Bicorn: An Optimistically Efficient Distributed Randomness Beacon

Kevin Choi[1][0009−0006−6890−7313], Arasu Arun[1][0009−0000−1009−2365], Nirvan Tyagi[2][0000−0002−7671−5681], and Joseph Bonneau[1,3][0000−0002−6349−0145]

[1] New York University
kc2296@nyu.edu
[2] Cornell University
[3] a16z crypto research

Abstract. We introduce Bicorn, an optimistically efficient distributed randomness protocol with strong robustness under a dishonest majority. Bicorn is a "commit-reveal-recover" protocol. Each participant commits to a random value, which are combined to produce a random output. If any participants fail to open their commitment, recovery is possible via a single time-lock puzzle which can be solved by any party. In the optimistic case, Bicorn is a simple and efficient two-round protocol with no time-lock puzzle. In either case, Bicorn supports open, flexible participation, requires only a public bulletin board and no group-specific setup or PKI, and is guaranteed to produce random output assuming any single participant is honest. All communication and computation costs are (at most) linear in the number of participants with low concrete overhead.

1 Introduction

Distributed randomness beacons (DRBs) aim to enable a group of n participants to jointly compute a random output (which we denote Ω) such that no participant or coalition of participants can predict or influence the outcome. Among many other applications, they are useful for cryptographically verifiable lotteries or leader election in efficient distributed consensus protocols.

A classic approach is *commit-reveal* [9]. First, all participants publish a commitment $c_i = \mathsf{Commit}(r_i)$ to a random value r_i. Next, participants reveal their r_i values and the result is $\Omega = \mathsf{Combine}(r_1, \ldots, r_n)$ for some suitable combination function (such as exclusive-or or a cryptographic hash). Commit-reveal protocols are simple, efficient, and secure as long as one participant chooses a random r_i value—assuming all participants open their commitments. However, the output can be biased by the last participant to open their commitment (a so-called *last-revealer attack*), as that participant will know all other r_i values and can compute Ω early. If the last revealer doesn't like the impending value of Ω, they can refuse to open, forcing the protocol to abort. Even if the last revealer is removed from subsequent protocol runs, this enables one bit of bias.

Related work. Several approaches exist to avoid last-revealer attacks. Commit-reveal-punish protocols impose a financial penalty on any participant who fails

to open their commitment. This penalty can be automatically enforced using modern cryptocurrencies [2,32], but this requires locking up capital and security relies on economic assumptions about the value of manipulation to the attacker.

Other protocols relax the security model of commit-reveal and assume an honest majority of participants. Many constructions enable a majority of participants to recover the input of a malicious minority of participants [7,8,19,20,24,26,27,34,35,37], using cryptographic tools such as publicly verifiable secret sharing (PVSS). Typically, these constructions can tolerate some threshold t of malicious participants failing to complete the protocol, with the trade-off that any coalition of $t+1$ participants can (secretly) learn the impending output early and potentially bias the protocol, leading to a requirement that $t < \frac{n}{2}$ (honest majority). These protocols are also often quite complex, with communication and computation costs superlinear in n. Another approach is to rely on threshold cryptography for participants to jointly compute a cryptographic function which produces Ω, such as threshold signatures in Dfinity [18], threshold encryption [22], or threshold inversion in RSA groups [3,4]. The drand DRB [1], which uses a chain of threshold BLS signatures, is now deployed publicly with a group of 16 participating nodes producing a new random output every 30 seconds.

A very different approach to constructing DRBs uses time-based cryptography, specifically using *delay functions* to prevent manipulation. The simplest example is Unicorn [28], a one-round protocol in which participants directly publish (within a fixed time window) a random input r_i. The result is computed as $\Omega = \mathsf{Delay}(\mathsf{Combine}(r_1, \ldots, r_n))$. By assumption, a party cannot compute the Delay function before the deadline to publish their contribution r_i and therefore cannot predict Ω or choose r_i in such a way as to influence it. This protocol retains the strong $n-1$ (dishonest majority) security model of commit-reveal, but with no last-revealer attacks. It is also simple and, using modern verifiable delay functions[4] (VDFs) [10], the result can be efficiently verified. The downside is that a delay function must be computed for every run of the protocol.

