An Unbiassable Random Oracle for Blockchains

A.W. Roscoe, Pedro Antonino and Jonathan Lawrence

Keywords: Blockchain, Unbiassable Random Oracle, Delay Encryption, Stochastic Certainty

Abstract: We introduce a new way of constructing a random oracle that generates a random number at chosen events — such as the generation of a new block — in a blockchain. We make fairly standard assumptions about the distribution of good and bad behaviour of contributing agents and do not require any dependably trustworthy player, internal or external. We give two variants on this oracle. These cannot be meaningfully biassed by any feasible coalition of bad agents, whether by their action or inaction.

1 INTRODUCTION

Blockchains frequently need to choose agents or groups of agents fairly and randomly, for example to create blocks in consensus mechanisms such as proof of stake or to verify proposed blocks. The choices made can affect the integrity of the chain and the allocation of rewards and penalties. If bad agents could realistically bias the mechanism behind these choices, they could potentially get their allies selected for such roles and thus corrupt the chain because more bad agents are selected than would be possible under a fair selection mechanism.

Ideally, we think of a blockchain as being a recursive and stochastic construction of a substantial data structure under which the integrity of the chain up to level N guarantees it up to level $N+1$. Having an unbiassable random oracle is a crucial part of this inductive process.

In this paper, we introduce such an oracle that addresses the inevitable possibility that bad agents may be selected to participate in oracle generation and that they potentially can bias the oracle by performing their late-stage functions or not. We ensure that if an agent chooses not to release crucial late-stage information, it is available through delay mechanisms, and therefore the oracle value is completely determined before anyone discovers it.

In the next section, we describe the background to this work: trust assumptions about the agents participating in the blockchain, the need for a regular supply of random numbers, existing approaches to the problem and the potential ways in which bad agents can try to influence oracle values. We then describe our solution, which depends on multi-party computation in two phases.

Section 3 presents the combinatorial analysis that underpins the security of our protocol. We then present two versions of our random oracle, the second being an extension and improvement on the first. First in section 4 we introduce the basic idea behind our random oracle, albeit in a naïve form which can be biassed by bad agents (by selective non-participation). Then, section 5 describes how delay encryption can augment the protocol to make our oracle unbiassable.

This paper is an expansion and deeper analysis of some of the ideas in the unpublished technical report (Roscoe and Chen, 2019). Related work in the area is discussed in section 6, and finally we summarise the main contributions of the paper in section 7.

2 BACKGROUND

Blockchains only make sense in environments where there is no agent that all users completely trust. They are thus decentralised systems in which trust emerges from collective actions and mutual checking. This paper applies to all types of blockchain but principally to those that choose block creators by a lottery. There are real advantages to being chosen to create blocks:

1. Financial rewards to block creators in many blockchains.
2. Ability to choose which transactions are included can give advantages such as gaining transaction fees and affecting markets.
3. Blocks will frequently contain material that con-

tribute to the security of the chain, with block creators having the duty to implement this security. While such mechanisms must be tolerant of some blocks being built by bad agents, they will typically require sufficient to be good and so the ability to increase the proportion of blocks built by the bad might well defeat the security that the mechanisms seek to establish. So, someone trying to create, for example, a successful fork may well need to be (along with allies) the creator of a group of blocks.

We believe that there is an advantage in choosing relatively recent block creators to perform some security related roles in maintaining a blockchain. Such agents will be motivated to protect the rewards they are establishing in association with their blocks; they may well have escrowed a good-behaviour deposit; and they have already been selected with great care. They are more likely than general agents to be awake and well informed about the current state of the chain.

