Incentive-Driven Verifiable Random Beacons

Author(s) Removed For Review

***Abstract*—Random beacons—sources of open, public randomness—play a critical role in many protocols. Their uses range from proofs of earliest possible creation date to secure multiparty computation to lottery systems. In this paper we describe a random beacon that always requires misbehavior by a configurable number of parties to meaningfully affect the outcome. Our design introduces an economic security parameter, such that a colluding set of parties must forgo at least that much money in order to violate security requirements. As long as the adversary's gain from cheating is less than the economic security parameter, we assume the adversary will not try to cheat. Participants in the protocol are assumed to be willing to accept bribes from the adversary, but the economic security parameter ensures that bribery-based strategies are too expensive to be viable for the adversary. Our random beacon is designed such that any party can trust the output of the random beacon, even if they did not participate in the computation of the output, as long as they trust the public keys and specific universal constants. In order to further discourage dishonest behavior, every critical step of our protocol is third-party auditable and backed by incentives with a floor in cost that make honest behavior more profitable than dishonest behavior. Each step of our beacon is verified by a smart contract to provide monetary reward or penalty, as appropriate, to protocol participants, ensuring honesty is the best fiscal policy.**

I. Introduction

Random beacons are sources of public randomness: at a pre-specified interval, they publish pseudorandom numbers that are globally visible. Security requires that the value of a beacon not be predictable in advance, nor meaningfully controllable by an adversary, assuming that security parameters are suitably chosen. Random beacons are useful in a variety of applications; perhaps the largest class of problems random beacons can solve is making making decisions where consensus on a common, randomly-chosen string is needed, but a multiparty coin-flipping protocol is not sufficiently trusted or feasible. Examples of such situations are performing lottery draws, where there are too many parties for all of them to participate in a coin-flipping protocol, and determining common reference strings, where new parties may wish to join at a later date and may not trust the result of a coin-flipping protocol. Many proposals for random beacons exist, so we establish a random beacon whose security is incentive-based.

In this paper, we propose a new design for a random beacon we call an "Incentive-Driven Verifiable Random Beacon" (IVRB), which we construct by building a post-processing layer for a previous random beacon design. This new design allows us to avoid any single points of failure in the beacon, meaning that no single compromise can lead to any of the attacks in our threat model, as discussed in section III. Furthermore, while our protocol could be performed in a peer-to-peer setting, we demonstrate how a smart contract over a blockchain can be used to enforce honest behavior. By implementing our protocol over a smart contract, we show that when the the adversary's motivations and participants' incentives are properly modeled, honest behavior is more profitable than dishonest behavior.

Our design relies on smart contracts, which are supported by many cryptocurrency systems. Parties to our protocol commit to future behavior by putting down a deposit that they will recoup, plus a small reward, if they behave as promised; but the deposit will be lost if a party violates its commitment. If the deposit is large enough, a party has a strong incentive to behave as promised, unless the adversary offers a bribe larger than the deposit to induce misbehavior. We will then argue that the total cost of bribes by the adversary must exceed a security parameter. We explore the use of smart contracts further in section V-B.

II. Related Work

Random beacons were first introduced by NIST in 2011 for applications requiring "public, time-bound, and authenticated" random numbers. The most obvious applications are proof of earliest possible creation date and resolution of lotteries. NIST also proposed more sophisticated applications such as Unpredicatable Sampling, Secure Authentication, and Secure Multiparty Computation[1]. This simple beacon model requires complete trust in the honesty of the random beacon provider. To combat this, Clark *et al.* introduced a method to to draw on financial market data as a randomness source[2]. However, it was unclear if this source could be manipulated, and financial data is sometimes hard to access[3]. In response, Bonneau *et al.* proposed the use of the Bitcoin blockchain as the source of unpredictable data. This imposes an expected cost to manipulate the data source, since manipulation would require a party to forgo a Bitcoin mining reward[4]. However, Bonneau's beacon is still subject to manipulation by an adversary who somehow convinces miners to forgo collecting rewards when doing so benefits the adversary. Pierrot and Wesolowski demonstrated that large resources are not required for this type of attack[5]. Bentov, Gabizon, and Zuckerman examine the incentives more closely in their work[6].

Some recent approaches have relied on verifiable delay functions (VDFs). These schemes function similarly to Bonneau's beacon, except the final result is determined by a VDF, making it infeasible to compute the beacon output at the time of block publication[7][8]. These mitigate block withholding attacks, but require a large latency. The VDFs also require continuous use of computational resources, and they make assumptions about limitations on the adversary's computational resources that may not hold true in practice.

Syta *et al.* published two related methods for generating public randomness, dubbed RandHerd and RandHound[9]. Both methods use a commit-then-reveal approach, thresholded by Shamir's secret sharing, and verified by Lagrange interpolation, but differ in how they distribute the derivation of the randomness result. The first uses a leader who coordinates the results, and can provide a transcript as proof of validity to a third party. The second breaks generation into a tree-like structure, with a randomly assigned leader[9]. Their final protocol is fairly efficient, but leaves some room for performance improvements. It also doesn't guarantee complete unbiasability, since a dishonest leader can forfeit their leadership to withhold an undesired outcome. Their protocol also requires that at least 2/3 of nodes be honest, and doesn't provide much tunability based on threat model.

The most closely related approach to ours, from a cryptographic construction standpoint, is DFINITY's construction, which, while similar, was developed independently. DFINITY is a blockchain-based protocol for running verified computations on top of existing blockchain software. To acheive consensus, DFINITY has created a blockchain based random beacon that uses a thresholded scheme called BLS which was introduced by Boneh, Lynn, and Shacham[10][11]. However, DFINITY's work does not examine the incentive structures of their scheme. Our proposal includes a robust incentive structure, a key change scheme, and extensions to examine the value of a random beacon as a non-public good. A second, more minor, distinction is the source of the round input. Both protocols require an input for each round. For DFINITY, the input comes from the previous round's beacon output, whereas ours comes from a Bonneau beacon, making our approach more resistant to prediction attacks, which we define below in section III.

Another similar randomness generation approach was that of Algorand. In Algorand, a secret key is selected well in advance, and a single user computes the randomness for each round in a verifiable manner. In order to manipulate the result, many sequential such users would need to collude[12]. Algorand requires a trusted setup to start the randomness generation, and Algorand does not handle aborts in a way that avoids bias[11].

A. Verifiable Random Functions

Verifiable random functions (VRF) are a cryptographic primitive that act as a pseudorandom function (PRF) with a proof of correctness. Informally, a VRF can be seen as a PRF that provides a commitment to its secret key, and on every output it provides a proof that its output is the unique value that is consistent with that commitment. Before the output is published, a party without knowledge of the secret key is not able to gain any information about the output.

We define VRFs formally, closely following Dodis[13]. A VRF is a quadruple of algorithms that run in poly(k) time:

1) A generator algorithm $SK, PK = \texttt{Gen}(1^k)$
2) An evaluator algorithm $v = f_{SK}(x), f : \{0,1\}^{a(k)} \rightarrow \{0,1\}^{b(k)}$
3) A prover algorithm $\pi = \Pi_{SK}(x)$
4) A verifier algorithm $V(PK, x, v, \pi)$ that outputs "yes" or "no"

where v is the output, π is the proof, and x is the input on which to evaluate the function. VRFs also must satisfy the following properties:

1) **Soundness**: there exist no values $(PK, x, y_1, y_2, \pi_1, \pi_2)$, $y_1 \neq y_2$ such that $V(PK, x, y_1, \pi_1) = V(PK, x, y_2, \pi_2) = \texttt{yes}$
2) **Completeness**: If $SK, PK = Gen(1^k), v = f_{SK}(x), \pi = \Pi_{SK}(x)$, then $V(PK, x, v, \pi) =$ yes
3) **Pseudorandomness**: Let $rand(S)$ output a random element $s \in S$. For any PPT algorithm $A = (A_1, A_2)$ given an oracle to f, Π who does not query x, the following experiment fails with probability at most $\frac{1}{2} + negl(k)$

$$(PK, SK) \leftarrow \text{Gen}(1^k)$$
$$(x, st) \leftarrow A_1^{f_{SK}(\cdot), \Pi_{SK}(\cdot)}(PK)$$
$$y_0 \leftarrow f_{SK}(x); y_1 \leftarrow rand\left(\{0,1\}^{b(k)}\right)$$
$$b \leftarrow rand\left(\{0,1\}\right)$$
$$b' \leftarrow A_2^{\Pi(\cdot) f(\cdot)}(y_b, st)$$

The experiment succeeds if $b = b'$

This ability to provide an psuedorandom output and a corresponding proof of correctness make VRFs an appealing choice for random beacons. An input can be chosen in a public manner such that even if the adversary completely controls the input, they will not be able to predict any information about the output, and hence will be unable to manipulate the output. Because a corresponding proof is provided, third parties will believe the output was correctly computed, as long as they trust the public keys were distributed correctly.

Our work aims to produce a random beacon that is verifiable to third parties who were not involved in the beacon, is robust to single points of failure, and has an unbounded and adjustable cost to predict or manipulate. To accomplish this, we extend the economic incentives of Bonneau *et al.*'s approach[4] on top of the thresholded verifiable randomness generation of Hanke *et al.*'s approach[11], and we add two extensions that allow random beacons to be used as non-public goods. We introduce an adjustable minimum cost of misbehavior that exceeds the anticipated maximum benefit to

the adversary of that misbehavior. We also aim to improve the robustness of the system to bias by removing all single points of failure. We reduce the load and trust on our systems by offloading the distribution and verification to the blockchain. We approach this by using a weak verifiable random function modified to require a threshold of participants, along with mandatory deposits and fees to provide adjustable incentives for the participants.

