Proofs-of-delay and randomness beacons in Ethereum

Benedikt Bünz[†], Steven Goldfeder*,Joseph Bonneau[†] *Princeton University, [†]Stanford University

Abstract—**Blockchains generated using a proof-of-work consensus protocol, such as Bitcoin or Ethereum, are promising sources of public randomness. However, the randomness is subject to manipulation by the miners generating the blockchain. A general defense is to apply a delay function, preventing malicious miners from computing the random output until it is too late to manipulate it. Ideally, this delay function can provide a short proof-of-delay that is efficient enough to be verified within a smart contract, enabling the randomness source to be directly by smart contracts. In this paper we describe the challenges of solving this problem given the extremely limited computational capacity available in Ethereum, the most popular general-purpose smart contract framework to date. We introduce a novel multi-round protocol for verifying delay functions using a refereed delegation model. We provide a prototype Ethereum implementation to analyze performance and find that it is feasible in terms of gas costs, costing roughly 180000 gas (US$ 0.11[1]) to post a beacon and 720000 gas (US$ 0.98) to resolve a dispute . We also discuss the incentive challenges raised by providing a secure randomness beacon as a public good.**

I. Introduction

Many security protocols require public randomness which cannot be manipulated or predicted by any party in the protocol. The simplest example is a public lottery in which a random winner most be chosen from a set of candidates. There are many more examples in the cryptographic literature, including unpredictable challenges for non-interactive zero-knowledge proofs [15], random audits of verifiable elections [2] or generating random parameters for cryptographic standards to preclude the presence of backdoors [5]. Rabin first coined the term *randomness beacon* [21] to describe an ideal service which publishes unpredictable random values at regular intervals that no party can manipulate. Rabin introduced the concept as a tool for fair exchange protocols, another important application.

[1]Throughout the paper we assume a gas cost of $2 \cdot 10^{-8}$ ETH per gas and an ETH price of 68 $/ETH

Because no ideal beacon exists, traditional approaches include emulating an ideal beacon using a trusted third party (such as the NIST beacon [19]) or generating randomness from the parties in a protocol using a commit-and-reveal approach [10] in which each party commits to a nonce before all commitments are opened and the nonces combined to form a random output. Basic commit-and-reveal protocols produce unpredictable randomness as long as at least one participant is honest. However, preventing malicious parties from selectively aborting the protocol (and hence manipulating the result) requires either provably verifiable secret-sharing techniques [16, 22] or financial penalties enforced by a programmable cryptocurrency to punish parties that abort [3]. Commit-and-reveal approaches are used in practice, typically in specific protocols where a semi-trusted set of authorities exists (e.g. the Tor protocol [13] uses a commit-and-reveal approach to establish a random seed to build routing paths through the network). A commit-and-reveal protocol called RANDAO [1] has been implemented using Ethereum in which anybody can commit to a random seed and parties which do not reveal forfeit a bond.

An alternative approach is to extract random values from a "naturally occurring" source of entropy such as stock-market data [12], national lottery results [5] or proof-of-work blockchains [6, 8, 20] such as Bitcoin [18] or Ethereum [23]. However, while costly to manipulate, each of these sources is vulnerable to malicious insiders (e.g. high-frequency traders, lottery administrators, or cryptocurrency miners).

A. Manipulating blockchain-based beacons

Our primary focus is on extracting randomness from proof-of-work blockchains. Consider the basic approach of simply agreeing to take a specific future block and running it through an extractor function to provide uniform randomness. Indeed, many Ethereum-based contracts, in particular gambling contracts, already apply this approach to generate randomness [4]. It has also been used in several Bitcoin-based protocols [9, 15]. In the absence of manipulation by miners, this approach provides good randomness. It can be shown that the min-entropy of each block must be, at minimum, equal to the difficulty of the current proof-of-work puzzle [8], or else a shortcut to computing the puzzle would exist.

However, this approach is subject to manipulation by miners [6, 8, 20] who can choose to withhold valid blocks they find if they produce a beacon output they do not want. This attack might be performed by a large mining pool [6, 20] or even by a non-miner who is able to bribe miners not to publish blocks [8]. These attacks are costly as blocks are valuable and thus discarding them represents a large opportunity cost. For example, fixing a single-bit beacon output would cost (on expectation) an entire block reward, equal to about US$16,250 for Bitcoin or US$340 for Ethereum at today's prices (note that Ethereum blocks are found $\sim$40 times more frequently).

This cost can be amplified somewhat by combining multiple blocks using a low-influence function to mitigate the effects of the last block [6]. This provides a limited defense, with n blocks increasing security by a factor of $\Theta(\sqrt{n})$.