Of course, some of these roles will be in helping to select the next block creator and validating the resulting block. Overall, we can expect the fundamental security proof of the blockchain to follow the following lines:

1. The proof is up to stochastic certainty: the likelihood of the blockchain not developing correctly is sufficiently small as to be ignored.
2. For this we need to understand the probability distribution of bad agents (and so possible bad behaviour) in the pool of agents in general and the block creators in particular.
3. The distribution of each block creator in turn will — in PoS models — depend on the correctness of the random oracle that chooses the nominee and the distributions amongst those operating the oracle. If the random oracle can be trusted by all to be genuinely random, then the likelihood of a good agent being chosen will exactly reflect the proportion of good ones amongst those selected from.
4. Thus, the keystone upon which the analysis depends will be an inductive argument that the block creators' goodness is no worse than a well understood distribution — most likely binomial — that we can analyse.
5. From this, we can draw conclusions about how large a pre-determined group of the block creators is required to make any function reliable enough for stochastic certainty.

3 COMBINATORIAL THRESHOLDS

Note that where many functions are required to be stochastically certain, then it is less certain that they absolutely all will be. So, for it to be certain up to 10^{-12} that an object requiring 10^6 individual actions to work, we would naturally require each of them to be certain up to 10^{-18}. This type of reasoning only works where an attacker only gets one or at most a few tries to break a stochastic barrier, which really means we must prevent searching from being a beneficial activity.

In this paper, we make the traditional distinction between good and bad agents, where the former follow protocol precisely: performing the tasks laid down accurately and keeping the secrets it is supposed to. We suppose that successive block creators are chosen from a population who bid to produce blocks, with the probability that one is chosen being proportionate to the size of its bid. Of course, that process has to be managed effectively in such a way that good block creators are motivated to take part and are therefore in a suitable majority amongst the competition. This choice will be made by the oracle, and successive choices are independent.

Note that the choice allows repetitions: a fresh choice is made from (potentially) the same population each time. It follows that where the blockchain ordains that a group of block creators such as those from block M to block $M+r-1$ are delegated a task, then the same agent might be picked several times; so while there may be less than r distinct agents involved, in total there will be r votes.

If the chance of each individual block creator being bad is no more than p, the chance of less than or equal to s of the r being bad is the sum of $p^k(1-p)^{r-k}C(r,k)$ for $k=0$ to s, where $C(r,k)$ is the usual combinatorial coefficient. It is well known that this binomial distribution is closely approximated by the normal one with mean rp and variance $rp(1-p)$. For the purposes of this paper, we will therefore estimate this sum by the cumulative normal distribution to s, which in turn can be calculated as a function of $(s-\mu)/\sigma$ where μ is the mean rp and σ is the standard deviation $\sqrt{rp(1-p)}$.

It follows that for any p and r, and any small $\varepsilon>0$ representing a tolerance for stochastic certainty, we can sometimes calculate numbers s such that, for example:

1. It is stochastically certain that r samples from the distribution do not contain as many as s bad agents, so any s (even when selected from the r by the enemy) must include at least one good one.

		ε value (given as $log_{10}\varepsilon$)					
		-3	-6	-9	-12	-15	-18
	equiv. M_σ	3.0902	4.76	6.002	7.03	7.943	8.759
p	0.3333	77	182	289	396	505	614
	0.25	29	68	109	149	190	231
	0.2	17	41	65	88	113	137
	0.15	10	24	38	52	66	80
	0.1	6	13	21	28	36	44
	0.05	3	6	9	12	15	18

Table 1: The table shows how many participants are needed to ensure that less than half are bad (with certainty $1-\varepsilon$), for each given likelihood p of any individual agent being bad. For each ε value, it also gives the equivalent multiple M_σ of the standard deviation σ, above the mean, that is exceeded with probability ε (see the appendix for further details).

2. It is stochastically certain that any s chosen from r must include more than half the good agents amongst the r.

In other papers we will illustrate how ensuring that the above limits apply can make blockchain features secure.

When $p < 0.5$, we can similarly calculate a minimum r such that with stochastic certainty the number of bad choices in r is strictly less than $r/2$, or less conservatively make the probability of this limit being breached below larger thresholds.

Table 1 allows us to judge the cost of using one of our oracle models introduced later. The formulae used for this calculation are given in the appendix.

In all these calculations we will normally assume that p is bounded above by $1/3$ as this corresponds to the limit for Byzantine agreement. It will be apparent that there are considerable benefits if one can justify using a yet smaller p.

