "How to use Bitcoin to design fair protocols". In: *CRYPTO*. 2014.

[BL85] Michael Ben-Or and Nathan Linial. "Collective coin flipping, robust voting schemes and minima of Banzhaf values". In: *FOCS*. 1985.

[Blu83a] Manuel Blum. "Coin flipping by telephone a protocol for solving impossible problems". In: *ACM SIGACT* 15.1 (1983).

[Blu83b] Manuel Blum. "How to exchange (secret) keys". In: *ACM ToCS* 1.2 (1983).

[BN00] Dan Boneh and Moni Naor. "Timed commitments". In: *CRYPTO*. 2000.

[Bon+18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. "Verifiable Delay Functions". In: *CRYPTO*. 2018.

[CCB24] Miranda Christ, Kevin Choi, and Joseph Bonneau. "Cornucopia: Distributed Randomness at Scale". In: *Advances in Financial Technologies*. 2024.

[CD17] Ignacio Cascudo and Bernardo David. "SCRAPE: Scalable randomness attested by public entities". In: *ACNS*. 2017.

[CD20] Ignacio Cascudo and Bernardo David. "Albatross: publicly attestable batched randomness based on secret sharing". In: *Asiacrypt*. 2020.

[Cho+23] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. "Bicorn: An optimistically efficient distributed randomness beacon". In: *Financial Crypto*. 2023.

[Cho+85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. "Verifiable secret sharing and achieving simultaneity in the presence of faults". In: *FOCS* (1985).

[CHS05] Ran Canetti, Shai Halevi, and Michael Steiner. "Hardness amplification of weakly verifiable puzzles". In: *TCC*. 2005.

[Cle86] Richard Cleve. "Limits on the security of coin flips when half the processors are faulty". In: *TOC*. 1986.

[CMB23] Kevin Choi, Aathira Manoj, and Joseph Bonneau. "SoK: Distributed Randomness Beacons". In: *IEEE Security & Privacy*. 2023.

[CSS19] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. *Homomorphic Encryption Random Beacon*. Cryptology ePrint Archive, Paper 2019/1320. 2019.

[CZ23] Chao Chen and Fangguo Zhang. "Verifiable delay functions and delay encryptions from hyperelliptic curves". In: *Cybersecurity* 6.1 (2023).

[DB84] D Dolev and A Broder. "Flipping Coins in Many Pockets". In: *FOCS*. 1984.

[De +19] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. "Verifiable Delay Functions from Supersingular Isogenies and Pairings". In: *Asiacrypt*. 2019.

[DN92] Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *CRYPTO*. 1992.

[Dra24] Drand. *Drand*. https://drand.love/. 2024.

[DS83] Danny Dolev and H. Raymond Strong. "Authenticated Algorithms for Byzantine Agreement". In: *SIAM J. Comput.* 12.4 (1983).

[FM97] Matthew K Franklin and Dahlia Malkhi. "Auditable metering with lightweight security". In: *Financial Crypto*. 1997.

[Fre+21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. "Non-Malleable Time-Lock Puzzles and Applications". In: *TCC*. 2021.

[Gal+21] David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. "Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons". In: *Euro S&P*. 2021.

[GM82] Shafi Goldwasser and Silvio Micali. "Probabilistic Encryption and How to Play Mental Poker Keeping Secret All Partial Information". In: *STOC*. 1982.

[GS98] David M Goldschlag and Stuart G Stubblebine. "Publicly verifiable lotteries: Applications of delaying functions". In: *Financial Crypto*. 1998.

[GSX20] Zhaozhong Guo, Liucheng Shi, and Maozhi Xu. "SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability". In: *IEEE Access* (2020).

[Haa99] Mads Haahr. "random.org: Introduction to Randomness and Random Numbers". https://www.random.org/mads/. 1999.

[HS66] F. C. Hennie and Richard Edwin Stearns. "Two-Tape Simulation of Multitape Turing Machines". In: *J. ACM* 13.4 (1966).

[Kel+19] John Kelsey, Luís TAN Brandão, Rene Peralta, and Harold Booth. *A reference for randomness beacons: Format and protocol version 2*. Tech. rep. National Institute of Standards and Technology, 2019.

[KWJ23] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. *SoK: Public Randomness*. Cryptology ePrint Archive, Paper 2023/1121. 2023.

[LM23] Russell WF Lai and Giulio Malavolta. "Lattice-based timed cryptography". In: *CRYPTO*. 2023.

[LMR83] Michael Luby, Silvio Micali, and Charles Rackoff. "How to Simultaneously Exchange a Secret Bit by Flipping a Symmetrically-Biased Coin". In: *FOCS*. 1983.

[LW15a] Arjen K. Lenstra and Benjamin Wesolowski. *A random zoo: sloth, unicorn, and trx*. Cryptology ePrint Archive, Paper 2015/366. 2015.

[LW15b] Arjen K. Lenstra and Benjamin Wesolowski. "A random zoo: sloth, unicorn, and trx". In: *IACR Cryptol. ePrint Arch.* (2015), p. 366. URL: http://eprint.iacr.org/2015/366.

[Mic22] Yan Michalevsky. *Cryptosat launched Crypto1 — the first cryptographic root-of-trust in space*. https://medium.com/cryptosatellite/cryptosat-launches-crypto1-the-first-cryptographic-root-of-trust-in-space-37dcc324fe65. May 2022.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. "Homomorphic time-lock puzzles and applications". In: *CRYPTO*. 2019.

[Obe19] Daniel Oberhaus. "A Programmer Solved a 20-Year-Old, Forgotten Crypto Puzzle". In: *Wired* (Apr. 2019).

[Pie18] Krzysztof Pietrzak. “Simple Verifiable Delay Functions”. In: *ITCS*. 2018.

[PS10] Rafael Pass and Abhi Shelat. “A course in cryptography”. In: *Theoretical Foundation of Cryptography* (2010).

[Qia17] Youcai Qian. *Randao: Verifiable Random Number Generation*. `randao.org/whitepaper/Randao_v0.85_en.pdf`. 2017.

[Rab83] Michael O. Rabin. “Transaction protection by beacons”. In: *Journal of Computer and System Sciences* (1983).

[RG22] Mayank Raikwar and Danilo Gligoroski. “SoK: Decentralized randomness beacon protocols”. In: *Australasian Conference on Information Security and Privacy*. 2022.

[RSW96] Ronald Rivest, Adi Shamir, and David Wagner. *Time-lock puzzles and timed-release crypto*. Tech. rep. Massachusetts Institute of Technology, 1996.

[Sch+23] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. “RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness”. In: *NDSS*. 2023.

[Sch99] Berry Schoenmakers. “A simple publicly verifiable secret sharing scheme and its application to electronic voting”. In: *CRYPTO*. 1999.

[Sha49] Claude E. Shannon. “The synthesis of two-terminal switching circuits”. In: *Bell Syst. Tech. J.* 28.1 (1949).

[Sta96] Markus Stadler. “Publicly verifiable secret sharing”. In: *Eurocrypt*. 1996.

[Syt+17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. “Scalable bias-resistant distributed randomness”. In: *IEEE Security & Privacy*. 2017.

[Wes19] Benjamin Wesolowski. “Efficient Verifiable Delay Functions”. In: *Eurocrypt*. 2019.

[Yak+20] David Yakira, Avi Asayag, Ido Grayevsky, and Idit Keidar. “Economically Viable Randomness”. In: *CoRR* (2020).