Low-cost attack opportunities might arise when colliding blocks are naturally found. Due to network latency, it is common for two miners to discover separate valid blocks at nearly the same time. Other miners may choose arbitrarily which block to attempt to extend, and can attempt to influence the beacon result at no cost when doing so. An attacker might also attempt to manipulate the network itself to prevent or delay propagation of a block producing an undesirable beacon output. Unlike withholding attacks, these attacks are difficult to provide precise cost estimates for and thus are a major impediment to relying on blockchain-based beacons.

Network attacks are a particular concern for an Ethereum-based blockchain, as Ethereum's low inter-block arrival time means colliding blocks are found frequently and hence attack opportunities are commonplace. For example, colliding blocks are found several times a day for Bitcoin (about one in one hundred blocks), but about 400 times per day in Ethereum[2]) as the inter-block time is much lower.

B. Preventing manipulation using delay functions

To prevent both network-manipulation and block-withholding attacks, we can apply a *delay function* to the block before extracting randomness [8]. Intuitively, the idea is that miners (or any other party) cannot determine the beacon result from a given block before some non-negligible amount of time has elapsed, at which point it is too late to attack as the blockchain has already moved on.[3]

A delay function, also called a slow-time function [17], is an inherently-sequential function that is believed to be difficult to compute in less than some adjustable amount of wall-clock time. Ideally, this function is a permutation so that it maintains the full-entropy of the input (although a pseudorandom function is a reasonable substitute at the relatively low iteration counts likely to be used in practice).

A trivial example of a delay function would be simply iterating a pseudorandom permutation (such as a block cipher with a random key) or a collision-resistant hash function. Assuming the underlying function is secure, this produces an effective delay function. Any user of the beacon would have to recompute this delay function to derive the beacon result. This may be acceptable in some non-latency sensitive applications such as generating cryptographic parameters [5] or election auditing [2], but it is likely not acceptable in

[2]This only counts blocks which are included as uncle blocks in the Ethereum chain. The real rate of stale block discovery is strictly higher.

[3]Of course, it would always be possible to attack retroactively by attempting to induce a deep fork of the blockchain, but we consider this attack out-of-scope as it violates a fundamental security assumption of the blockchain.

latency-sensitive applications such as a gambling contract in Ethereum.

A better delay function might produce a proof that the output is correct which is faster to verify than recomputation. We call such a proof a *proof-of-delay*. A classic example is computing modular square roots, first suggested by Dwork and Naor [14]. For $p \equiv 3 \pmod 4$, computing $y \equiv \sqrt{x} \equiv x^{\frac{p+1}{4}} \pmod p$ requires $\Theta(\lg p)$ modular squarings, whereas verifying that $x \equiv y^2 \pmod p$ requires just one squaring. Lenstra and Wesolowski [17] proposed the Sloth function, chaining together multiple modular square roots to induce an arbitrary amount of delay, with verification still faster by a factor of $\Theta(\lg p)$. Currently, this is the most efficient proof-of-delay known.

C. *Our contributions*

We show that a proof-of-delay built using modular square roots is not efficient enough for use in Ethereum. Given the extremely limited (and costly) computation environment provided by Ethereum, verifying a sufficiently long proof-of-delay (e.g., for a $\sim$10 minute delay) is cost-prohibitive and impossible due to network constants. As we show in Section V even the cheapest computation that can be done in a second on a standard notebook takes on the order of 22 million gas which is equivalent to USD$29.92. This is not only very expensive but also already above the global per block gas limit which is currently at 4 million gas[4].

Instead, we propose using a *refereed delegation of computation* protocol to build a public randomness beacon (Section II). The basic idea is that a designated authority (a beacon maintainer) computes a delay function and publishes the result to a smart contract providing queryable access to the beacon results. Trust in this authority is mitigated by the use of a challenge protocol that any third party can initiate to prove that the authority has published an incorrect beacon result.

We implemented our protocol (Section V) and evaluate its efficiency. It is very cheap in the absence of any challenge (80000 gas/US$ 0.11) and provides acceptable performance in the event of a challenge (720000 gas/US$ 0.98 to mount a successful challenge and up to 280000 gas/US$ 0.38 to defend against an invalid one) using the conservative parameters from Section V.

Our second contribution is to analyze the incentives of running a public randomness beacon (Section IV). A randomness beacon is inherently a public good and hence it is not suitable to be administered as a for-profit business. We discuss potential economic models and the difficulty of incentivizing faithful behavior by the beacon authority as well as vigilant checking by third parties.