Our approach. We introduce the Bicorn family of DRB protocols, which retain the advantages of Unicorn while enabling efficient computation of the result (with no delay) if all participants act honestly. The general structure is:

- Each of n participants chooses a random value r_i and publishes $c_i = \mathsf{TCom}(r_i)$ using a timed commitment scheme [14] TCom before some deadline T_1.
- In the optimistic case, every participant opens their commitment by publishing r_i. The DRB output is $\Omega = \mathsf{Combine}(r_1, \ldots, r_n)$. In this case, the protocol is equivalent to a classic commit-reveal protocol.
- If any participant does not publish their r_i value, it can be recovered by computing $r_i = \mathsf{ForceOpen}(c_i)$, a slow function requiring t steps of sequential work which cannot be evaluated quickly enough for a malicious coalition of participants to learn honest participants' committed values early. The result

[4] The original Unicorn proposal used modular square roots in a prime-order group. We consider using a modern VDF instead.

Ω is the same as in the optimistic case, even if all participants don't reveal their committed values.

This protocol structure was used in a recent proposal by Thyagarajan et al. [38]. They observe that by using a homomorphic commitment scheme, the commitments can be combined and only a single forced opening is required, instead of opening every withholding participant's commitment separately. Asymptotically, their protocols require linear ($O(n)$) communication and computation costs when run with n participants.

However, Thyagarajan et al. use a general-purpose CCA-secure timed commitment scheme suitable for committing to arbitrary messages, which introduces significant practical complexity and overhead. Our key insight is that constructing a DRB does not require a general-purpose commitment scheme; it is sufficient to use a special restricted commitment scheme which only enables committing to a pseudorandom message. As a result, our protocols are considerably simpler and offer much better concrete performance.

Contributions. We introduce the Bicorn family of protocols, which comes in three flavors with slightly different security proofs and practical implications:

- Bicorn-ZK, which requires each participant to publish a zero-knowledge proof of knowledge of exponent. This imposes the highest practical overhead but offers the simplest security proof.
- Bicorn-PC, in which participants "pre-commit" their contribution before the protocol. This is the simplest version, though it adds an extra communication round (which can be amortized over multiple runs).
- Bicorn-RX, which utilizes a randomized exponent to prevent manipulation attacks. This is the most efficient version in practice, though the security proof relies on stronger assumptions.

In Section 3, we prove security of our constructions by reducing to the RSW assumption [33] in the algebraic group model (AGM) [25], except for Bicorn-ZK where we assume a zero-knowledge proof of knowledge of exponent (ZK-PoKE) exists. The Bicorn-RX variant assumes a random oracle. In Section 6, we report on concrete implementations of these protocols in Ethereum, showing that our constructions are practical and incur 3–8× increase in per-user cost compared to commit-reveal (but with no manipulation due to aborts) and 5–7× compared to Unicorn (but with no delay function required in the optimistic case).

2 Overview

2.1 Protocol Outline

We specify all three of our protocol variants in Protocol 1. Our protocols are initialized via a security parameter λ and a delay parameter t, and work over a *group of unknown order*, which we denote $\mathbb{G}$ (see preliminaries in Section 3). In addition to the group $\mathbb{G}$, the public parameters include a pair (g, h), where g is a generator of the group and $h = g^{2^t}$. If desired, a Wesolowski [41] or Pietrzak [30]

proof of exponentiation can enable efficient verification that h was computed correctly. Note that this setup only needs to be run once ever (for a specific delay parameter t) and can be used repeatedly (and concurrently) by separate protocol instances; the number of participants does not need to be known and may dynamically change over time.

The common structure of Bicorn protocols is:

- Each of n participants chooses a random value α_i and publishes $c_i = g^{\alpha_i}$. The value c_i can be viewed as the input to a VDF whose output is $(c_i)^{2^t}$, with α_i serving as a trapdoor to quickly compute $(c_i)^{2^t} = (g^{\alpha_i})^{2^t} = (g^{2^t})^{\alpha_i} = h^{\alpha_i}$. Without knowledge of α_i this value is slow to compute. Depending on the security assumptions made, α_i can be sampled from different distributions. We abstract this choice by parameterizing by a uniform distribution $\mathcal{B}$ from which α_i is sampled.
- Participants "open" their commitment c_i by revealing a value $\tilde{\alpha}_i$. It can be quickly verified that $\tilde{\alpha}_i$ is the correct α_i by verifying that $c_i = g^{\tilde{\alpha}_i}$.
- *Optimistic case:* Given all correct α_i values, the DRB output Ω is the product $\Omega = \prod_{i\in[n]} h^{\alpha_i}$, which is unpredictable as long as at least one of the α_i values was randomly chosen and is easy to compute if all α_i values are correctly revealed.
- *Pessimistic case:* If any participant withholds α_i (or chose c_i without knowledge of the corresponding α_i), then the missing value h^{α_i} can be recovered (slowly) by computing $h^{\alpha_i} = (c_i)^{2^t}$, equivalent to evaluating a VDF. If multiple participants withhold α_i, naively one must compute each missing value h^{α_i} individually. A more efficient approach (which works even if all participants withhold α_i) is to first combine each participant's contribution into the value $\omega = \prod_{i\in[n]} c_i$. The output can then be computed via a single slow computation as $\Omega = \omega^{2^t}$, which is identical to the output $\Omega = \prod_{i\in[n]} h^{\tilde{\alpha}_i}$ computed in the optimistic case.