III. Threat Model

In order to analyze beacon designs, we first need to establish a threat model. In our threat model, the parties in our protocol and our adversary are economically motivated. We imagine that for every round of our protocol, the adversary considers some outputs of the beacon to be favorable and the rest to be unfavorable.

We assume that we know an upper bound V on the monetary value (or utility) that the adversary places on achieving a favorable output rather than an unfavorable one, in each round of the protocol. This V acts as a security parameter in our model. By assumption, an adversary is unwilling to pay more than V to gain an advantage on a single random beacon output. We further assume that adversarial gain over multiple rounds follows the triangle inequality. In other words, if an adversary stands to gain V_1 from manipulating round a and V_2 from manipulating round b, then the adversary stands to gain at most $V_1 + V_2$ from manipulating rounds a and b together.

We assume our adversary is computationally bounded; that is, our adversary can only run algorithms that complete in probabilistic polynomial time (PPT). We denote the computational security parameter k. We assume that if the adversary promises to pay bribes to other parties conditional on their behavior, they will trust that the adversary will pay if they behave as requested.

We show security by showing that any computationally feasible strategy taken by the adversary to defeat the security properties of the random beacon will reduce the adversary's expected utility by more than V.

Because our primary goal is to remove bias in outputs, we prioritize integrity over availability. Therefore we do not require that the protocol be resistant to denial of service. We do require, however, that any denial of service attack is oblivious, in the sense that no adversary can deny service in a way that affects the ratio of expected favorable to expected unfavorable outputs. We assume that whenever the output of the protocol is suppressed, observers will always notice within a short time period, since beacon outputs are scheduled in advance. If the protocol fails to complete in the time required, we denote the output with the $\perp$ symbol. Users of the protocol will also treat syntactically invalid outputs as $\perp$.

Each round of the protocol defines two times $t_{\text{cutoff}} < t_{\text{publish}}$. Honest participants will publish the results of their shares of the protocol between these two times.

We consider two primary types of attacks: **prediction attacks** and **biasing attacks**. We formalize security properties against these attacks in section VI-A.

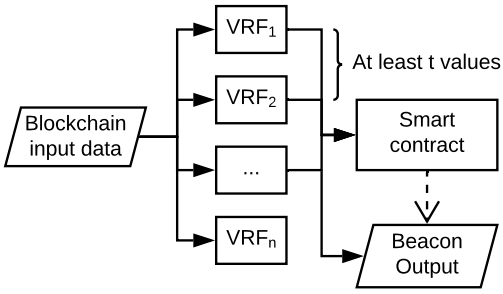

Fig. 1. An overview of our random beacon protocol. The "blockchain input data" would typically be Bonneau *et al.*'s random beacon. Each VRF_i is executed by an individual participant. Values are collected from at least t such VRFs, which a smart contract verifies and assembles into the beacon output.

In a **prediction attack** the adversary recovers useful, non-public information about the output before the cutoff time t_{cutoff}.

In a **biasing attack** the adversary biases the distribution of outputs to increase the ratio of the probability of a favorable output to the probability of an unfavorable output. The distribution of outputs may include $\perp$.

For simplicity, our incentive model does not include compromises of participants that occur due to intrusions. In a real world setting, the costs can be adjusted according to the security model. To make the adjustments, compromises can be modelled as 0 cost bribes. The incentive model would need to be adjusted accordingly, which we discuss in section V-A.

IV. Protocol

At its simplest, our protocol involves taking some universally available input x and passing it to a VRF $F_{SK}(x)$ with a proof $\Pi_{SK}(x)$, and using $F_{SK}(x)$ as the beacon output. We refine the details of this protocol to provide additional security guarantees.

To remove the single point of failure of a single machine computing F_{SK}, we distribute the key SK into n key shares, SK_is, where some threshold $t < n$ of output shares, $F_{SK_i}(x)$s, is sufficient to reconstruct $F_{SK}(x)$. Each SK_i is distributed to a different server we call "participants."

In order to create an incentive to publish their output share, we require every participant i put down a deposit d though a smart contract prior to engaging in the protocol. Immediately after the valid deposit is posted, a fee f is paid to a smart contract by a third party we refer to as the "coordinator". The smart contract will only pay a participant who provides an input (v, π) such that $V(PK, x, v, \pi) = \text{yes}$.

Since our beacon input x needs to be easily accessible and verifiable from the blockchain, we use the blockchain random beacon described by Bonneau *et al.*[4]. The input needs to be accessible from the smart contract. In this way, our protocol is a post-processing of Bonneau *et al.*'s random beacon to improve the security properties. The protocol as described to this point is illustrated in figure 1.

While our protocol does not rely on a trusted third party for security, the coordinator is entrusted with ensuring the economic viability of our protocol. The coordinator's roles involve creating appropriate smart contract(s) and posting fees to pay the participants. Should the coordinator fail to perform their duties, participants should be permitted to recover their deposits without posting the output to their evaluation of F_{SK_i}.

A. Choice of VRF

A number of VRF constructions have been proposed, most of which rely on the construction of a unique signature, also known as a verifiable unpredictable function (VUF), that is then transformed to a VRF through a standard, slow transformation. The original VRF construction of Micali depends on the RSA assumption[14]. In contrast, most other VRFs use variants of the Diffie-Hellman assumption, which often improved performance. Dodis and Lysyanskaya both propose VRFs that require multiple dependent rounds of execution[13][15]. For our application, having multiple rounds makes enforcing and fulfilling the smart contracts difficult, since failures on earlier rounds can lead to good actors not being paid, and success on earlier rounds can lead to bad actors recovering part of their deposit. Brakerski *et al.* propose a variant of a VRF called a weak VRF that provides marginally weaker soundness and completeness than a standard VRF. While weak VRFs remain secure to adversarially chosen keys, the pseudorandomness property only holds for random inputs[16]. The weak VRF is defined with the same quadruple of functions, but has the following properties:

1) **Relaxed Soundness**: For all but a 2^{-k} fraction of x values, and for all $(PK, y_1, y_2, \pi_1, \pi_2)$ such that $y_1 \neq y_2$ it holds that $Pr\left[V(PK, x, y_1, \pi_1) = V(PK, x, y_2, \pi_2) = \texttt{yes}\right] \leq negl(k)$.
2) **Relaxed completeness**: If $SK, PK = Gen(1^k), v = f_{SK}(x), \pi = \Pi_{SK}(x)$, then $V(PK, x, v, \pi) =$ yes, except with probability less than $negl(k)$
3) **Weak pseudorandomness**: Let $rand(S)$ output a random element $s \in S$. For any PPT algorithm A given an oracle to f, Π who does not query x, the following experiment fails with probability at most $\frac{1}{2} + negl(k)$

$$(PK, SK) \leftarrow Gen(1^k)$$

$$x \leftarrow rand\left(\{0,1\}^{a(k)}\right)$$

$$y_0 \leftarrow f_{SK}(x); y_1 \leftarrow rand\left(\{0,1\}^{b(k)}\right)$$

$$b \leftarrow rand\left(\{0,1\}\right)$$

$$b' \leftarrow A^{f_{SK}(\cdot), \Pi_{SK}(\cdot)}(PK, x, y_b)$$

The experiment succeeds if $b = b'$ and the inputs to the oracle queries were chosen randomly.

In the case of a weak VUF, instead of **weak pseudorandomness** we have **weak unpredictability**, defined as follows:

Weak unpredictability: Let $rand(S)$ output a random element $s \in S$. For any PPT algorithm A given an oracle to f, Π who does not query x, the following experiment fails with probability at most $\frac{1}{2} + negl(k)$

$$(PK, SK) \leftarrow Gen(1^k)$$

$$x \leftarrow rand\left(\{0,1\}^{a(k)}\right)$$

$$y \leftarrow f_{SK}(x)$$

$$y' \leftarrow A^{f_{SK}(\cdot), \Pi_{SK}(\cdot)}(x)$$

The experiment succeeds if $y = y'$ and the inputs to the oracle queries were chosen randomly.

We construct our beacon using the construction of a weak VRF from Brakerski *et al.*, because it does not require multiple sequential steps to compute in a distributed manner, and because we found it to be the most straightforward to modify. To address the requirement of random inputs and to facilitate the use smart contracts to validate the outputs, the Bonneau *et al.* beacon is a natural choice. We demonstrate the security of this approach in section VI-B.

We slightly modify Brakerski *et al.*'s construction using Dodis's distributed VUF construction to provide a (t,n)-threshold distributed VUF. To do this, we begin with the Brakerski VUF construction. Ahead of generation, public parameters $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g$ are chosen. $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ are groups of prime order, such that there is a known bilinear mapping $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}^T$, and $g \in \mathbb{G}_1$ is a generator of $\mathbb{G}_1$. On Gen, the secret key $a \in \mathbb{G}$ is chosen, and the corresponding public key g^a is published. The unique signature function is $F_a(x) = r^a$ where $r = \text{hash}_{\mathbb{G}_2}(x), r \in \mathbb{G}_2$, where x is the public input. The signature is verified by ensuring (g, r, r^a, g^a) forms a valid decisional Diffie-Hellman (DDH) tuple.[1]

We modify this VUF by distributing a with Shamir's secret sharing. A dealer chooses a polynomial $p_t(x)$, where $p_t(0) = a$. The dealer then secretly sends $a_i = p_t(i)$ to participant i, and publishes public key g^{a_i}. If the dealer is not trusted, the keys can be immediately refreshed as described in section IV-B. Each participant now has a secret key for an independent Brakerski weak VUF, but when at least t participants' outputs are known, the value r^a can be computed by Lagrange interpolation in the exponent of the r^{a_i} values. Likewise, individual VUF output can be verified by ensuring (g, r, r^{a_i}, g^{a_i}) is a valid DDH tuple, and the global VUF output can be verified by ensuring (g, r, r^a, g^a) is a valid DDH tuple. This modification uses the same method as the conversion Dodis proposes to his VRF to support thresholding[13].