[4]https://ethstats.net/

II. Protocol

We start with any *deterministic* function f as our building block. The only essential requirements of is that it is *compositionally-sequential*: that is, there should be no algorithm for computing its composition:

$$f^n(x) = (f \circ f \ldots f)(x) \qquad (n \text{ iterations}) \quad (1)$$

which is faster than n times the cost of computing f. This property should hold for most cryptographic hash functions or pseudorandom permutations. Note that the compositionally-sequential property implies that there are no loops in f's functional graph (or at least, they are not feasible to find), as loops in the function would enable long compositions to be efficiently computed. Permutations are guaranteed to have no loops, whereas for secure hash functions they are believed to be computationally intractible to find.

Equivalently, we can choose an f with an asymmetric verification cost, such as modular square roots [14] or chained modular square roots [17]. In this case, it is possible to compute f^{-1} much faster than f and therefore F^{-1} can also be computed.

Given such a function f, we build our delay function F for a desired delay time t as:

$$F(x) = f^t(x) \qquad (2)$$

We assume that a designated prover, who maintains a beacon service, will compute the value $y = F(X)$ and publish it to the beacon contract. The question becomes, how can we gain confidence that $y = F(X)$ as claimed?

A. Recomputation

The simplest solution is to directly verify that $y = F(x)$ (either by recomputation or computing $F^{-1}(y)$ if F has a cheaply computable inverse). However, this puts a low limit on t as computation by the beacon contract is expensive. This approach is not practical, although we will explore in Section V the costs of this approach.

B. Refereed delegation model

Due to the limits on t imposed by complete recompuation, we can use a *refereed delegation of compuation model*, as first formalized by Canetti et al. [11]. In this model the beacon service include a fidelity bond to the contract, which is forfeited if it can be proven that an incorrect $y' \neq F(X)$ has been published. Our goal is to make it efficient for independent verifiers (or *referees*) to check that the computation was performed correctly.

1) One-round refereeing protocol: A simple addition to the protocol is for the prover to publish a series of *checkpoints* representing intermediate states in the computation of F. For example, the prover might publish k checkpoints y_i for $i \in [1, k]$ as follows:

$$y_i = f^{\frac{i \cdot t}{k}}(x) \tag{3}$$

If we define $y_0 = x$, then we have the property that for all $i \in [1, k]$:

$$y_i = f^{\frac{t}{k}}(y_{i-1}) \tag{4}$$

The final result of F is simply the last checkpoint:

$$y = F(X) = f^t(x) = f^{\frac{k \cdot t}{k}}(x) = y_k \tag{5}$$

Now, any verifier can prove that the result is incorrect simply by specifying a single i for which Equation 4 does not hold. There are two advantages to this approach. First, recomputation can now be done in parallel (with up to k processors) rather than requiring a sequential computation. Second, the beacon contract can now check any claim of misbehavior with $m = \frac{t}{k}$ invocations of either f or f^{-1}, as appropriate.

This approach also has the advantage that the prover simply needs to upload its k checkpoints and then claims by verifiers can be adjudicated with no further action.

The downside is that this approach requires uploading k values to the contract, which is expensive, and therefore large k is prohibitive. We are left with the tradeoff that $m \cdot k = t$, where m is the number of invocations of f/f^{-1} that can be performed and k is the number of intermediate states which can be uploaded. We consider concrete parameters in Section V.

2) n-round refereeing protocol: The non-interactive model provides a limited tradeoff. To enhance it, observe that the prover is in fact claiming that a certain pair y_{i-1}, y_i is incorrect, that is, that $y_i \neq f^{\frac{t}{k}}(y_{i-1})$. The prover can instead make a slightly more specific claim by providing a y'_i such that $y'_i = f^{\frac{t}{k}}(y_{i-1})$. Now the prover's claim is similar to the verifier's original claim, only for a new iteration count $t' = \frac{t}{k}$.

We can therefore run a second round in which the original verifier is now the prover checking the original prover's claim. The original verifier should itself provide a series of k checkpoints, to which the original prover can claim any pair of which are not valid.

In general, we can build an n-round protocol using this intuition, with the prover and verifier taking turns publishing a series of checkpoints until the distance between checkpoints is small enough (a distance m apart) that it is economical for the contract to directly verify. This protocol is defined in Protocol 1.

Note that a successful challenge does not prove that $y \neq F(x)$ (nor does a failed challenge prove that $y = F(x)$). In either case, a malicous prover or challenger can "lose" the protocol by choosing a pair of checkpoints incorrectly at any given step. However, the protocol does provider the property that if $y = F(x)$, an honest (and online) prover can never be successfully challenged. Similarly, if $y \neq F(x)$, and honest and online challenger can always mount a successful challenge.