By itself this protocol is insecure, because a malicious participant need not choose c_i by choosing a value α_i and computing g^{α_i}. An adversary j who has precomputed a desired output $\Omega_* = (\omega_*)^{2^t}$ and is able to publish last can compute a malicious contribution:

$$c_j = \omega_* \cdot \left(\prod_{i\in[n], i\neq j} c_i \right)^{-1} \tag{1}$$

This will cancel out every other participant's contribution and force the output value Ω_*. There are three ways to prevent this attack, each leading to a protocol variant with slightly different properties, which we will present in the following subsections. We present the protocols combined for comparison in Protocol 1.

$\mathsf{Setup}(\lambda, t)$		(run once for all protocol runs)
1. Run $(\mathbb{G}, g, A, B) \xleftarrow{\$} \mathsf{GGen}(\lambda)$ to generate a group of unknown order 2. Compute $h \leftarrow g^{2^t}$, optionally with $\pi_h = \mathsf{PoE}(g, h, 2^t)$ 3. Output $(\mathbb{G}, g, h, \pi_h, A, B)$		
$\mathsf{Prepare}()$		(run by each participant i)
$\alpha_i \xleftarrow{\$} \mathcal{B}$ $c_i \leftarrow g^{\alpha_i}$ $\pi_i \leftarrow \mathsf{ZK\text{-}PoKE}(g, c_i, \alpha_i)$	$\alpha_i \xleftarrow{\$} \mathcal{B}$ $c_i \leftarrow g^{\alpha_i}$ $d_i \leftarrow H(c_i)$	$\alpha_i \xleftarrow{\$} \mathcal{B}$ $c_i \leftarrow g^{\alpha_i}$
$\mathsf{Precommit}(d_i)$		(run by each participant i)
–	Publish d_i	–
..... *deadline* T_0 ...		
$\mathsf{Commit}(c_i, \pi_i)$		(run by each participant i)
Publish c_i, π_i	Publish c_i	Publish c_i
..... *deadline* T_1 ...		
$\mathsf{Reveal}(\alpha_i)$		(run by each participant i)
Publish α_i	Publish α_i	Publish α_i
$\mathsf{Finalize}(\{(\tilde{\alpha}_i, c_i, d_i, \pi_i)\}_{i=1}^n)$		(optimistic case, once per protocol run)
1. $\forall_j$ Verify proof π_j – else: remove user j 2. $\forall_j$ Verify $c_j = g^{\tilde{\alpha}_j}$ – else: go to $\mathsf{Recover}$ $\Omega = \prod_{i\in[n]} h^{\tilde{\alpha}_i}$	1. $\forall_j$ Verify $d_j = H(c_j)$ – else: remove user j 2. $\forall_j$ Verify $c_j = g^{\tilde{\alpha}_j}$ – else: go to $\mathsf{Recover}$ $\Omega = \prod_{i\in[n]} h^{\tilde{\alpha}_i}$	1. $b_* \leftarrow H(c_1 \| \ldots \| c_n)$ 2. $\forall_j$ Verify $c_j = g^{\tilde{\alpha}_j}$ – else: go to $\mathsf{Recover}$ $\Omega = \prod_{i\in[n]} \left(h^{H(c_i \| b_*)}\right)^{\tilde{\alpha}_i}$
$\mathsf{Recover}(\{(c_i, d_i, \pi_i)\}_{i=1}^n)$		(pessimistic case, once per protocol run)
$\Omega = \left(\prod_{i\in[n]} c_i\right)^{2^t}$	$\Omega = \left(\prod_{i\in[n]} c_i\right)^{2^t}$	$\Omega = \left(\prod_{i\in[n]} c_i^{H(c_i \| b_*)}\right)^{2^t}$

Protocol 1: All Bicorn protocol variants: Bicorn-ZK (left column), Bicorn-PC (center column), and Bicorn-RX (right column).