To convert our VUF to a VRF, we have two options. We can use the transformation described by Micali, which requires computing multiple parallel VUF outputs[14]. However, this would increase the communication complexity and data stored on the blockchain, which is not ideal. It would also increase

[1]Under a bilinear mapping, DDH is easy: to check $c = a \cdot b$ in (g, g^a, g^b, g^c), we check if $e(g, g^c) = e(g^a, g^b) \implies e(g, g)^c = e(g, g)^{ab} \implies c = a \cdot b$.

the computation time required by the smart contract, since all outputs would need to be verified. In practice, we rely on a hash as an approximation of a random oracle, and let $F_{SK}(x) = \text{hash}\left(\Pi_{SK}\left(x\right)\right)$.

The soundness and the unpredictability properties of VRF outputs make manipulation attacks against our protocol very difficult. We show this more formally in section VI.

B. Key refreshing

To help mitigate leakage of secret keys, VRF keys can be refreshed. A party c, the "leader", picks some random value b_c and chooses some random polynomial $p_c(x)$ of degree $t-1$ such that $p_c(0) = b_c$. c publishes g^{b_c}. For each participant i, c secretly sends participant i the value $p_c(i)$ and publishes $g^{p_c(i)}$. Observers confirm the Lagrange interpolation of the $g^{p_c(i)}$s is g^{b_c}. If failure is recorded, the changes from this polynomial must be discarded, and the cause of failure can be investigated offline. In the case of a failure, a replacement leader can be chosen to execute the key refresh protocol.

Several such leaders can perform this protocol simultaneously. After all leaders have submitted, each participant computes its new secret

$$a_i' = a_i + \sum_c p_c(i)$$

All observers compute the new participant commitments

$$g^{a_i{}'} = g^{a_i} \cdot \prod_c g^{p_c(i)}$$

and the new VRF commitment $g^{a'}$ as the Lagrange interpolation in the exponent of t values of $g^{a_i{}'}$.

To ensure integrity, each participant i locally computes its public key from its secret key, and verifies that this matches the published public key. Using Lagrange interpolation on at least t public keys that do not include its own, the participant recovers $g^{p_c(x)}$, and recovers an interpolated value for $g^{p_c(i)}$, and verifies that this matches the published public key. If either of these checks fail, the participant indicates a protocol failure and the new keys from c must be discarded.

The outputs after a key change cannot be predicted as long as any one party c is honest. If C parties contribute to the refreshing, then the overall communication complexity will be $O(Cn)$. It may be desirable to only perform refreshing every $O(n)$ rounds to reduce the cost down to an amortized $O(C)$ per round, or to perform the key refresh protocol with a constant number of parties, giving communication complexity of $O(n)$ per round.

We recommend that the submitting parties be the participants, since the participants already play a role in the protocol, and it is reasonable to assume at least one of the participants will be honest. Strictly speaking, however, using the participants as the submitting parties is not necessary for security; it is only necessary that one submitting party be honest.

V. Incentives

In this section we develop an incentive system to enforce honest behavior. These incentives creates a cost to various modes of failure for our beacon. These incentives are modeled such that they can defend against adversaries that are arbitrarily well funded, albeit with linearly increasing costs and practical limitations.

Broadly speaking, we want to reward compliance with the protocol and punish misbehavior. Our goals with the incentive structure can be broken into two behaviors we target:

1) Ensure participants produce the correct output at an appropriate time.
2) Ensure participants do not leak outputs for unpublished inputs, where an unpublished input is any input that was not used for a previous beacon output and is not in use for the current beacon output.

In section V-A, we discuss how we structure our incentives. In section V-B, we discuss how we enforce our incentive structure through a smart contract.

A. Incentive structure

Let a V-adversary be an adversary who is willing to forfeit monetary value of at most V to predict or bias the output of the random beacon.

As we will show in section VI-B, since we use a (t, n) threshold system, an adversary will need to bribe $n - t + 1$ participants to affect the protocol's output. Then $b_{\text{max}} \leq V/(n - t + 1)$ is the maximum bribe the adversary will be willing to pay each of these participants. We can establish a minimum bribe value b_{min}, that a participant will insist on receiving, by enforcing a loss of $-b_{\text{min}}$ for any participant who misbehaves. A trivial solution would be to provide a payout of b_{min} to any participant who provides an output that passes verification. However, this is likely infeasible, since these payouts would likely have to be quite large. Instead, we require each participant to pay a deposit d to participate, and on publishing their verifiable output the participant recovers their deposit and receives an additional payout f. To prevent bribing participants into not publishing, we choose some $b_{\text{min}} > b_{\text{max}}$ and d, f such that $d + f > b_{\text{min}}$, giving us $(n-t+1)(d+f) > V \implies d > \frac{V}{n-t+1} - f$. In practice, we expect f to be a fixed value.

Lemma 1. *If the adversary needs to bribe $n - t + 1$ participants to prevent publication, then preventing publication through bribery is infeasible for a V-adversary if $(n - t + 1)(d + f) > V$.*

The approach described so far still allows for participants to reveal unpublished outputs at an arbitrarily small cost with no penalty. Therefore, we need to provide an additional incentive for participants not to reveal such outputs. We can place a bounty on unpublished outputs – that is, the valid output for any valid input that doesn't match any previous input seen

in the protocol.[2] Then, any participant who lowers sells their secret key for less than the bounty price risks having their bounty claimed by someone else, leaving them with a loss. Let W be the value to the adversary of knowing the secret key a. If we require each participant to place their deposits several rounds in advance, such that there are always at least k deposits placed by each participant at any given time, and we use the entire deposit and fee as the bounty, we get a bounty of $k(d+f)$, which becomes the minimum price at which a participant will be willing to sell their secret key. Since the adversary must recover t secrets to mount a successful attack, we must satisfy $t \cdot k(d+f) > W$. If we let keys be refreshed every l rounds[3], then, by the triangle inequality, we have $W = l \cdot V$, so $\frac{t \cdot k}{l}(d+f) > V$.

Lemma 2. *If the adversary needs to bribe t participants to learn the beacon output for a unpublished input, then learning the beacon output for an unpublished input is infeasible through bribery for a V-adversary if $\frac{t \cdot k}{l}(d+f) > V$.*

To avoid the risk that a participant claims their own bounty as a mechanism for a selective suppression – that is, a participant expects an undesirable output, and therefore claims their own bounty to back out at no cost – we only pay out submission rewards and deposits at the next output round. In the time between, a bounty can be claimed on that reward and deposit, redirecting that reward and deposit to the claimant. In this way, the output must be published for a bounty to be claimed.

Additional considerations must be made for the costs of running an honest protocol. The cost per round to pay fees is $n \cdot f$. Considering deposits alone, the minimum participant capital for entry is $k \cdot d$, while the maximum return on investment for participants is then $\frac{f}{k \cdot d}$. Since participants will have additional setup, upkeep, and transaction fees (e.g. equipment, rent, utilities, maintenance, patches, etc.), particularly since high availability is required, the return on investment (ROI) must be carefully considered and should be chosen to minimize opportunity cost to participants while also minimizing fee costs.

Recall that we have defined the following variables:

1) V: The anticipated maximum value or utility the adversary stands to gain through dishonest behavior in one round
2) n: The total number of participants
3) t: The threshold, i.e. the number of participants needed to reconstruct the output
4) d: The value of the deposit each participant must place
5) f: The reward paid after each output publication to honest participants
6) k: The number of simultaneous deposits; approximately the number rounds in advance deposits are placed
7) l: The number of rounds between each key refresh

And we've identified the following constraints to satisfy:

1) $(n-t+1)(d+f) > V$
2) $\frac{t \cdot k}{l}(d+f) > V$
3) Participant ROI = $\frac{f}{k \cdot d}$
4) Cost to coordinator per round = $n \cdot f$

Since there's little value in greatly exceeding V, we can assume: $\frac{t \cdot k}{l} = (n-t+1)$, so our threshold becomes

$$t = \frac{n+1}{\frac{k}{l}+1} \approx n\frac{l}{k}$$

The total cost is more interesting in terms of V, so

Cost per round $> n \cdot \left(\frac{V}{n-t+1} - d\right) = n \cdot \left(\frac{V \cdot l}{t \cdot k} - d\right)$

As we discussed at the end of section III, in a real world setting, intrusions are possible. Let c be the upper bound on the number of intrusion, then then the threshold value t is substituted for $t-c$ for constraint 1, and substituted for $t+c$ for constraint 2.

B. Smart contract enforcement

In order to enforce the aforementioned incentives, we propose the use of a smart contract to collect deposits, pay rewards, and manage bounties. Our goal is that we pay participants who comply with the protocol, and we punish participants who do not. As long as some threshold comply with the contract, our smart contract should ensure our security guarantees are met. We present pseudocode for the smart contract in appendix B.

The timing of events is as follows. First the participants place their respective deposits to the smart contract at some time in advance of a designated "input" block, then the manager provides the smart contract the rewards to distribute. When the designated block is published, participants publish their output. After the bounty claim time has passed, the deposit and reward are paid to the participants who published. These events are also shown in figure 2. The first such block is chosen as some block after the system is initialized, and all future blocks are a fixed interval after that. That is, suppose there is some block number b_1 that is the designated first output time, and new outputs are desired every δ blocks, then the i^{th} designated block occurs at block number $b_i = b_1 + i\delta$.