3) Efficiency: Efficiency can be meaured in terms of the number of rounds, the amount of data sent to the contract, and the amount of computation performed by the contract. Given the total amount of work t performed by the prover, the number of rounds n, the number of checkpoints k sent in each round, and the number of iterations m, the invariant must hold that:

$$t = m \cdot k^n$$

Protocol parameters:

- $f()$: core function
- t: desired number of invocations of f by the prover
- $F()$: aggregate function, equivalent to $f^t()$
- x: input value
- y: output value, $y = F(x)$
- m number of invocations of f or f^{-1} to be checked by the contract
- n: number of rounds in the protocol. Without loss of generality, we assume n is even.
- k: number of checkpoints per round
- y^i_j: the jth checkpoint of round i
- d_i: the distance between checkpoints in round i. That is, $y^i_j = f^{d_i}(y^{i-1}_j)$. Note that $d_i = \frac{t-m}{k^i}$.
- P: the prover for the overall protocol
- C: the challenger for the overall protocol
- P_i: the challenger for round i. If i is odd, $P_i = C$. Otherwise $P_i = C$.
- b: a flag set by the contract as a result of its direct verification after the final round of the protocol.

The protocol then proceeds between the prover P and a challenger C as follows, after P computes $y = F(x)$. The first round requires both parties to post bonds and is thus slightly different:

1) P sends a fidelity bond of value b_P to the contract.
2) P publishes the final value y and the initial checkpoints $y^0_0, \ldots y^0_k$ with $y^0_0 = x$ and $y^0_k = y$.
3) Any challenger may post a fidelity bond of value b_C to begin the protocol.

The protocol then proceeds through n rounds, with the prover and challenger alternating roles in each round. Each round i proceed as follows:

1) If i is odd, then the prover P_i for this round is C. Otherwise, P_i is P.
2) P_i claims that a checkpoint y^{i-1}_j for $j \in [1, k]$ is incorrect.
3) If $i < n$, P_i publishes a series of checkpoints $y^i_0 \ldots y^i_k$ with $y^1_0 = y^{i-1}_{j-1}$ and $yi_k \neq y^{i-1}_j$.

If the prover P_i fails to provide a new set of checkpoints in any round i within a given time period, the protocol aborts and P_{i-1} is considered the winner.
If all n rounds complete, then the contract computes $f^m(y^{n-1}_j)$ for the checkpoint y^{n-1}_j claimed to be incorrect in the final round. The contract then sets a flag b to true iff $f^m(y^{n-1}_j) = y^{n-1}_{j+1}$.
Alternately, if f^{-1} is easier to compute then the contract can compute $f^{-m}(y^{n-1}_{j+1})$ and set b to true iff $f^{-m}(y^{n-1}_{j+1}) = y^{n-1}_j$.
Assuming n is even, then if b is true the challenger C is the winner and the challenge is upheld. If b is false, the final prover $P_n = P$ is the winner and the challenge is rejected. Note that the protocol simply declares a winner, and proves nothing about if $y = F(x)$. We discuss possible implications of the protocol in Section IV.

Protocol 1: n-round refereed computation prototocl

Assuming neither party aborts, the n-round protocol really consists of $n+1$ rounds, including a 0th round in which the prover publishes the claimed value $y \overset{?}{=} F(x)$. The total amount of data sent to the contract is $k \cdot n$ checkpoint values (elements from the range of f) and n indices of challenged checkpoints (each an integer in the range $[1, k]$.

III. Security and Correctness

To show the security of Protocol 1, we model the interaction as a game between a prover P and a challenger C. We show that an honest prover will always have a winning strategy against any challenger, whereas for every dishonest prover, there will always exist a challenger with a winning strategy. Moreover, we show that the respective winning strategy is simply following the steps of Protocol 1 honestly.

This is still not enough, however, for the beacon to be secure. The outcome of the game does not prove whether or not the prover is honest and posted the correct beacon output. For example, an honest prover may incorrectly answer a challenge (or abort) causing it to lose the game. Alternatively, a dishonest prover may receive a challenge for which it can win (i.e. the challenger did not challenge at the correct checkpoint). The security of the protocol instead rests on the existence of winning strategies as well as an incentive structure that ensures that rational actors will follow their winning strategies.

In this section, we prove the existence of and identify the winning strategies. In Section IV we show how to incentivize the players to follow these strategies.

First, we prove a simple lemma that will aid us in our proofs.

Lemma 1 (Transition Point). *If $y_k^i \neq f^{k \cdot d_i}(y_0^i)$, there exists some $j \in [1,k]$, such that checkpoint y_j^i is incorrect, but checkpoint y_{j-1}^i is correct. We call y_j^i the* **transition point**.