2.2 Bicorn-ZK: Using Zero-Knowledge Proofs

The conceptually simplest fix is for each user to publish, along with their commitment c_i, a zero-knowledge proof-of-knowledge $\pi_i = \mathsf{ZK\text{-}PoKE}(g, c_i, \alpha_i)$ of the discrete logarithm of c_i to the base g_i (i.e. α_i). This version (Bicorn-ZK) is specified in Protocol 1 (left). This removes the attack above, as an adversary who computes c_j via Equation 1 will not know the discrete log of c_j to the base g. Such proofs can be done in groups of unknown order particularly efficiently in this case. The use of a fixed base g enables the simpler $\mathsf{ZKPoKRep}$ protocol of Boneh et al. [11] (possibly in combination with their proof aggregation PoKCR protocol).

Participants publishing invalid proofs are removed, and the protocol can continue and still produce output. Attempting to participate with an invalid proof is equivalent to not participating at all (though participants who do so might need to be blocked or penalized financially to deter denial-of-service attacks).

It might be tempting to optimize the protocol by not verifying each proof π_i in the optimistic case, instead checking directly that $c_i = g^{\tilde{\alpha}_i}$ using the revealed value $\tilde{\alpha}_i$. However, this would introduce a subtle attack: a malicious participant could publish a correctly generated $(c_i, \tilde{\alpha}_i)$ pair but with an invalid proof $\tilde{\pi}_i$. Next, after all other participants have revealed their α values, the attacker can compute the impending result Ω with their own contribution included, as well as the alternative Ω' if it is removed. They could then choose which output is produced, introducing one bit of bias into the protocol: by publishing $\tilde{\alpha}_i$, they will remain in the protocol (as $\tilde{\pi}_i$ is not checked) and Ω will result, whereas by withholding $\tilde{\alpha}_i$ they will force the pessimistic case, in which they will be removed on account of the faulty $\tilde{\pi}_i$ and Ω' will result. Thus, it is important to verify every participant's proof π_i in both cases to prevent this attack.

2.3 Bicorn-PC: Using Precommitment

Another approach to prevent manipulation is to add an initial precommitment round where participants publish $d_i = H(c_i)$, preventing them from choosing c_i in reaction to what others have chosen. This version (Bicorn-PC) is specified in Protocol 1 (center). Participants can decline to reveal their committed c_i, in which case they are removed and the protocol can continue safely. Because participants will not have time to compute the impending output before choosing whether to reveal, this does not introduce any opportunity for manipulation.

Note that the precommitted values d_i can be published at any point prior to T_0 (the point at which participants start revealing their actual commitment c_i). If the protocol is run iteratively, it is possible for participants to publish any number of precommitments d_i in advance (or a single commitment to a set of d_i values using a set commitment construction such as a Merkle Tree), making the protocol a two-round protocol on an amortized basis.

	Protocol	Rounds	Communication	Assumptions
§2.2	**Bicorn-ZK**	2	$n(\langle\mathbb{G}\rangle + \langle\mathcal{B}\rangle + \lvert\pi\rvert)$	RSW, ZK-PoKE
§2.3	**Bicorn-PC**	3	$n(\langle\mathbb{G}\rangle + \langle\mathcal{B}\rangle + \lambda)$	RSW, AGM
§2.4	**Bicorn-RX**	2	$n(\langle\mathbb{G}\rangle + \langle\mathcal{B}\rangle)$	RSW, AGM, ROM

Table 1: A brief comparison of the Bicorn variants. See Figure 1 for notation ($\langle\mathbb{G}\rangle$ and $\langle\mathcal{B}\rangle$ are the sizes of elements from $\mathbb{G}$ and $\mathcal{B}$, respectively) and Section 3 for a background on the RSW assumptions, the algebraic group model (AGM), the random oracle model (ROM), and zero-knowledge proof of knowledge of exponent (ZK-PoKE).