4 A NAÏVE ORACLE PROTOCOL

A random oracle should produce a bit string V, of specified length, that all users of the blockchain agree on, on the occurrence of some defined event E that all can detect — such as a time, a specified elapsed time since the timestamp on some block, or an external event. No one should have any meaningful information about V before E, but it should be available to all shortly after E. We are only concerned with oracle values that are generated within the blockchain. These are necessary if there is no sufficiently trusted external source or collectively trusted group of sources: an example of the latter would be a set of random number beacons that are combined together (e.g. by xor) where all agree that at least one of them is trustworthy and independent of the others.

This observation depends on the fact that given k values, at least one of which is both random (i.e. independent, uniformly distributed bits) and independent of the others, one can combine the k (for example by xor) so that the result is random. It does not matter that as many as $k-1$ may be attempts to subvert the process. We note that these latter bad values must be created without knowledge of the good values, otherwise they would not be independent. Therefore, values contributed by good participants must not be disclosed until after all values have been committed. Thus, in both multi-sourced external solutions and internally generated random number generators, there is a significant benefit in all contributors irrefutably committing their shares to ensure that these cannot be altered once they have seen some others.

The following is a prototype random oracle for a blockchain, where the agents running the blockchain are given additional tasks to support it.

1. Well before the event E that will trigger the oracle, a set of contributors is chosen. These must be distinct agents and it must be stochastically certain that at least two of them are good.
2. Each A amongst these contributing agents is expected to commit a random share S_A it has created by placing $hash(A,E,S_A)$ in the chain. It might do this via a special transaction or A might place it while creating a block. Thus, when a second agent is told S_A this can be checked against the commitment. If any A does not commit a value by some block clearly before E, then its contribution is excluded.
3. On the occurrence of E, A places S_A on its bulletin board, so all agents can collect all the S_A, check their integrity and form the output value V.
4. Before using V, an agent should create or quote a certificate for its validity.

This protocol works provided at least two distinct good agents participate, and all the nominated agents who commit a value deliver their S_A values correctly[1]. The reason for requiring two good agents is to ensure that *no one* knows the oracle value V in advance: if there were only one, and it was aware of this, and knew the behaviour of the bad agents involved, it might know V in advance. It follows that in selecting agents to contribute shares, we should avoid choosing duplicates and yet allow for this in the analysis. We can ensure that at least two good agents are present

[1]Enforcing the distinctness of agents is difficult in the setting of a blockchain since the same agent can assume different identities while participating in the protocol. We are exploring different trust models and variations of our protocol to account for this.

by means of the concepts of stochastic certainty and stochastic forcing introduced earlier.

What this overlooks is the issue of non-participation. Under the assumptions about the presence of bad agents, any group of agents large enough to ensure that there are one or two good agents will very likely contain bad agents too. Commitment will have forced these to choose their shares so they cannot change them. However, they still have the option not to reveal them when E arrives. The consequences of this depend on several factors:

1. How dependable is the communication framework supporting the chain: can agents be reasonably penalised for failing to deliver material? By and large the more decentralised a system is, the less we can depend on universally dependable communication.
2. If not all shares are delivered, is that particular run aborted, or do the users of the oracle make do with the shares that they do have?

If it is reasonable to expect all to be in a position to deliver their obligations consistently rather than being blocked by communications issues and attacks by third parties, then possibly the protocol above is sufficient: non-delivery would clearly mark an agent as bad. Even so, the bad agents would be in a position to wait until all the good agents have revealed their own shares — at which point the bad ones in coalition will know the value V before everyone else does — and can opt for it to be delivered or not. So even in this case the choices made by the oracle are not unbiassable and the designer following this route needs to assess the associated impact.

If it is not reasonable to make this assumption, then potentially the enemy can permanently block the oracle or say yes or no to each and every value.

Exactly the same argument will apply for any oracle that works by getting some nominated participants to make a contribution to V where (most likely) they cannot predict each other's but are committed to what they will do and can prove they have done so faithfully.