Proof. The first checkpoint, y_0^i is supplied by the contract and will always be correct. In particular, $y_0^0 = x$ and for $i \in [1,n]$, $y_0^i = y_{j-1}^{i-1}$, where y_j^{i-1} is the checkpoint that was challenged from the previous round. Thus, if we iterate through checkpoints $y_1^i, \cdots, y_k^i$ in order, the first incorrect one that we encounter will be a transition point. Moreover, we are guaranteed to encounter an incorrect checkpoint since $y_k^i \neq f^{k \cdot d_i}(y_0^i)$, which means the the last checkpoint, y_k^i, is incorrect. □

We now use this lemma to show that an honest prover always has a winning strategy, and that there is always a winning strategy for a challenger against a dishonest prover.

We prove these as separate theorems. Our protocol has a recursive structure, and the prover in one round becomes the challenger in the next. This is reflected in out proof as our theorems reference one another. The argument is not circular as every time one theorem references the other, the round is incremented, and each theorem proves its own base case for the final round.

Theorem 1. *Let P be a computationally unbounded prover and C be a computationally bounded challenger. If in any round $i \in [1,n]$, P posts a checkpoint $\hat{y}_k^i \neq f^{k \cdot d_i}(y_0^i)$, there exists a winning strategy for C.*

Proof. To prove this theorem, we explicitly construct the winning strategy.

If, $i = n$, C will be declared the winner by the contract. In particular, the contract will itself compute $y_k^i = f^{k \cdot d_i}(y_0^i)$. Since $\hat{y}_k^i \neq y_k^i$, C will be declared the winner.

If $i < n$:

1) Compute $y_k^i = f^{k \cdot d_i}(y_0^i)$ and compare each checkpoint with the one posted by the prover.
2) Find the transition point y_j^i (i.e. the smallest j such that the checkpoint posted by the prover does not equal y_j^i) and post this as the challenge. Since $\hat{y}_k^i \neq f^{k \cdot d_i}(y_0^i)$, by Lemma 1, such a point is guaranteed to exist.
3) In round $i+1$ post the correct checkpoints between $y_0^{i+1} = y_{j-1}^i$ and $y_k^{i+1} = y_j^i$.
4) At this point, the challenger has assumed the role of the honest prover and thus has a winning strategy by Theorem 2.

Thus, in any round there is a winning strategy for C against a dishonest prover.

□

Theorem 2. *Let P be a computationally bounded prover and C be a computationally unbounded challenger. If in any round $i \in [1,n]$, P is an honest prover, then there exists a winning strategy for P against any C.*

Proof. Since P is an honest prover, all of the checkpoints that it posted were correct. Thus, if $i = n$, the contract will itself compute $y_k^i = f^{k \cdot d_i}(y_0^i)$ and declare P to be the winner.

If $i < n$, then for $j \in [1,k]$, for every checkpoint value, y_j^i, that the prover posted it will be the case that $y_j^i = f^{k \cdot d_i}(y_{j-1}^i)$. To challenge, C will have to choose a j and claim that y_j^i is incorrect. In particular, it will have to post a value $\hat{y}_j^i \neq y_j^i$ and claim that this is the correct value.

But since f is deterministic, we have that $\hat{y}_j^i \neq f^{k \cdot d_i}(y_{j-1}^i)$. Thus, C assumes the role of a dishonest prover in the next round, and by Theorem 1, there exists a winning strategy for P. □

IV. Incentives

In this section we will discuss the incentive structure required for running a public beacon. In particular, four related behaviors need to be incentivized:

1) There must be an incentive for some party to compute the delay function—that is to post *some* beacon output value onto the blockchain.
2) There must be a stronger disincentive to posting an *incorrect* output value.
3) Independent parties must have an incentive to check the posted value and initiate a challenge if it is incorrect.

4) Independent parties must have a disincentive to challenging a correct value.

These incentives can be intertwined. We define the following values to reason about the incentive structure:

1) c_p is the cost of computing the beacon output
2) b_p is a fidelity bond posted by the prover
3) c_c is the cost of verifying the beacon output
4) b_c is a fidelity bond required to initiate a challenge
5) r_p is the reward collected by the prover for posting a beacon value that was not successfully challenged within a set time period
6) r_i is the reward collected by the prover for posting an incorrect value that goes unchallenged
7) p_i is the probability that a value posted is incorrect
8) p_c be the probability that someone will challenge an incorrect value

A. Beacons as a public good

A randomness beacon meets the two economic criteria of a *public good*. The utility of a beacon is non-excludable (anybody can use the beacon's output for free, as it is visible on the blockchain). Its utility is also non-rivalrous (utility is not diminished as more entities use the service). As a public good, we should not expect that a beacon will be provided by a private firm. Instead, it requires a community effort to fund.

This might take the form of traditional nation-state governments. For example, the US National Institute of Standards and Technology (NIST) currently operates a (non-verifiable) beacon as a public service [19]. They could potentially fund a verifiable blockchain-based beacon instead.