In the case where the protocol fails, expired contracts must result in the funds being destroyed, in order to avoid the case where the recipient of the funds of a dead contract colludes with a participant to reduce the disincentive. In the case where the protocol succeeds, some penalty should still be levied, but the penalty need not be the full forfeiture.

A cryptocurrency that supports this contract must support computing the verification functions. For our choice of weak VRF this involves arithmetic on large numbers under a modulus as well as bilinear map and respective group operations. The cryptocurrency also needs to support detecting the current block number and retrieving the hash for past blocks. This last requirement rules out Bitcoin, but Ethereum remains as

[2]In order to remain secure under the weak pseudorandomness property, the input provided should be a hash preimage of the actual input. The inability to produce such a proof without some type of oracle access of this is immediate from lemma 3 in section VI-B.

[3]specifically, l is how many rounds will pass before the keys are refreshed by an honest submitter

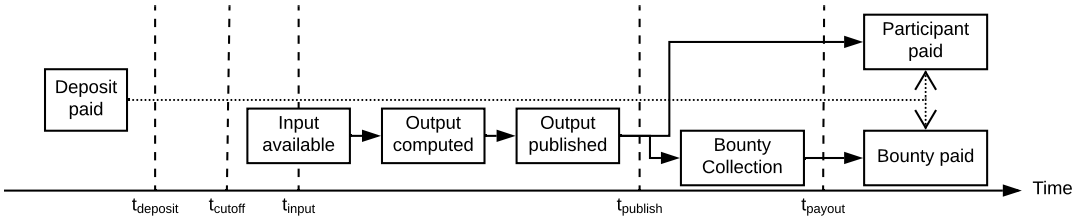

Fig. 2. The timeline of a single round of the smart contract; rounds can be pipelined. There are 5 key times: t_{deposit}, the latest possible time for a participant to pay the deposit; t_{cutoff}, the latest possible time for a user to make a decision from the beacon output; t_{input}, the time when the input becomes available; t_{publish}, the time by which participants must publish to receive their deposit and the reward; and t_{payout}, the time by which bounties must be submitted. Immediately after t_{payout}, deposits are refunded and rewards are paid, to either the participant or bounty collector depending on the circumstances.

a somewhat viable choice for implementation. We discuss the difficulties with an Ethereum implemention in section VII-B.

Note that claiming someone else's reward is preceded by the participant publishing their output, so there's never a situation where a participant would claim their own bounty to avoid publishing their output.

VI. Security

In this section, we will sketch a proof of the security of our protocol. We break our argument into three steps. First we briefly define what security means for our protocol, and we consider how the security of weak VUFs imply the security of our protocol, then we consider the security of our distributed computation modification to Brakerski's weak VUF, finally we consider the security of refreshing the keys. For second and third sections, we show that these modifications maintain the original security properties of Brakerski's weak VUF. The definition for weak VUFs can be found in section IV-A.

A. Security Definitions

In this section, we very define the security goals of the protocol as a whole. We say a random beacon RB is (V,k)-secure if it is secure against adversaries with V bribing power and $poly(k)$ computational power. Let $\text{RB}_{SK}(z)$ be the random beacon output for time $z \in \mathbb{Z}$ and secret key SK; let $\text{RB}^i_{SK_i}(z)$ be the random beacon share for participant i, time z, and participant secret SK_i. In a distributed setting, $\text{RB}_{SK}(z)$ is composed of any t of n shares $\text{RB}^i_{SK_i}(z)$, which we denote with the combine function: $\text{RB}_{SK}(z) = combine(O)$ where O is a mapping such that $O : i \mapsto \text{RB}^i_{SK_i}(z), |O| \geq t$. If the adversary attempts to influence $\text{RB}_{SK}(z)$, then let $z \in \mathbb{Z}[i]$,[4] where $\text{Re}(z)$ is the beacon output time, and $\text{Im}(z)$ is the adversarial action, with $Im(z) = 0$ corresponding to no adversarial action.

As we outlined in on our threat model in section III, we have the following desired properties:

1) **Prediction Resistance**: Predicting the output requires compromise of at least t participants. More formally, let C be the set of compromised participants $C \subset [n]$ and $S = \{SK_i | i \in C\}$ be their keys. Then we say RB has **prediction resistance** if for all PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following experiment succeeds with probability at most $negl(k)$:

$$SK, PK \leftarrow Gen(1^k)$$
$$C, st = \mathcal{A}_1^{\text{RB}_{SK}(\cdot),\text{RB}^1_{SK_1}(\cdot),\ldots,\text{RB}^n_{SK_n}(\cdot)}(PK)$$
$$S = \{SK_i | i \in C\}$$
$$z, v' = \mathcal{A}_2^{\text{RB}_{SK}(\cdot),\text{RB}^1_{SK_1}(\cdot),\ldots,\text{RB}^n_{SK_n}(\cdot)}(st, PK, S)$$
$$v = \text{RB}_{SK}(z)$$

The experiment succeeds when $v \neq \perp \wedge v = v'$, and for all z' accessed by oracle queries, $Re(z') < Re(z)$

2) **Unbiasability**: Changing the output requires compromise of at least t participants. More formally, let $p \in [0,1]$ be a probability and $\mathcal{R} : \{0,1\}^{b(k)} \cup \{\perp\} \to \{0,1,\perp\}$ be a predicate for which $p \cdot b(k)$ inputs map to $\{0\}$ and $(1-p) \cdot b(k)$ inputs map to $\{1\}$, where $\mathcal{R}(\perp) = \perp$. We say RB has **unbiasability** if for all PPT algorithms $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, the following experiment succeeds with probability at most $\max(p, 1-p) + negl(k)$:

$$SK, PK \leftarrow Gen(1^k)$$
$$C, z, \mathcal{R}, b' = \mathcal{A}_1^{\text{RB}_{SK}(\cdot),\text{RB}^1_{SK_1}(\cdot),\ldots,\text{RB}^n_{SK_n}(\cdot)}(PK)$$
$$\text{where } b' \in \{0,1\}$$
$$S = \{SK_i | i \in C\}$$
$$O = \{\text{RB}^i_{SK_i}(z) | 1 \leq i \leq n\}$$
$$O' = \mathcal{A}_2^{\text{RB}_{SK}(\cdot),\text{RB}^1_{SK_1}(\cdot),\ldots,\text{RB}^n_{SK_n}(\cdot)}(PK, S, O)$$
$$\text{Where } \forall i \in C, i \in O' \wedge \forall i \notin C, i \notin O'$$
$$O'' : i \mapsto \begin{cases} O'_i & \text{if } i \in C \\ O_i & \text{if } i \notin C \end{cases}$$
$$v' = combine(O'')$$
$$b_{\text{new}} = \mathcal{R}(v')$$
$$O_{\text{original}} = \{\text{RB}^i_{SK_i}(Re(z)) | 1 \leq i \leq n\}$$
$$v_{\text{original}} = combine(O_{\text{original}})$$

[4]The Gaussian Integers, which are complex numbers where both the real and imaginary parts are integers i.e. $\mathbb{Z}[i] = \{a + b\ i | a, b \in \mathbb{Z}\}$

$$b_{\text{original}} = \mathcal{R}(v)$$

The experiment succeeds when $b_{\text{new}} = b' \vee (b_{\text{original}} \neq b' \wedge v' = \perp)$, and for all z' accessed by oracle queries, $Re(z') < Re(z)$.

3) **Pseudorandomness**: Let $rand(S)$ output a random element $s \in S$. For any PPT algorithm $A = (A_1, A_2)$ given an oracle to f, Π who does not query x, the following experiment fails with probability at most $\frac{1}{2} + negl(k)$

$$(PK, SK) \leftarrow \text{Gen}(1^k)$$
$$(z, st) \leftarrow A_1^{RB_{SK}(\cdot)}(PK)$$
$$y_0 \leftarrow RB_{SK}(z); y_1 \leftarrow rand\left(\{0,1\}^{b(k)}\right)$$
$$b \leftarrow rand\left(\{0,1\}\right)$$
$$b' \leftarrow A_2^{RB_{SK}(\cdot)}(y_b, st)$$

The experiment succeeds if $b = b'$, and for all z' accessed by oracle queries, $Re(z') < Re(z)$.

B. Protocol security

The prediction resistance and unbiasability of our random beacon comes primarily from the thresholding secrecy and completeness properties of our modified weak VUF, defined in the next section, VI-C.

Prediction resistance is evident from the thresholding secrecy property that any group of less than t participants cannot learn any additional information about the output or keys, as long as $|C| < t$. Then, the condition $|C| < t$ is fulfilled by lemma 2, giving us prediction resistance.

Our analysis of **unbiasability** considers unbiasability from two angles: manipulation and selective suppression. Manipulation is any attack that provides a $\frac{1}{poly(k)}$ advantage in ensuring that $b_{\text{new}} = b'$. Selective suppression is any attack that provides a $\frac{1}{poly(k)}$ advantage in ensuring that $v' = \perp$ when $b_{\text{original}} \neq b'$.

In order to improve the probability that $b_{\text{new}} = b'$, the adversary needs to predict the value of b_{new} with non-negligible probability, when $|C| \leq t - 1$. This violates the **prediction resistance** property.

In order to improve the probability that $v' = \perp$ when $b_{\text{original}} \neq b'$, the adversary will have to induce a $\perp$. There are two ways to do so: by changing the $Im(z)$ value to one that induces a $\perp$, or by bribing participants to not publish. For the former, this would require predicting the outcome before choosing z with non-negligible probability, violating **prediction resistance**. For the latter, from the thresholding completeness property of our modified weak VUF, the adversary would need $|C| \geq n - t + 1$ to cause a $\perp$. By lemma 1, we can fulfill the condition that $|C| < n - t + 1$. Therefore, the adversary is not able to induce a $\perp$. Since both of these approaches are mutually incompatible, the probability of success is the maximum of the two, which provides only a negligible advantage.