To avoid the indefinite blocking of the oracle, one might define that the oracle be formed from the contributions of the parties who do publish their S_A or do whatever else the protocol calls them to do at the point where V is assembled. However, this would give a coalition of m bad agents contributing shares even more control as they would have 2^m choices to pick from.

5 DELAY AS SELF-OPENING COMMITMENT

The solution to the problem of selective non-participation must be to prevent bad agents from knowing the consequences of participating or not, in any meaningful way. While it is clear that an agent participating or not will certainly change the oracle value V, that action cannot be termed biasing if the distribution of V from the point of view of all participants remains the same.

Our objective is therefore to ensure that agents become fully committed to entering a share in the oracle before anyone can have any idea what V will be. For this they must put something into the public domain that will only reveal its value at E, or at least after agents are already committed, and where this revelation happens by itself, or in such a way that bad agents cannot prevent it.

For us, the obvious way to achieve this is to employ *delay encryption*, originally proposed in (Rivest et al., 1996). This creates an object $delay(t,X)$ such that anyone knowing it will eventually be able to know X, but not until at least t time units from its creation. There are essentially two types of delay encryption: *inexact* and *exact*.

The first of these, inexact delay encryption, depends on anyone trying to use it being required to do a sequential computation of such length that it will take them at least t. We term it inexact because a typical user will inevitably take longer than the minimum time we assume an opponent would take to complete it. Typically, one assumes an approximate ratio $K = t_{user}/t_{opponent}$ of 10–20. This mismatch between theoretical minimum and likely actual decryption times can impede progress when the result of the decryption is needed for subsequent processing. There are various schemes (Rivest et al., 1996; Cai et al., 1993) based on iterated squaring, which typically have fast verification schemes: given X and a value that purports to be $delay(t,X)$, this can be verified quickly. There are also schemes based on non-invertible operations such as (iterated) hashing. We could use such a scheme as follows: a group of participants each delay-encrypts their share of the oracle and ensures that this is visible in the blockchain before T/K, where T is the time from creation of the oracle to when it is meant to open. Only those participants whose delayed shares are registered by a block meeting this specification are used.

The creators of the shares may well still be asked to release their shares at E, but doing so would not save a lot of work because the work to decrypt the delayed shares will have already been done. If there are two ways in which an S_A can become public, then whichever way actually occurs will have to be checked against an unambiguous commitment of the sort discussed earlier.

Depending on whether or not the delay method is verifiable quickly or not, either a representative sample of agents decrypt these shares (and publish them for verification by all), or all do so that by T they all have the means to create the oracle. Alternatively, they could all know that one of the shares was not properly created so they can penalise the relevant party. The latter, we assume, is much preferable to penalising inaction.

It is clear that this method involves agents in a fair amount of sequential work for each share they want to decrypt. Equally, this method ensures that no agent can bias the oracle when implemented properly, but it is possible for well-equipped parties to know the oracle early in the interval between T/K and T: before we would ideally like.

We will term this the *MPC1* oracle because it involves only one round of multi-party computation, namely the shares of V. The delay calculations are individual. In its favour this method does not depend on the trustworthiness of agents other than that enough are chosen and that two distinct good agents contribute shares.

The second form of delay encryption we term exact. In it, time is measured explicitly by agents who release the material that reveals the encrypted X. The obvious implementations of this employ *a trusted third party* or TTP. Typically, it will have the capability of releasing X at any time, but will in fact only release it at E. It might actually escrow X itself, or it might know a key that allows people to decrypt X at any time, and release that key at E — the key would probably be the dual of a key released earlier as identified with E.

Assuming the existence of a TTP is generally thought not to be in the spirit of blockchain. Thus, it would be good if we could implement exact delay without one, instead taking advantage of the trust model that underpins the blockchain itself. This is possible provided that we are confident we can pick a set of M agents — not necessarily distinct — of whom less than half are bad, and the good will perform the role laid out below including keeping the secrets we require. We define K, the threshold, to be $[(M+1)/2]$, the smallest integer at least $M/2$. Note that we are assuming that K is larger than the number of bad agents amongst them, and that also $M-K$ is greater than or equal to the number of bad agents.