Short of nation-state support, the Ethereum Foundation might be a natural institution to fund a randomness beacon. The foundation's mission is to *promote and support research, development and education to bring decentralized protocols and tools to the world....*

Alternately, a beacon might be supported by crowdfunding or by an industry consortium of gambling services seeking to increase public confidence in their business.

1) Incentivizing computation: Assuming public funding to support the beacon, some party must actually compute it in return for payment. This might simply be a contracted service provider who is given sole charter to publish beacon outputs. An alternate approach is an open-competition model in which the first party that posts an output receives a reward. Two caveats are required here: first, a commitment to the output (along with a bond) should be published first to establish primacy before revealing the output value, to prevent a network attacker from observing an output value about to be posted and posting it themselves. Second, the party publishing the output must post a fidelity bond greater than the reward, which is lost if they fail to open their commitment or are successfully challenged.

We assume that even with open competition, the equilibrium state will be a single entity computing the result. Unlike cryptocurrency mining, in which even small miners are expected to find the winning block with probability equal to their overall share of the hash power, when computing a delay function the party with the fastest hardware will always win. Thus a single dominant beacon operator is likely to emerge.

This makes the open-competition model similar to the contracted provider model in practice. However, open competition has a significant benefit in that it incentivizes hardware upgrades. A designated service provider with a long-term contract will have little incentive to maintain the fastest hardware to compute the delay function. In the open-competition model, on the other-hand, someone with faster hardware will usurp the beacon operator if they fail to upgrade their hardware, and we can be sure that the beacon is being computed at a speed on-par with the most efficient hardware.

In any event, in order to run the beacon there must be a profitable reward for computing the correct value. That is, $r_p > c_p$.

B. Incentivizing Correctness

Assuming that incorrect result will always be challenged successfully if posted, we can view the fidelity bond posted as a means to prevent spam and to fund successful challenges. We would like to be able to ensure that there is never an incentive to post an incorrect result, but this is not the case if there is a non-negligible probability that an incorrect result will go unchallenged. The reason is that the potential gains r_i from posting an incorrect-but-unchallenged result are external to the protocol and unbounded. For example, an incorrect result can be used to ensure that the party posting the value wins a lottery with some arbitrarily prize. Thus, $(r_i + r_p) \cdot (1 - p_c)$ can always be greater than $p_c \cdot b_p$.

One potential fix would be to try to bound r_i, that is, to try to limit the amount of money any party can make by fixing the beacon output. This could take the form of limiting the prize value in any lottery depending on the beacon. This is very difficult to achieve in practice. First, it is impossible to police what third-party services rely on the beacon. Second, a cheating prover might pseudonymously have stake in multiple different third-party services, so the total incentive r_i is difficult to infer.

A simpler approach is to assume that we can achieve $p_c = 1$, that is, to ensure that an incorrect value will always be caught.

1) Reputation: While we have hitherto assumed b_p was a bond paid by the prover into the beacon contract, we can generalize b_p to include anything the prover stands to lose from being caught cheating.

Reputation is one such non-monetary asset that a beacon operator risks losing by posting an incorrect beacon value. If the beacon operator is a reputable organization (e.g. NIST), then b_p will contain not only the monetary bounty but also the "reputation cost" of cheating. By posting an incorrect value, the beacon operator risks losing both its monetary bond and its credibility.

Interestingly, the organization's reputation not only inflates b_p, but it also will likely inflate p_c – that is, people may be eager to expose a reputable organization and will be likely to check the beacon value even if they have no monetary stakes on the output.

C. Incentivizing verification

We can incentivize parties to verify (and challenge if appropriate) by offering an on-chain challenge reward, r_c, to successful challengers, paid out of the prover's fidelity bond. We should obviously require that $r_c > c_c$ (the reward for challenging is greater than the cost of verification) or else it would not be rewarding to challenge.

However, note that paying the entire bond to the challenger is a mistake. This would enable a cheating prover to notice if a challenge is being initiated on the network and frontrun by quickly challenging itself instead. This correct challenge would enable the prover to retain its fidelity bond. Thus, a significant portion of the fidelity bond should be burned,

to ensure that the prover will lose money for posting an incorrect value even if it recoups the challenger's reward r_c.

While we must ensure that $r_c > c_c$, this is not enough as we assume most posted values will be correct and there is no reward possible if a correct value is posted. Thus, we must ensure that $r_c \cdot p_i > c_c$, taking into account the (hopefully small) probability that the posted value is incorrect.