We can see that both approaches to bias attacks have at most a negligible advantage. Because the probabilities of the success of the two approaches are joined with an OR, any combination of approaches can at most benefit from the summation of their advantages, which is still a negligible advantage.

The **pseudorandomness** of the beacon depends on ensuring the weak unpredictability property of the weak VUF is preserved when using a Bonneau beacon. The Bonneau beacon uses a randomness extractor in their protocol; assume, under the random oracle model, that this extractor is a random oracle. For the sake of argument, assume the adversary can choose the input value to the random oracle. Then, by lemma 3 below, the beacon remains pseudorandom.

Lemma 3. *If the adversary is allowed to choose between a polynomial number of randomly chosen VUF inputs, the weak unpredictability property of the weak VUF still holds.*

Proof sketch: Let Q be the set of outputs from queries to the random oracle, then let the adversary choose one particular value $q \in Q$ as the input for the weak VUF. If we provide the adversary the set $\{f_{SK}(q')|q' \in Q \setminus \{q\}\}$, then these values become oracle queries, and this situation matches the definition of the weak unpredictability property. Hence, an adversary choosing among $poly(k)$ queries to a random oracle preserves the weak unpredictability property.

C. Distributed weak VUF

In this section, we show that relaxed completeness, relaxed soundness, and weak unpredictability hold for our distributed weak VUF. We also define properties of thresholding. The properties shown in this section are largely derived from the properties of Dodis's VRF[13].

Since, as a whole, our distributed weak VUF has a corresponding weak VUF with identical inputs and outputs, completeness and soundness clearly hold.

In order to show weak unpredictability holds for our scheme, we show that our scheme is equivalent to a single step of Dodis's distributed VRF. Since that VRF has stricter pseudorandomness properties, a PPT algorithm that predicts our output would be able to predict Dodis's output, violating the standard VRF pseudorandomness property, hence also violating the weak unpredictability property.

As a reminder, Dodis's VRF involves creating a set of secret keys a_j for each round in his protocol. The input x is run through a binary encoding function C, and the resulting VRF output and proof are both $g^{\Pi_j a_j^{C_j(x)}}$. This is performed in rounds where $g_0 = g$ and

$$g_j = \begin{cases} g & \text{if } j = 0 \\ g_{j-1} & \text{if } C_j(x) = 0 \\ (g_{j-1})^{a_j} & \text{if } C_j(x) = 1 \end{cases}$$

The computation of $(g_{j-1})^{a_j}$ is distributed in an identical manner to our distribution of r^a. If we assume there exists a PPT algorithm to predict r^a more a small fraction better than random, then we can use that same algorithm to predict g_j from g_{j-1}. This would violate the pseudorandomness property

of Dodis's VRF, which is a contraction. Hence such an algorithm cannot exist.

We additionally define the following thresholding properties:

Thresholding completeness: *Let x be an input to a weak VUF evaluation function f_{SK}. Let SK be distributed into n shares $SK_1, ..., SK_n$ with a threshold of t. Then any subset of size t of $f_{SK_i}(x)|i \in [n]$ is sufficient to compute $f_{SK}(x)$*

Thresholding secrecy: *Let x be an input to a weak VUF evaluation function f_{SK}. Let SK be distributed into n shares $SK_1, ..., SK_n$ with a threshold of t. Then any subset of size less than t of $f_{SK_i}(x)|i \in [n]$ can only compute $f_{SK}(x)$ with negligible probability.*

Thresholding completeness is immediate from the construction of our distributed weak VUF.

Thresholding secrecy is immediate from the security of Dodis's VRF construction.

D. Key refreshing

In this section, we show that relaxed completeness, relaxed soundness, and weak unpredictability hold for the distributed weak VUF after using our key refreshing scheme.

Since this mechanism does not produce invalid keys, and F_{SK} of our modified weak VUF is not changed, relaxed completeness and relaxed soundness clearly hold.

We now need to show that the weak unpredictability property still holds. Since no weak keys exist, any state the adversary puts the system will not introduce any new predictive ability. However, to show weak unpredictability still holds, we need to show that the state transition does not improve the adversary's predictive ability. We note that no new information about the secret keys is gained. Everything published by participants is computable by the adversary from the adversary's knowledge of the secret key adjustments. The adversary, therefore, gains no more information about the system. Since the adversary gains no new information and is unable to force the system to use a weak key, the adversary is not able to predict the output any better and the weak unpredictability property still holds.

We also show that key reuse is not a concern after the key is refreshed, provided the adversary does not know the key change. Assume there is some PPT algorithm to predict $r^{a+\Delta a}$ we denote as $\mathcal{A}\left(g, r, g^a, g^{\Delta a}, r^a\right)$, with $a' = a + \Delta a$ and $g^{a'} = g^a \cdot g^{\Delta a}$. Suppose have some g, g^x, g^y. We pick any value c and compute

$$\begin{aligned}
&\left(g^c\right)^{-1} \mathcal{A}\left(g, g^y, g^c, g^x, \left(g^y\right)^c\right) \\
&= \left(g^c\right)^{-1} \cdot \left(g^y\right)^{c+x} \\
&= g^{xy}
\end{aligned}$$

Which violates the computational Diffie-Hellman hardness assumption. Hence, no such algorithm exists.

Even if the adversary contributes to a', it is sufficient that any one contributor not share their contribution with the adversary to keep the adversary from predicting $r^{a'}$.

VII. Implementation

We have provided a proof-of-concept implementation. This comes in 2 parts. The first part is the beacon implemented on a model blockchain, demonstrating output verification and public key refreshing. The second part is a partial implementation of the beacon in an Ethereum smart contract, demonstrating output verification and monetary incentives. Due to limitations in Ethereum's pairing support, we were not able to implement public key refreshing in the smart contract.

The model blockchain implementation has 2783 lines of code written in Java and runs over a network. The Java implementation depends on the Java Pairing-Based Cryptography (JPBC) library, which is a Java interface for the Pairing-Based Cryptography (PBC) written in C. [17] [18]

A. Model Blockchain

We provide a sample implementation of the full protocol over a mock blockchain. We have broken this implementation into two parts: an event-based abstract beacon, and an implementation on a mock blockchain.

Our abstract beacon is broken down into Participants and Users. Where a Participant watches the blockchain to see when they should publish the next output and waits for new secret key changes to track and publish corresponding public key changes to. Users watch for outputs and public key changes on the blockchain to locally record those. Users can also submit secret key changes, but in practice the only users authorized to do so would likely be those also holding a Participant.

Our full implementation provides a networked implementation of the beacon, on a model blockchain. This blockchain collects entries over a fixed time period, then broadcasts all entries collected since the last period, simulating the process of periodically appending to a ledger. The machine that hosts the blockchain is programmed with behavior to validate beacon outputs and key changes, simulating a smart contract. For each beacon output, one participant is selected round robin to make a key change, which entails a single participant performing the steps outlined in section IV-B. After N rounds, a full refresh will have occurred.

To evaluate our implementation, we set up a network on Google Cloud Platform[19]. Our network places each participant and the host on separate VM instances, scattered geographically across Asia, North America, Europe, South America, and Australia. All VMs were run on Google Cloud Platform's "g1-small" machines, which are stated to have 1.7GB of memory and 0.5 virtual CPUs. We ran the protocol run until 100 beacon outputs had accumulated, at which point we terminated the execution and collected measurements.

The results from our evaluation are shown in figure 3. We can see that the computation time for the participants does not vary much between the number of participants. This would be expected, since, relative to the number of participants, the beacon output operations are constant time complexity, and the key refreshing amortizes to constant time complexity. We can also see that the computation time climbs for the host, which is again unsurprising since the time complexity

Num. of Part.	Thresh.	Host CPU Time	Avg. Part. CPU Time (Std. Dev.)	Network Data
3	2	0.22 s	0.48 s (0.006s)	2.42 kB
5	3	0.62 s	0.37 s (0.078s)	3.95 kB
10	7	1.65 s	0.52 s (0.094s)	7.84 kB

Fig. 3. The measurement results from our full protocol implementation with a simulated blockchain and smart contract. The host simulates both the blockchain and the smart contract. The participants each simulate a single IVRB participant. The CPU times are per beacon output and per participant. CPU times do not include sleep time and time to send and receive network communications. Network data is the sum of the data sent by all participants and the host per beacon output. Total run time is not included since the test includes a simulated blockchain that enforces fixed block times, so most of the run time is spent sleeping. All tests were run with 100 beacon outputs. The total cost to execute all 3 simulations was approximately USD$0.55.

to compute output values is quadratic relative to the number of participants.

B. Ethereum smart contract

We implemented of a reduced version of our protocol in an Ethereum smart contract. In particular, our implementation provides the following features:

- Deposit collection and tracking
- A mechanism to compute a universal r
- Participant submission, verification and tracking
- Deposit and fee payout
- Bounty submission and payout

We also provide a sketch of public key tracking. Due to the limited field and group operations implemented natively in Ethereum, we were not able to implement the following

- Combined output computation
- Public key update verification and computation
- Provably secure determination of r, which should be done by hashing onto the curve, as described by Fouque *et al.*[20]

At the start of the protocol, the participants are identified having by their Ethereum public key hard-coded into the contract. Other parameters are also hard-coded into the contract, such as the deposit amount, time interval between outputs, etc. Without public key update verification, public keys would need to be hard-coded as well. However, with public key update verification, public keys could be initialized to zero, and updated immediately.