We direct the M agents each to publish a "public" key $pk(j,E)$ to be associated with the event E, by placing it in the blockchain, and to keep the corresponding $sk(j,E)$ secret until E, and then reveal it. We note that, by the assumptions above, less than K of these keys are revealed before E and at least K of them are revealed at E.

To create $delay(E,X)$, namely a value that will reveal X at E, anyone can therefore form the (M,K) threshold encryption of X, namely a tuple of M values such that knowing $K-1$ of them reveals nothing about X, but knowing K allows X to be extracted precisely. The delay encryption then consists of these M values, with the j^{th} encrypted by $pk(j,E)$. We know that X will, thus, be revealed to anyone holding this delay when E occurs, and that it is not revealed to anyone before this.

This, of course, explains the point of Table 1: it shows how large M has to be under different assumptions.

With this oracle, we would still expect agents to release their shares S_A openly at E. This time, work to anticipate non-release is not required as it was with the version using sequential computation. And we should require a hash commitment in the chain to guarantee that the value opened is correct whichever way it opens. As before, the shares that contribute to the oracle are exactly those that have this hash commitment and the delay encryption in the chain by some defined point well before E: certainly we would expect this set to be immutable by E to deny a possible choice mechanism.

We will term this the *MPC2* oracle, because it relies on two phases on multi-party computation: one for delay and one for the value V itself.

It makes sense to compare the MPC1 and MPC2 oracles. MPC1 needs fewer trust assumptions than MPC2, which may be a big advantage when we would otherwise inhabit the more expensive regions of Table 1. Of course, not needing to spread delay encryption amongst many parties itself apparently saves work and clearly does save a large number of public key encryptions. On the other hand, the decryptions in MPC1 have to begin immediately after the delayed values are posted, and cannot be avoided if the delayed values are posted at E as they can with MPC2. These decryptions are intense though sequential calculations: hopefully each agent will have more cores than shares to decrypt. MPC2 essentially guarantees that no enemy will have the value of the oracle before E, but this is an explicit possibility with MPC1, albeit not so early as to allow bias. Thus, we have two related ways of achieving the desired oracle, each with its own pros and cons.

6 RELATED WORK

Methods for generating and sharing random numbers have been the subject of much research and development; sometimes in their own right and sometimes as building blocks of a more complex protocol (Random.org, 1998; Kelsey et al., 2019; Cachin et al., 2005; Canetti and Rabin, 1993; Feldman and Micali, 1989; Gennaro et al., 1999; Micali and Sidney, 1995; Naor et al., 1999; Bhat et al., 2021; Das et al., 2022). In this section, we focus on distributed methods that are used to construct a common source of randomness, a random beacon (Rabin, 1983b). For a more detailed comparison, a systematic review of such work is presented in (Raikwar and Gligoroski, 2022).

Some distributed methods rely on a trusted setup (Cachin et al., 2005; Micali and Sidney, 1995; Naor et al., 1999). They depend on a *trusted dealer* to initialise the parties involved in the protocol. Once initialised, these interact to generate several random numbers in such a way that no party (dealer excluded) can predict in advance the numbers being jointly computed. The need for a trusted setup makes these approaches unsuitable in the context of blockchains, given their trust assumptions. It is worth mentioning that some of these papers propose alternative methods in which the trusted setup is avoided.

As the need for a trusted setup step is clearly undesirable in many domains, work has been carried out on alternative methods that avoid such a step (Micali and Sidney, 1995; Canetti and Rabin, 1993; Feldman and Micali, 1989; Feldman and Micali, 1997), some of which we discuss next.