This is still not enough to ensure that verification will actually occur for two reasons. First, given that only the first correct challenger can be paid (otherwise the same challenger might be pseudonymously paid multiple times), all potential challengers might assume they are unlikely to win the race to challenge and hence not bother to verify, similar to the way that an open competition to compute the beacon value is likely to reach an equilibrium of only one party computing it. A malicious prover might actually try to amplify this effect by occasionally posting an incorrect value and quickly challenging, making it appear that a very fast challenger exists that other cannot compete with.

A second problem is that all potential challengers might rationally conclude that given an ecosystem with challengers, p_i is likely to be zero and hence there is no incentive to verify. This is an example of a *bystander effect*.

To combat these problems, we again may have to rely on public funding to subsidize a market for challengers. This can be done by paying for the challenge rewards (and lost bonds) from intentionally posting some incorrect results. By regularly posting incorrect values (at random times) and seeing that they are challenged, the beacon funders can be assured that a healthy ecosystem of verifiers actually exists. Knowing that these incorrect values are being posted should also overcome the bystander problem as verifiers know that it is worthwhile to verify posted results.

D. *Disincentivizing incorrect challenges*

Finally, we must discourage invalid challenges. To do this, we require that the challenger puts up a fidelity bond b_c, that it will lose in the event that its challenge is found to be incorrect. As we assume that the prover will always complete the protocol and disprove false challenges, there is no real incentive for a challenger to do this other than to deny or delay service to the beacon, and thus a small fee should discourage such an attack.

V. Costs and Parameters

In this section we discuss realistic parameters and evaluate costs for our Ethereum implementation of Protocol 1. Our implementation is available online.[5]

Our main objective in selecting parameters is to minimize the overall cost of maintaining the beacon service and of challenging an incorrect result. As discussed in Section IV, public funding is likely needed both to run the beacon service and to ensure challenges are regularly executed. The lower the transaction costs imposed by these protocols, the less public funding is needed. In Ethereum the cost of executing functionality on a contract is measured in *gas* which is paid for using *ether*, the built-in currency of Ethereum. This means that all execution carries a precise monetary cost.

A. *Selecting the delay function f*

When selecting the delay function the objective is that it can cheaply be verified relative to the time it took the prover to compute. The best delay function therefore minimizes the verification cost required per real-world compute time. We evaluate two basic approaches: using a cryptographic hash function and using modular square roots. As discussed in Section I, modular square roots currently provide the best known delay function with asymmetric verification time. However, Ethereum provides a built-in instruction for the Keccak-256 hash function [7] (now standardized as SHA-3) which costs only 40 gas, making it relatively cheap to verify by recomputation. Ethereum also provides a built-in instruction for SHA-256, although it costs more at which costs only 76 gas.

We implemented modular squarings in Solidity (i.e. the verification of a modular square root) computed the gas cost for 512-bit and 1024-bit moduli. We also implemented iterated Keccak directly in EVM (assembly language). We benchmarked computing both functions on an Intel Core i5-6360U CPU using OpenSSL. The results are provided in Table I.

As can be seen, the hash functions provide a better tradeoff today given the very cheap cost of computing them in Ethereum today. Keccak is marginally better than SHA-256, again due to the current gas price schedule. For the future, there exists a proposal[6] to add RSA signature verification as a cheap precompiled contract to Ethereum, which would change the calculation and likely favor modular square roots. RSA signature verification is essentially a modular squaring and such a precompiled contract would allow for cheap verification of modular square roots.

B. *Parameter selection and cost estimation*

We evaluated our prototype implementation of Protocol 1 for the multi-round protocol using Keccak-256 as the basic function f. In addition to selecting a core function f, we need to choose an iteration count t (with $F() = f^t()$), the number of checkpoints per rounds k and the number of iterations m performed by the contract in the final round of the protocol. We fixed $t = 2^{40}$ to simulate a very long delay function (on the fastest hardware available[7], this would take roughly 3500 seconds, or 1 hour, to compute) and selected the optimal m and k for different number of rounds.

In Figure 1 we present the costs for different optimal parameterizations given a fixed number of rounds. We computed the results for different values of k as additional rounds may have costs beyond the simple gas consumed . Most importantly, the number of rounds increases the time it takes to mount a challenge and therefore also the

[5]https://gist.github.com/anonymous/d30af9d7ea52c0882ed433cd675b5207

[6]https://github.com/ethereum/EIPs/issues/74

[7]according to https://bench.cr.yp.to

function	computation cycles	verification gas	verification USD	ratio USD/10^6 cycles
$\sqrt{x} \pmod{p}$, 256-bit modulus	10500	97	$1.32 \cdot 10^{-4}$	0.012
$\sqrt{x} \pmod{p}$, 512-bit modulus	20998	36,492	0.050	2.364
$\sqrt{x} \pmod{p}$, 1024-bit modulus	28070	109,882	0.149	5.324
Keccak-256 (SHA-3)	1233	48.6	$6.61 \cdot 10^{-5}$	0.054
SHA-256	1203	84.6	$1.15 \cdot 10^{-4}$	0.111