In order to facilitate zk-SNARKs, Ethereum has natively implemented some finite field operations and group operations on the Barreto-Naerhig curves[21][22]. Let $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ be the elliptic curve groups that comprise the Barreto-Naerhig curves. Then the implemented operations are:

1) Scalar exponentiation under modulo: $b^e \mod m$ for large integer inputs b, e, m
2) Addition in $\mathbb{G}_1$: $X + Y$ for inputs $X, Y \in \mathbb{G}_1$
3) Scalar multiplication in $\mathbb{G}_1$: aX for inputs $a \in \mathbb{Z}_{2^{256}-1}$ and $X \in \mathbb{G}_1$
4) Pairing product check: $\prod_{i=0}^{k} e(a_i, b_i) = e(P_1, P_2)$ for fixed generators P_1, P_2, and inputs $a_0, a_1, ..., a_k \in \mathbb{G}_1$, $b_0, b_1, ..., b_k \in \mathbb{G}_2$

In order to implement our full protocol, we would additionally need

1) Scalar addition under modulo: $a + b \mod m$ for large integer inputs a, b, m
2) Scalar multiplication under modulo: $ab \mod m$ for large integer inputs a, b, m
3) Addition in $\mathbb{G}_2$: $X + Y$ for inputs $X, Y \in \mathbb{G}_2$
4) Scalar multiplication in $\mathbb{G}_2$: aX for inputs $a \in \mathbb{Z}_{2^{256}-1}$ and $X \in \mathbb{G}_2$

Test case	Cost: Gas (USD)
Contract initialization	3,985,409 ($2.39)
Submit an output (keccak256)	497,421 ($0.30)
Claiming a bounty (keccak256)	891,901 ($0.54)
Submit an output (HMAC)	510,917 ($0.31)
Claiming a bounty (HMAC)	904,690 ($0.54)

Fig. 4. Gas cost for each test. "keccak256" vs "HMAC" refers to the extractor function used to compute r. The contract initialization only needs to be run once and applies globally. Submitting outputs will occur once per participant per round in normal operation. Claiming bounties is not expected to happen frequently in normal operation. USD estimates are rough estimates only, and are subject to significant change as the cost of Gas and Ether change.

Ethereum contracts are limited in their computation time through a resource called "Gas". In order to ensure our system will stay within reasonable gas consumption, we created a method to test the output submission functionality and a method to test the bounty claiming functionality. We show the Gas usage of each of those tests in Figure VII-B. For reference, the global gas limit, or the sum total of gas that can be used for a block, across all smart contracts, is 8 million Gas, as of the writing of this paper[23][21]. Additionally, as of the writing of this paper, the expected cost of one Gas is 5 Gwei[23], which translates to USD6×10^{-7}[24].

VIII. Paying for fees

While we have addressed economic incentives for participants, we have not discussed how to raise the funds to pay the fees to participants. In the simplest situation, a publicly-minded, well funded actor, such as a government, agrees to cover the fees. Otherwise money will have to be raised somehow. To do this, we propose taking random beacons slightly outside the realm of a public good. By definition, public goods are both non-rivalrous, meaning that one party's use does not diminish another party's utility; and non-excludable, meaning that it is not possible to prevent a particular party from using the good[8]. We propose an extension for making a beacon excludable, which we call "proof withholding"; and

we propose an extension for the case where a beacon may be rivalrous, which we call "output withholding"

A. Proof Withholding

In this extension, the beacon value is published with no associated proof. In order to verify the output, a user must pay to receive a proof of the output. An interactive proof would need to be performed with each participant. A non-interactive proof would allow the user to share the proof, which would make the good public.

This is trivially implemented by switching to a field where DDH is hard, and using a discrete logarithm proof in zero knowledge[25].

The difficulties here are twofold:

1) Avoiding freeloading
2) Ensure those who want to pay can pay

For the first, we are concerned about the case where a group collaboratively creates the challenges to a zero-knowledge proof, enabling them to share the proof among themselves. We address the first by requiring the verifier commit to their challenge bit before the beginning of each round, and reveal their commit after the round. This guarantees the verifier knew the challenge bit, and therefore could have colluded with the prover. This makes the proof only useful to the verifier.

For the second, while we do want to exclude those who do not pay, we also do not wish to exclude those who are willing to pay. While, ideally, most of the verification should be solved off-chain, it may be the case that a malicious participant chooses not to engage in the interactive proof with a verifier. To solve this, each participant must have have a smart contract over which a single step of the zero knowledge proof is performed, which includes the aforementioned challenge commitment scheme. If the participant answers the challenge, the verifier will refund the participant's gas costs as well as pay a fixed fee, through a deposit placed when the verifier starts the challenge. Should a participant fail to answer a challenge, a deposit will be forfeit and will need to be replaced if the participant wishes to keep participating in the beacon protocol.

B. Output Withholding

We additionally propose an extension wherein the beacon output is withheld, but a proof is published that the recipient can recover the beacon output (proof of receipt). To avoid the proof itself from being used as a random beacon, every participant must be able to manipulate the proof.

To accomplish this, we start with a standard beacon setup. The recipient chooses some b_{ij}s for each participant i and a small number of js, and publishes $g^{b_{ij}}$ for each i, j. For participant i, the recipient secretly shares each b_{ij} for each j. During beacon computation, each participant chooses a linear combination of js:

$$l_i = \sum_j \alpha_{ij} b_{ik}$$

and computes

$$F_{SK_i} = r^{a_i l_i}$$

The participant publishes $(\alpha_{i1}, \alpha_{i2}, ...)$ and $g^{a_i l_i}$. Since the recipient knows the b_is, they can recover l and compute ${l_i}^{-1}$ to recover r^{a_i}. A third party can verify the recipient's ability to recover the value by computing

$$g^{l_i} = g^{\sum_j \alpha_{ij} b_{ik}} = \prod_j g^{\alpha_{ij} b_{ij}} = \prod_j \left(g^{b_{ij}}\right)^{\alpha_{ij}}$$

and verifying both $(g, g^{a_i}, g^{l_i}, g^{a_i l_i})$ and $(g, g^{a_i l_i}, r, r^{a_i l_i})$ form valid DDH tuples. A participant can manipulate the public output by choosing a different combination of αs, while still leaving both the value of the beacon output unchanged.

Reconstruction of a single system proof is not possible, but it is also not necessary since each share is individually verifiable.

C. Output Withholding Security

In this section, we discuss the security of output withholding, introduce and modify some security properties, and demonstrate that those properties hold. For simplicity, we discuss the security in a non-distributed setting, but the results from section VI-C can be used to generalize these results to a distributed setting.

Strictly speaking, the relaxed soundness property does not hold for this scheme. This is because any combination of α_js has a corresponding proof. Instead, we let every combination of $(\alpha_1, \alpha_2, ...)$ be a distinct VUF. Clearly, relaxed completeness holds for our scheme. We can see weak unpredictability holds after output withholding by observing that each linear combination is equivalent to a distinct key refreshing.

We need to modify our notion of **relaxed soundness**, instead showing that each x corresponds to a single $F_{SK}(x)$, regardless of $\vec{\alpha}$. We need a property, **output withholding**, which states the withheld output should not be recoverable from the proof or public output, except with negligible probability. Finally, we need a property that guarantees the participant is able to affect the output, which we will call **engineerability**.

More formally, a **output withholding weak VUF** has the following functions:

1) $\begin{pmatrix} SK, PK, S\vec{K}' = \{SK'_0 ... SK'_m\}, \\ P\vec{K}' = \{PK'_0 ... PK'_m\} \end{pmatrix} \leftarrow Gen(1^k)$
2) $v' \leftarrow F'_{S\vec{K}'}(x, \vec{\alpha})$
3) $\pi' \leftarrow \Pi'_{S\vec{K}'}(x, \vec{\alpha})$
4) $v, \pi \leftarrow \text{Recover}(S\vec{K}', x, \vec{\alpha}, v', \pi')$
5) $V(PK, x, v, \pi)$
6) $V'(PK, P\vec{K}', x, \vec{\alpha}, v', \pi')$

$\vec{\alpha}$ is an array of length m, with elements of size $c(k)$. The values are chosen in any manner when F' is called.

With the following properties:

1) **Relaxed Completeness**: If $\left(SK, PK, S\vec{K}', P\vec{K}'\right) = \text{Gen}(1^k)$ then for all $x, \vec{\alpha}, v' = F'_{S\vec{K}'}(x, \vec{\alpha})$, it holds that $V(PK, PK, x, \text{Recover}(S\vec{K}', x, \vec{\alpha}, v', \pi')) = \text{True}$ and $V'(PK, P\vec{K}', x, \vec{\alpha}, v', \pi') = \text{True}$, expect with probability $negl(k)$
2) **Relaxed Soundness**: If $\left(SK, PK, S\vec{K}', P\vec{K}'\right) = \text{Gen}(1^k)$, then there exist no values $x, \vec{\alpha_1}, \vec{\alpha_2}$

such that $\text{Recover}(S\vec{K}', x, \vec{\alpha_1}, F'_{S\vec{K}'}(x, \vec{\alpha_1})) \neq \text{Recover}(S\vec{K}', x, \vec{\alpha_2}, F'_{S\vec{K}'}(x, \vec{\alpha_2}))$, expect with probability $negl(k)$

3) **output withholding**: For all PPT algorithms A given oracle access to F, F'

$$Pr\left[w = v \;\middle|\; \begin{array}{l} SK, PK, S\vec{K}', P\vec{K}' \leftarrow Gen(1^k) \\ x \leftarrow \{0,1\}^{a(k)}, \vec{\alpha} \leftarrow \{0,1\}^{c(k)\times m} \\ v \leftarrow Recover_{S\vec{K}'}\left(F'_{SK,S\vec{K}'}(x, \vec{\alpha})\right) \\ w \leftarrow F \end{array}\right]$$