Unsurprisingly, many of the works that have proposed distributed methods to generate random numbers (Canetti and Rabin, 1993; Feldman and Micali, 1997; Micali and Sidney, 1995; Kiayias et al., 2017; Gilad et al., 2017) involve solving some version of the Byzantine Agreement problem (Lamport et al., 1982; Pease et al., 1980) using randomised protocols (Rabin, 1983a). More recently, with the advent of blockchains, research into random beacons in their own right has flourished (Das et al., 2022; Syta et al., 2017; Bhat et al., 2021; Cascudo and David, 2017; Nguyen-Van et al., 2019; RANDAO, 2016; Drand, 2017; Ephraim et al., 2020; Schindler et al., 2020; Schindler et al., 2021).

Some methods rely on verifiable secret sharing techniques (VSS) (Chor et al., 1985; Feldman, 1987) in which participants share, verify, and combine locally created random shares (Syta et al., 2017; Cascudo and David, 2017; Das et al., 2022; Bhat et al., 2021; Schindler et al., 2020; Kiayias et al., 2017) to generate a random number. They can tolerate a number of misbehaving participants: these cannot conspire to influence or predict the output of the beacon before its revelation stage.

RANDAO (RANDAO, 2016; Edgington, 2022) is an interactive approach in which participants contribute locally-generated random shares, which are later combined. The first instance of such an approach was implemented as a smart contract using a commit-reveal protocol. Later, it was included as part of the Beacon Chain randomness generation process — see the Randomness chapter in (Edgington, 2022) — where each block produced contributes towards the output of the random beacon. These approaches allow the last contributor to the random number to have some influence over which number is output by the beacon.

Dfinity (Camenisch et al., 2021; Camenisch et al., 2022) is a blockchain protocol that uses a threshold signature scheme, based around BLS signatures (Boneh et al., 2001), to create a random beacon. The beacon is initialised with a well-known agreed-upon value, and all subsequent values are given by a *unique* signature on the previous value emitted which is jointly computed by the contributors to the beacon. The initialisation of the threshold scheme proposed, however, requires either a trusted setup or a secure key distribution protocol.

Some interactive protocols have used homomorphic encryption (amongst other cryptographic primitives) to create a random beacon (Nguyen-Van et al., 2019; Cherniaeva et al., 2019). In these protocols, random shares locally generated by the parties involved in the production of the beacons output are combined using homomorphic encryption. This combination is later decrypted to reveal the beacon's output.

Algorand (Gilad et al., 2017) is a blockchain protocol that relies on verifiable random functions (Micali et al., 1999) to implement a random beacon. This beacon serves as an input (i.e. seed) to randomly select participants to perform certain tasks, such as producing and validating blocks. Block producers compute the next value of the beacon by running a verifiable random function on the previously computed value of the beacon. If the producer fails to propose a beacon value, the hash of the previous beacon value concatenated with the current round number is used.

There are proposals for using blockchain data, as a source of randomness, to create a random beacon (Bonneau et al., 2015). The block producers in such a system have some influence over the data that is included in block and, hence, on this kind of beacon. Some extensions of this initial idea have used delay functions (Bünz et al., 2017; Bonneau et al., 2015). The use of a delay function to compute the beacon's output would mean that when the output of the beacon is revealed, the producers are no longer able to manipulate the data they have included in blocks involved in producing this output.

Some approaches have used verifiable delay functions (VDFs) (Boneh et al., 2018) to create a random beacon (Ephraim et al., 2020; Schindler et al., 2021). They rely on functions that, assuming they take t time to compute, computing their i-fold composition takes about $i \cdot t$, and parallelism does not improve on this time. Moreover, one can verify the correctness of an output of this function for a specific i, and can do so efficiently without having to recompute the i-fold composition. With such a function, a random beacon can be created by setting an initial random seed and having the i^{th} output of the beacon as the i^{th} composition of this function applied to the seed.

7 CONCLUSIONS

In this paper we have demonstrated that delay encryption can create unbiassable random oracles in ways that treat the threat of selective non-participation as a primary issue.

Delay encryption is a very useful tool for keeping diverse participants honest, when no one is sure who is trustworthy. Almost always, as here, the extra security comes from preventing untrustworthy parties from withdrawing in a way that is beneficial to them. Since reliable and secure random oracles are so important to blockchains, we believe that the two related approaches we have given to building these oracles are worthwhile.