TABLE I: Computation costs (using data for AMD64;Skylake from https://bench.cr.yp.to) vs. verification costs (in Ethereum, expressed in both gas and US dollars at the exchange rate of at the time of this writing) for different delay functions. Computation costs are expressed in cycles on a skylake amd64 chip. Verification costs in Ethereum are expressed in both gas and US dollars, at the exchange rate of 1 ether $\approx$ \$68 and 1 gas $\approx 2 \cdot 10^{-8}$ ether. The computation/verification ratio is expressed in terms of the gas cost (measured in usd) of verifying a million cycles of computation.

overall challenge period even if no challenge is mounted. We therefore suggest setting the number of rounds to 6 ($k = 2^4, m = 2^8$) for $t = 2^{40}$ as it only has marginally higher cost than the optimum value of 9 rounds ($k = 2^3, m = 2^{10}$). The added benefit of 3 fewer rounds probably outweighs the slight increase in gas cost in practice.

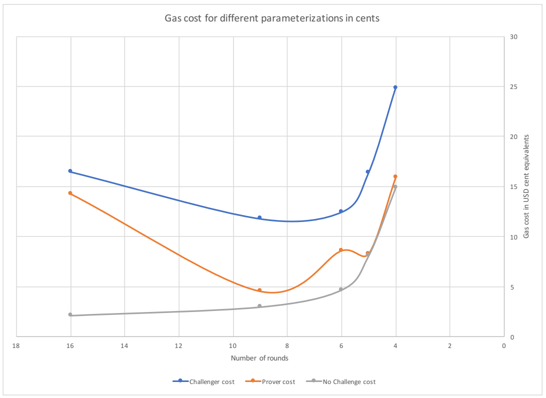

Fig. 1: Gas costs (in USD) for the protocol with $t = 2^{40}$, for different values of k (with m chosen optimally given the choice of k).

References

[1] Randao: A dao working as rng of ethereum. Technical report, 2016.
[2] B. Adida. Helios: Web-based open-audit voting. In *USENIX Security Symposium*, volume 17, pages 335–348, 2008.
[3] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure Multiparty Computations on Bitcoin. In *IEEE Security and Privacy*, 2014.
[4] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts. Technical report, Cryptology ePrint Archive: Report 2016/1007, https://eprint. iacr. org/2016/1007, 2016.
[5] T. Baignères, C. Delerablée, M. Finiasz, L. Goubin, T. Lepoint, and M. Rivain. Trap me if you can–million dollar curve. Technical report, IACR Cryptology ePrint Archive, 2015: 1249, 2015.
[6] I. Bentov, A. Gabizon, and D. Zuckerman. Bitcoin beacon. *arXiv preprint arXiv:1605.04559*, 2016.
[7] G. Bertoni, J. Daemen, M. Peeters, and G. Van Assche. Keccak. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 313–314. Springer, 2013.
[8] J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. *URL https://eprint.iacr.org/2015/1015.pdf*, 2015.
[9] J. Bonneau, A. Narayanan, A. Miller, J. Clark, J. A. Kroll, and E. W. Felten. Mixcoin: Anonymity for Bitcoin with accountable mixes. *Financial Cryptography and Data Security (FC)*, 2014.
[10] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.
[11] R. Canetti, B. Riva, and G. N. Rothblum. Practical delegation of computation using multiple servers. In *ACM CCS*, 2011.
[12] J. Clark and U. Hengartner. On the Use of Financial Data as a Random Beacon. *Usenix EVT/WOTE*, 2010.
[13] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. Technical report, DTIC Document, 2004.
[14] C. Dwork and M. Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, 1992.
[15] C. Garman, M. Green, I. Miers, and A. D. Rubin. Rational Zero: Economic Security for Zerocoin with Everlasting Anonymity. In *BITCOIN*, 2014.
[16] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 295–310. Springer, 1999.
[17] A. K. Lenstra and B. Wesolowski. A random zoo: sloth, unicorn, and trx. *IACR Cryptology ePrint Archive*, 2015:366, 2015.
[18] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf, 2008.

[19] R. Peralta et al. NIST Randomness Beacon. 2011.
[20] C. Pierrot and B. Wesolowski. Malleability of the blockchains entropy. 2016.
[21] M. O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.
[22] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford. Scalable bias-resistant distributed randomness.
[23] G. Wood. Ethereum: A secure decentralized transaction ledger. http://gavwood.com/paper.pdf, 2014.