2.4 Bicorn-RX: Using Pseudorandom Exponents

Finally, we can prevent manipulation by raising each participant's contribution c_i to a unique (small) exponent which depends on all other participants' contributions. Specifically, we define b_* to be the hash of all c_i values: $b_* = H\left(c_1||c_2||\ldots||c_n\right)$. We then raise each value c_i to the pseudorandom exponent $b_i = H(c_i \parallel b_*)$. The intuition is that modifying any contribution c_i will induce new exponents on each participant's contribution which prevents an adversary from forcing the value $\omega = \prod_{i\in[n]} c_i^{H(c_i\|b_*)}$ to a fixed value. A similar technique was used by Boneh et al. [13] to prevent rogue-key attacks in BLS multi-signatures. This version (Bicorn-RX) is specified in Protocol 1 (right).

2.5 Comparison

Each of these leads to a secure protocol, albeit reducing to slightly different computational assumptions, as we will prove in Section 5. All of our protocols reduce to the RSW assumptions with Bicorn-PC and Bicorn-RX requiring the algebraic group model (AGM) for the security reductions and Bicorn-RX also assuming a random oracle. Bicorn-ZK doesn't require the AGM explicitly but instead assumes a secure zero-knowledge proof of knowledge of exponent (ZK-PoKE) for which efficient existing protocols are proven secure only in the AGM [11].

Each protocol also offers slightly different performance trade-offs, though asymptotically all require $O(n)$ broadcast communication by participating nodes and $O(n)$ computation to verify the result. While Bicorn-PC incurs an extra round, Bicorn-ZK incurs extra computational overhead which may be significant in some scenarios (e.g. smart contracts). Bicorn-RX requires only two rounds and does not require the user to produce proofs but requires extra group exponentiations which incur slightly higher costs than Bicorn-PC.

3 Preliminaries

Algebraic group model. In some of our security proofs, we consider security against *algebraic* adversaries which we model using the algebraic group model,

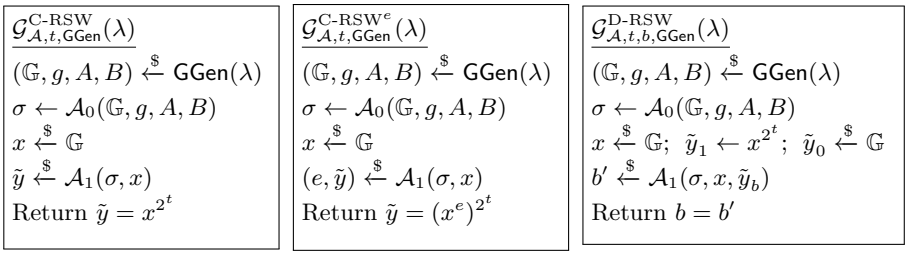

Fig. 1: Security games for the repeated squaring hardness assumptions: computational RSW (left), computational power-of-RSW (center), and decisional RSW (right).

following the treatment of [25]. We call an algorithm $\mathcal{A}$ *algebraic* if for all group elements Z that are output (either as final output or as input to oracles), $\mathcal{A}$ additionally provides the representation of Z relative to all previously received group elements. The previously received group elements include both original inputs to the algorithm and outputs received from calls to oracles. More specifically, if $[X]_i$ is the list of group elements $[X_0, \ldots, X_n] \in \mathbb{G}$ that $\mathcal{A}$ has received so far, then, when producing group element Z, $\mathcal{A}$ must also provide a list $[z]_i = [z_0, \ldots, z_n]$ such that $Z = \prod_i X_i^{z_i}$.

Groups of unknown order and RSW assumptions. Our protocols will operate over cyclic groups of unknown order. We assume an efficient group generation algorithm $\mathsf{GGen}(\lambda)$ that takes as input security parameter λ and outputs a group description $\mathbb{G}$, generator g, and range $[A, B]$ where A, B, and $B - A$ are all exponential in λ; the group $\mathbb{G}$ has order in range $[A, B]$. We assume efficient algorithms for sampling from the group ($g \xleftarrow{\$} \mathbb{G}$) and for testing membership.

There are a few currently known options with which to instantiate a group of unknown order. One option that requires only a transparent setup is through class groups of imaginary quadratic order [15]. However, class groups typically incur high concrete overheads. Instead, one may opt for more efficient RSA groups, which require a trusted setup or multiparty computation "ceremony" [21] to compute the modulus $N = pq$ without revealing safe primes p, q. Looking forward, we will require our group to additionally be cyclic and satisfy the low order assumption [12]. So instead we will use the group $\mathbb{QR}^+_N$, the group of signed quadratic residues modulo N (we refer to Pietrzak for more details [30]).