4) **Engineerability**: Let $p \in [0,1]$ be a probability and $\mathcal{R} : \{0,1\}^{b(k)} \to \{0,1\}$ be a predicate for which $p \cdot b(k)$ inputs map to $\{0\}$ and $(1-p) \cdot b(k)$ inputs map to $\{1\}$. There exists a PPT algorithm B such that for all PPT algorithms A the following experiment succeeds with probability at most $\min(p, 1-p)^{poly(k)} + negl(k)$

$$(SK, PK, S\vec{K}', P\vec{K}') \leftarrow Gen(1^k)$$
$$(x, \mathcal{R}, b) \leftarrow A(SK, PK, S\vec{K}', P\vec{K}')$$
$$\vec{\alpha} = B^{\mathcal{R}(\cdot)}(SK, PK, S\vec{K}', P\vec{K}', x, b)$$
$$v' = F'_{SK,S\vec{K}'}(x, \vec{\alpha})$$
$$b' = \mathcal{R}(v')$$

The experiment succeeds if $b \neq b'$

5) **Weak pseudorandomness**: Let $rand(S)$ output a random element $s \in S$. For any PPT algorithm $A = (A_1, A_2)$ given an oracle to f, Π who does not query $(x, *)$, the following experiment fails with probability at most $\frac{1}{2} + negl(k)$

$$(SK, PK, S\vec{K}', P\vec{K}') \leftarrow Gen(1^k)$$
$$x \leftarrow rand\left(\{0,1\}^{a(k)}\right)$$
$$(\vec{\alpha}, st) \leftarrow A_1^{F'_{SK,S\vec{K}'}(\cdot,\cdot), \Pi'_{SK,S\vec{K}'}(\cdot,\cdot)}(x)$$
$$y_0 \leftarrow F'_{SK,S\vec{K}'}(x, \vec{\alpha}); y_1 \leftarrow rand\left(\{0,1\}^{b(k)}\right)$$
$$b \leftarrow rand\left(\{0,1\}\right)$$
$$b' \leftarrow A_2^{F'_{SK,S\vec{K}'}(\cdot,\cdot), \Pi'_{SK,S\vec{K}'}}(x, y_b, st)$$

The experiment succeeds if $b = b'$

Before moving onto the proofs, we want to motivate the **engineerability** property. This property is critical for ensuring no adversary can construct a "pirate" random beacon from the value-withheld beacon. That is, we desire that any randomness extracted from the public portion of a value-withheld IVRB is no better than the Bonneau *et al.* random beacon it was constructed from, with the idea that any random beacon construction from the public portion of our value-withheld IVRB may as well be built on the Bonneau *et al.* random beacon.

We note that in the definition for **engineerability**, the visibility of $\mathcal{R}$ to B might appear to be a limitation, since a real adversary might hide $\mathcal{R}$ but make a commitment to it. However, any such adversary could choose a participant to secretly reveal $\mathcal{R}$ to, and therefore choose a favorable $\vec{\alpha}$. In the distributed setting, the adversary would need only collude with a single participant to manipulate the output. This violates **manipulation resistance**, giving a weaker random beacon than our complete beacon, which requires collusion with at least t participants to manipulate the output. This means any random beacon constructed from a predicate on the public portion of our value withheld beacon can be manipulated for minimal cost. Therefore, the Bonneau *et al.* random beacon is a at least as secure a choice as a random beacon constructed from a predicate on the public portion of our value-withholding IVRB.

Correctness is immediate from the construction. **Soundness** and **pseudorandomness** are immediate since they reduce to the corresponding wVUF properties.

Now we sketch the proof of **engineerability**. Informally, let B be a random guess and check algorithm. More concretely, B is the following algorithm:

loop $poly(k)$ iterations
  $\vec{\alpha} \leftarrow$ random $\left(\{0,1\}^{c(k)\times m}\right)$
  $v' = F'_{SK,S\vec{K}'}(x, \vec{\alpha})$
  $b' = \mathcal{R}(v')$
  if $b = b'$ **then**
    return $\vec{\alpha}$
  end if
end loop
return $\perp$

To show **engineerability**, we will show that for any A, B terminates in polynomial time with high probability. For the sake of argument, suppose we used a random oracle H in place of F'. Because the output of $H(x, \vec{\alpha})$ is random, then b' is 0 with probability p and 1 with probability $(1-p)$. Observe that for a fixed $\vec{\alpha}$, the functions (Gen, F', V') constitute a wVUF. Therefore, if the adversary has not evaluated $F'_{SK,S\vec{K}'}(x, \vec{\alpha})$, the result is indistinguishable from H. Because the adversary runs in $poly(k)$ time and there are at least 2^k values for alpha, the probability the adversary has observed a particular output is $\frac{poly(k)}{2^k}$, so in each loop of B, $b = b'$ with probability at least $\min(p, 1-p) - \frac{poly(k)}{2^k}$. On a success, B outputs $\vec{\alpha}$. On a failure, B repeats this process. After $poly(k)$ attempts, B fails each attempt with probability $\left(\min(p, 1-p) - \frac{poly(k)}{2^k}\right)^{poly(k)}$. After expanding this expression, there will be one term that is $\min(p, 1-p)^{poly(k)}$, and $poly(k)$ terms that are of the form $\pm\left(\frac{poly(k)}{2^k}\right)^{poly(k)} = \pm negl(k)$, giving us

$$Pr[b \neq b']$$
$$= \min(p, 1-p)^{poly(k)} + poly(k) \cdot negl(k) - poly(k) \cdot negl(k)$$
$$= \min(p, 1-p)^{poly(k)} + negl(k) - negl(k)$$
$$\leq \min(p, 1-p)^{poly(k)} + negl(k)$$

hence the claim.

Next, we sketch the proof of **output withholding**. To show this, we first define 3 PPT algorithms:

$$\mathcal{A}_1\left(g, g^a, \left\{g^{b_1}, ..., g^{b_m}\right\}, \{\alpha_1 ... \alpha_m\}, g^{al}, r, r^l, r^{al}\right) = r^a$$

where $l = \sum_{i=1}^{m} \alpha_i b_i$

$$\mathcal{A}_2\left(g, g^a, g^b, g^{ab}, r, r^{ab}\right) = r^a$$

$$\mathcal{A}_3\left(g, g^a, g^b, g^{ab}, g^c, g^{abc}\right) = g^{ac}$$

Observe that algorithm $\mathcal{A}_1$ is the algorithm that models the adversary recovering the output from the proof.

For any chosen set of α_is and any set $\{b_1, ...b_{m-1}\}$, $\mathcal{A}_2$ can be efficiently computed with $\mathcal{A}_1$. This is done by choosing $\{\alpha_1, ..., \alpha_m\}$ and $\{b_1, ..., b_{m-1}\}$, and then computing g^{b_m}.

If we let $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a bilinear mapping, then $r \in \mathbb{G}_1$, $g \in \mathbb{G}_2$, then $\mathbb{G}_1$ and $\mathbb{G}_2$ are of the same order, so a mapping from from $\mathbb{G}_1$ to $\mathbb{G}_2$ exists, and $\mathcal{A}_3$ can be computed with $\mathcal{A}_2$.[5]

Therefore, showing that $\mathcal{A}_3$ cannot exist suffices to show $\mathcal{A}_1$ does not exist under the above assumptions. To show this, we need only notice this algorithm would allow an adversary to solve the Generalized Computational Diffie-Hellman assumption, as defined by Dodis. [13] The definition of the assumption, which follows Dodis[13] closely, is as follows:

Let $\mathbb{G}$ be a cyclic group of prime order q with generator g. Let L be some integer, and $a_1 ... a_L$ be elements of $\mathbb{Z}_q$. Let $[L] = \{1...L\}$, and given $I \subseteq [L]$, let $a_I = \prod_{i \in I} a_i \text{mod} q$. Let $G(I) = g^{a_I}$. We allow the adversary oracle access to G and we let $I_1 ... I_t$ be the subsets the adversary accessed with G. We define a predicate $\mathcal{R}(J, I_1, ..., I_t)$ which decides if the oracle accesses are "legal". We call any predicate is false if $J \in \{I_1 ... I_T\}$ "non-trivial". The generalized computational Diffie-Hellman assumption is then as follows:

We say a group $\mathbb{G}$ follows the **generalized computational Diffie-Hellman assumption** of order $L = L(k)$ relative to a non-trivial predicate $\mathcal{R}$ if for any PPT adversary $\mathcal{B} = \{B\}$ who has called oracle G on subsets $I_1 ... I_t$ satisfying $\mathcal{R}(J, I_1, ..., I_t)$, then:

$$Pr\left[y = y' \middle| \begin{array}{r} (\mathbb{G}, q, g) \leftarrow \text{Setup}(1^k); \\ (a_1, ..., a_L) \leftarrow \mathbb{Z}_{\text{ll}}; \\ (J, y') \leftarrow B^{G(\cdot)}(\mathbb{G}, q); \\ y = G(J); \end{array}\right] \leq negl(k)$$

For a weaker and more plausible assumption, we choose $L = 3$ and define $\mathcal{R}$ such that it is true if and only if $J = \{a, c\}$ and $\{I_1, ..., I_t\} = \{\{a\}, \{b\}, \{c\}, \{ab\}, \{abc\}\}$. Clearly, we can use $\mathcal{A}_3$ to solve this problem, which is a contradiction. Hence, no PPT algorithm satisfying $\mathcal{A}_1$ exists.

[5]Note that in practice such mappings are not always known, nor are they always efficient[26]. Regardless, we may assume the adversary has efficient oracle access to such a mapping, since any adversary $\mathcal{B}$ that has such a mapping can simulate an adversary $\mathcal{B}'$ who does not have such a mapping, so if no $\mathcal{B}$ exists, then no $\mathcal{B}'$ exists.