Blockchains frequently have to make fair decisions that are demonstrably independent of their participants. A random oracle of the sort we have created is key to this, as well as to the continued integrity of the chain.

Our two options, one employing significant sequential computation and the other exploiting the blockchain itself as a synthetic TTP, each have their advantages and drawbacks. We imagine that which to use will depend on circumstances.

ACKNOWLEDGEMENTS

We would like to thank Ivo Maffei for his work on, and discussions about, delay encryption as implemented by sequential computation. He has greatly improved our understanding of this. We also thank Phil White and Garry Lancaster for their experimental implementations of these and other blockchain ideas.

REFERENCES

Bhat, A., Shrestha, N., Luo, Z., Kate, A., and Nayak, K. (2021). Randpiper: Reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, CCS '21, page 3502–3524, New York, NY, USA. Association for Computing Machinery.

Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. (2018). Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer.

Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the weil pairing. In Boyd, C., editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 514–532, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bonneau, J., Clark, J., and Goldfeder, S. (2015). On bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*, page 1015.

Bünz, B., Goldfeder, S., and Bonneau, J. (2017). Proofs-of-delay and randomness beacons in ethereum. https://jbonneau.com/doc/BGB17-IEEESB-proof_of_delay_ethereum.pdf.

Cachin, C., Kursawe, K., and Shoup, V. (2005). Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246.

Cai, J.-y., Lipton, R., Sedgewick, R., and Yao, A.-C. (1993). Towards uncheatable benchmarks. In *[1993] Proceedings of the Eigth Annual Structure in Complexity Theory Conference*, pages 2–11.

Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.-A., Shoup, V., and Williams, D. (2021). Internet computer consensus. Cryptology ePrint Archive, Paper 2021/632. https://eprint.iacr.org/2021/632.

Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.-A., Shoup, V., and Williams, D. (2022). Internet computer consensus. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, PODC'22, page 81–91, New York, NY, USA. Association for Computing Machinery.

Canetti, R. and Rabin, T. (1993). Fast asynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, STOC '93, page 42–51, New York, NY, USA. Association for Computing Machinery.

Cascudo, I. and David, B. (2017). Scrape: Scalable randomness attested by public entities. In Gollmann, D., Miyaji, A., and Kikuchi, H., editors, *Applied Cryptography and Network Security*, pages 537–556, Cham. Springer International Publishing.

Cherniaeva, A., Shirobokov, I., and Shlomovits, O. (2019). Homomorphic encryption random beacon. Cryptology ePrint Archive, Paper 2019/1320. https://eprint.iacr.org/2019/1320.

Chor, B., Goldwasser, S., Micali, S., and Awerbuch, B. (1985). Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395.

Das, S., Krishnan, V., Isaac, I. M., and Ren, L. (2022). Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2502–2517.

Drand (2017). Drand: Distributed randomness beacon. https://drand.love/. Accessed: 2022-11-23.

Edgington, B. (2022). Upgrading ethereum: A technical handbook on ethereum's move to proof of stake and beyond. https://eth2book.info/bellatrix. Accessed: 2022-11-23.

Ephraim, N., Freitag, C., Komargodski, I., and Pass, R. (2020). Continuous verifiable delay functions. In Canteaut, A. and Ishai, Y., editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 125–154, Cham. Springer International Publishing.

Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438.

Feldman, P. and Micali, S. (1989). An optimal probabilistic algorithm for synchronous byzantine agreement. In Ausiello, G., Dezani-Ciancaglini, M., and Della Rocca, S. R., editors, *Automata, Languages and Programming*, pages 341–378, Berlin, Heidelberg. Springer Berlin Heidelberg.

Feldman, P. and Micali, S. (1997). An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933.

Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1999). Secure distributed key generation for discrete-log based cryptosystems. In *Proceedings of the 17th International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'99, page 295–310, Berlin, Heidelberg. Springer-Verlag.

Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68, New York, NY, USA. Association for Computing Machinery.