IX. Conclusions

We have proposed a model for producing public-randomness that is readily configurable to handle a broad range of threats. Our protocol guarantees unbiasability, unpredictability, verifiability, and the ability to audit most steps. Our protocol is designed with an incentive structure that, when correctly tuned, provides no economic incentive for a participant to be dishonest and ensures bribery is too expensive for adversaries. Unlike many previous protocols, there is never a single actor who can affect the outcome. We protect against biasing and prediction attacks with a protocol that depends on a combination of economic and computational security parameters.

Our work has some limitations that need to be considered carefully before adoption. Paying the participant fees, transaction fees, and other costs could be very expensive. Our fee structure is also linear in cost relative to security – a super-linear incentive structure would better scale to high powered adversaries. In our analysis, we have primarily considered Ethereum as a candidate cryptocurrency for implementation, but as we discussed in section VII-B, Ethereum has limitations that hinder our ability to fully implement our protocol. Other cryptocurrencies or future version of Ethereum may prove more effective.

Future work could further refine economic incentives to provide more specific recommendations for each value n, t, k, d, f, or explore the possibility for stronger incentives at a lower cost. Future work could also adapt our method for use with a standard VRF, rather than a weak VRF, for improved security. One limitation we have is that the proof of a withheld value can be shared with third parties, so once a withheld value is shared with a dishonest party, it can be freely distributed. A mechanism to verify a withheld value in zero-knowledge would be valuable to ensure withheld beacon outputs cannot be freely distributed, but simply transforming to a space where DDH is hard would lose the proof of receipt.

Acknowledgment

Acknowledgements removed for review.

References

[1] R. P. et al, "Nist randomness beacon," 2011.

[2] J. Clark and U. Hengartner, "On the use of financial data as a random beacon." *EVT/WOTE*, vol. 89, 2010.

[3] J. A. Halderman and B. Waters, "Harvesting verifiable challenges from oblivious online sources," in *Proceedings of the 14th ACM Conference on Computer and Communications Security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 330–341. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315287

[4] J. Bonneau, J. Clark, and S. Goldfeder, "On bitcoin as a public randomness source." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1015, 2015.

[5] C. Pierrot and B. Wesolowski, "Malleability of the blockchain's entropy," *Cryptography and Communications*, vol. 10, no. 1, pp. 211–233, 2018.

[6] I. Bentov, A. Gabizon, and D. Zuckerman, "Bitcoin beacon," *arXiv preprint arXiv:1605.04559*, 2016.

[7] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual International Cryptology Conference*. Springer, 2018, pp. 757–788.

[8] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.
[9] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *Security and Privacy (SP), 2017 IEEE Symposium on*. Ieee, 2017, pp. 444–460.
[10] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the weil pairing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 514–532.
[11] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," *arXiv preprint arXiv:1805.04548*, 2018.
[12] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 2017, pp. 51–68.
[13] Y. Dodis, "Efficient construction of (distributed) verifiable random functions," in *Public Key Cryptography*, vol. 2567. Springer, 2003, pp. 1–17.
[14] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE, 1999, pp. 120–130.
[15] A. Lysyanskaya, "Unique signatures and verifiable random functions from the dh-ddh separation," in *Annual International Cryptology Conference*. Springer, 2002, pp. 597–612.
[16] Z. Brakerski, S. Goldwasser, G. N. Rothblum, and V. Vaikuntanathan, "Weak verifiable random functions." in *TCC*, vol. 5444. Springer, 2009, pp. 558–576.
[17] A. De Caro and V. Iovino, "jpbc: Java pairing based cryptography," in *Proceedings of the 16th IEEE Symposium on Computers and Communications, ISCC 2011*. Kerkyra, Corfu, Greece, June 28 - July 1: IEEE, 2011, pp. 850–855. [Online]. Available: \url{http://gas.dia.unisa.it/projects/jpbc/}
[18] B. Lynn, "On the implementation of pairing-based cryptosystems," Ph.D. dissertation, Stanford University Stanford, California, 2007.
[19] Google. Google cloud platform. [Online]. Available: https://cloud.google.com/
[20] P.-A. Fouque and M. Tibouchi, "Indifferentiable hashing to barreto–naehrig curves," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2012, pp. 1–17.
[21] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1–32, 2014.
[22] P. S. Barreto and M. Naehrig, "Pairing-friendly elliptic curves of prime order," in *International Workshop on Selected Areas in Cryptography*. Springer, 2005, pp. 319–331.
[23] Ethereum network status. [Online]. Available: https://ethstats.net/
[24] Cryps.info usd to gwei. [Online]. Available: https://www.cryps.info/en/USD_to_Gwei/
[25] R. Cramer, I. Damgård, and P. MacKenzie, "Efficient zero-knowledge proofs of knowledge without intractability assumptions," in *International Workshop on Public Key Cryptography*. Springer, 2000, pp. 354–372.
[26] S. D. Galbraith, K. G. Paterson, and N. P. Smart, "Pairings for cryptographers," *Discrete Applied Mathematics*, vol. 156, no. 16, pp. 3113–3121, 2008.

APPENDIX

A. Notation Guide

In order to mitigate confusion regarding the notation, we have assembled a list of notation in our paper. This list is not comprehensive, but we have included the most important symbols and their meanings. In most cases, if a symbol cannot be found in this list, it was only used in one section. Note that a handful of symbols have different meanings in different contexts, e.g. V is both the verifier algorithm in the context of VRFs and the utility to the adversary in the context of the incentive structure.

1) Security parameters:

Symbol	Meaning
k	The computational security parameter
V	The monetary security parameter
$poly(k)$	Any function which can be expressed as less than or equal to a positive polynomial of k
$negl(k)$	Any function which is negligibly small in k, specifically for every positive polynomial $poly(k)$: $$negl(k) < \frac{1}{poly(k)}$$

2) VRFs, general:

Symbol	Meaning	Source
Gen	Key generation/setup algorithm	II
Π	Prover algorithm: publishes a proof	II
V	Verifier Algorithm: Verifies a proof	II
f	Evaluator algorithm: Publishes the output of the VRF	II
SK	The secret key	II
PK	The public key	II
x	The (public) function input	II
π	The proof: $\pi \leftarrow \Pi_{SK}(x)$	II
v	The VRF output: $v \leftarrow f_{SK}(x)$	II

3) VRFs and bilinear mappings:

Symbol	Meaning	Source
$\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$	Groups that together have an associated bilinear mapping	IV-A
e	The bilinear mapping: $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ such that $e(g^a, g^b) = ab \cdot e(g, g)$	IV-A
g	Generator of $\mathbb{G}_1$	IV-A
r	Generator of $\mathbb{G}_2$, depends on x as $r =$ hash(x)	IV-A
a	Secret key; i.e. $SK = a$	IV-A
$\mathcal{S}_i$	In a distributed VRF, for arbitrary symbol $\mathcal{S}$: $\mathcal{S}$ with regards to participant i	IV-A

4) Key Refreshing:

Symbol	Meaning	Source
c	The submitting party for a key refresh	IV-B
C	The number of key refresh submitting parties over a given interval	IV-B
b_c	The secret change c is making to the secret key a	IV-B
$p_c(x)$	The secret polynomial c will use to distribute b_c in parts	IV-B

5) Beacon output withholding:

Symbol	Meaning	Source
b_{ij}	jth shared secret between participant i and the designated recipient	VIII-B
α_{ij}	The public weighting of the jth secret for participant i. Chosen by participant i.	VIII-B
l_i	The linear combination that obscures the output	VIII-B

B. Sample Contract Pseudocode

Here, we present pseudocode for the smart contract in section V-B. The smart contract should follow the following procedure for submitting an output:

function SUBMIT(i, π)
  Check p_i has paid the deposit
  PK $\leftarrow$ Public key of p_i
  $b \leftarrow$ block number of designated block
  $y \leftarrow$ hash of block b
  $\epsilon \leftarrow$ number of blocks before expiration
  $r \in \mathrm{G}_1$, chosen with y
  if current block $\geq b + \epsilon$ **then**
    Handle expired deposit
  else if submitter = $p_i \wedge$ Verify(PK, r, π) **then**
    Record π_1
    Schedule payment to p_i
  end if
end function

The smart contract should follow the following procedure for claiming a bounty:

function CLAIMBOUNTY(i, x, π)
  Check p_i has paid the deposit
  PK $\leftarrow$ Public key of p_i
  $s \leftarrow$ block number of scheduled payout
  $y \leftarrow$ hash of x
  $r \in \mathrm{G}_1$, chosen with y
  if currentBlock $< s \wedge$ Output was submitted $\wedge$ Verify(PK, r, π) $\wedge$ r has not been observed before **then**
    Record π_1
    Redirect payment to submitter
  end if
end function

Optionally, the key changes can be tracked as well.

The leader of a public key change will call the following procedure on the smart contract to declare intention to propose a key change:

function PROPOSEPKCHANGE(ΔPK)
  Choose id as a nonce
  Save $\Delta PK_{id} = \Delta PK$ **return** id
end function

When a participant receives a secret key change, they then submit the change in the public key through the smart contract, as follows:

function SUBMITPKCHANGE($i, id, \Delta PK$)
  Let id be a nonce
  Save $\Delta PK_{i,id} = \Delta PK$
  if Length of $\Delta PK_{*,id} = n$ **then**
    Interpolate $\Delta PK'$ from $\Delta PK_{*,id}$
    if $\Delta PK = \Delta PK'$ **then**
      Accept PK Change id
    else
      Reject PK Change id
    end if
  end if
end function