Kelsey, J., T. A. N. Brandão, L., Peralta, R., and Booth, H. (2019). A reference for randomness beacons: Format and protocol version 2. https://nvlpubs.nist.gov/nistpubs/ir/2019/NIST.IR.8213-draft.pdf. Accessed: 2022-11-23.

Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In Katz, J. and Shacham, H., editors, *Advances in Cryptology – CRYPTO 2017*, pages 357–388, Cham. Springer International Publishing.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Micali, S., Rabin, M., and Vadhan, S. (1999). Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE.

Micali, S. and Sidney, R. (1995). A simple method for generating and sharing pseudo-random functions, with applications to clipper-like key escrow systems. In Coppersmith, D., editor, *Advances in Cryptology — CRYPT0' 95*, pages 185–196, Berlin, Heidelberg. Springer Berlin Heidelberg.

Naor, M., Pinkas, B., and Reingold, O. (1999). Distributed pseudo-random functions and kdcs. In Stern, J., editor, *Advances in Cryptology — EUROCRYPT '99*, pages 327–346, Berlin, Heidelberg. Springer Berlin Heidelberg.

Nguyen-Van, T., Nguyen-Anh, T., Le, T.-D., Nguyen-Ho, M.-P., Nguyen-Van, T., Le, N.-Q., and Nguyen-An, K. (2019). Scalable distributed random number generation based on homomorphic encryption. In *2019 IEEE International Conference on Blockchain (Blockchain)*, pages 572–579.

Pease, M., Shostak, R., and Lamport, L. (1980). Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234.

Rabin, M. O. (1983a). Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409.

Rabin, M. O. (1983b). Transaction protection by beacons. *Journal of Computer and System Sciences*, 27(2):256–267.

Raikwar, M. and Gligoroski, D. (2022). Sok: Decentralized randomness beacon protocols. https://arxiv.org/abs/2205.13333.

RANDAO (2016). RANDAO: A DAO working as RNG of Ethereum. https://github.com/randao/randao. Accessed: 2022-11-23.

Random.org (1998). Random.org: True random number service. https://www.random.org/. Accessed: 2022-11-23.

Rivest, R., Shamir, A., and Wagner, D. (1996). Time lock puzzles and timed release cryptography. Technical report, Technical report, MIT/LCS/TR-684.

Roscoe, A. and Chen, B. (2019). Delay and escrow in the blockchain. https://tbtl.com/wp-content/uploads/2020/11/delayblock.pdf. Accessed: 2022-11-23.

Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., and Weippl, E. R. (2021). 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.

Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. (2020). Hydrand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 73–89.

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

Wikipedia contributors (2022). Normal approximation to a binomial distribution. https://en.wikipedia.org/wiki/Binomial_distribution#Normal_approximation.

Wolfram Alpha LLC (2022). Wolfram alpha. https://www.wolframalpha.com/.

APPENDIX

The calculations in Section 3 are based on approximating the cumulative distribution function of the binomial with N trials and probability p, by the normal distribution with mean $\mu = Np$ and variance $\sigma^2 = Np(1-p)$. The main reference is the Wikipedia article (Wikipedia contributors, 2022).

We first calculated an approximation for probabilities of the form 10^{-k} of the multipliers w_k such that $Pr[X > w_k\sigma] = 10^{-k}$. Some of these values are quoted in the article; others were derived by numerically solving the following approximation derived from the two-sided one there.

$$k\,ln(10) = w_k^2/2 + ln(w_k) + ln(\sqrt{2\pi})$$

Here one-sided means we are interested in the random variable being exceptionally large only; two sided means we add the probability of it being far away from the mean on either side.

We then calculated the thresholds on N for each p and k such that $\mu + w_k\sigma = N/2$ and rounded up.

In Table 1, we also give the multiple M_σ of the standard deviation, such that the probability that the normal distribution exceeds $M_\sigma\sigma$ above the mean, equivalent to each given value of ε.

The calculations were made using Wolfram Alpha (Wolfram Alpha LLC, 2022).