The security of our constructions is based on the assumption, originally proposed by RSW [33], that, given a random element $x \in \mathbb{G}$, the fastest algorithm to compute $y = x^{(2^t)}$ takes t sequential steps. We use three RSW assumptions; we provide security games in Figure 1.

Randomizing exponent sizes. We recall a useful lemma for randomizing group elements [29].

Lemma 1. *For any cyclic group $\mathbb{G}$ and generator g, if $r \xleftarrow{\$} \mathcal{B}$ is chosen uniformly at random, then the statistical distance between g^r and the uniform distribution over $\mathbb{G}$ is at most $\frac{|\mathbb{G}|}{2|\mathcal{B}|}$.*

Looking forward, we will use this lemma in our security proofs to replace a generator taken to the power of a large exponent of size $|\mathcal{B}| \approx 2^{2\lambda} \cdot |\mathbb{G}|$ with a random element. Alternatively, one may opt for the stronger *short exponent indistinguishability (SEI) assumption* [23] which asserts that an adversary cannot computationally distinguish between a uniformly random element of $\mathbb{G}$ and g^r for $r \xleftarrow{\$} [0, 2^{2\lambda}]$. The latter assumption enables significant efficiency gains in practice, with participants publishing 32-byte α values instead of 288 bytes.

Non-interactive zero-knowledge proofs. A *non-interactive proof system* for a relation $\mathcal{R}$ over *statement-witness* pairs (x, w) enables producing a proof, $\pi \leftarrow \mathsf{Prove}(pk, x, w)$, that convinces a verifier $\exists w : (x, w) \in \mathcal{R}$, $0/1 \leftarrow \mathsf{Verify}(vk, \pi, x)$; pk and vk are proving and verification keys output by a setup, $(pk, vk) \leftarrow \mathsf{Keygen}(\mathcal{R})$. A *non-interactive argument of knowledge* further convinces the verifier not only that the witness w exists but also that the prover *knows* w, and if proved in *zero-knowledge*, the verifier does not learn any additional information about w. In this work, we will make use of proof systems for two relations. First, we use PoE for the following relation for proofs of exponentiation in groups of unknown order [11,30,41]: $\{((x, y \in \mathbb{G}, \alpha \in \mathbb{Z}), \perp) : y = x^\alpha\}$. Second, we use ZKPoKE (realized by $\mathsf{ZKPoKRep}$ from [11]) for zero-knowledge proofs of knowledge of exponent in groups of unknown order: $\{((x, y \in \mathbb{G}), \alpha \in \mathbb{Z}) : y = x^\alpha\}$.

4 Timed DRBs: Syntax and Security Definitions

We first define a timed DRB using a generalized syntax which captures all of our protocol variants. A timed DRB protocol DRB with time parameter t is a tuple of algorithms $(\mathsf{Setup}, \mathsf{Prepare}, \mathsf{Finalize}, \mathsf{Recover})$. We describe them below for a run of the protocol with n participants:

- $\mathsf{Setup}(\lambda, t) \xrightarrow{\$} \mathsf{pp}$: The setup algorithm takes as input a security parameter λ and a time parameter t and outputs a set of public parameters pp.
- $\mathsf{Prepare}(\mathsf{pp}) \xrightarrow{\$} (\alpha_i, c_i, d_i, \pi_i)$: The prepare algorithm is run by each participant and outputs a tuple of opening, commitment, precommitment, and proof. The precommitment is contributed during the $\mathsf{Precommit}$ phase (see Protocol 1). The commitment and proof are contributed during the Commit phase, and the opening is contributed during the Reveal phase. The length of the Commit phase is dictated by the time parameter t.
- $\mathsf{Finalize}(\mathsf{pp}, \{(\alpha_i, c_i, d_i, \pi_i)\}_{i=1}^n) \to \Omega$: The finalize algorithm is run after the Reveal phase and verifies the contributions of participants to optimistically produce a final output Ω or returns $\perp$ indicating the need to move to the pessimistic case.
- $\mathsf{Recover}(\mathsf{pp}, \{(c_i, d_i, \pi_i)\}_{i=1}^n) \to \Omega$: The recover algorithm performs the timed computation to recover the output Ω without any revealed α values.

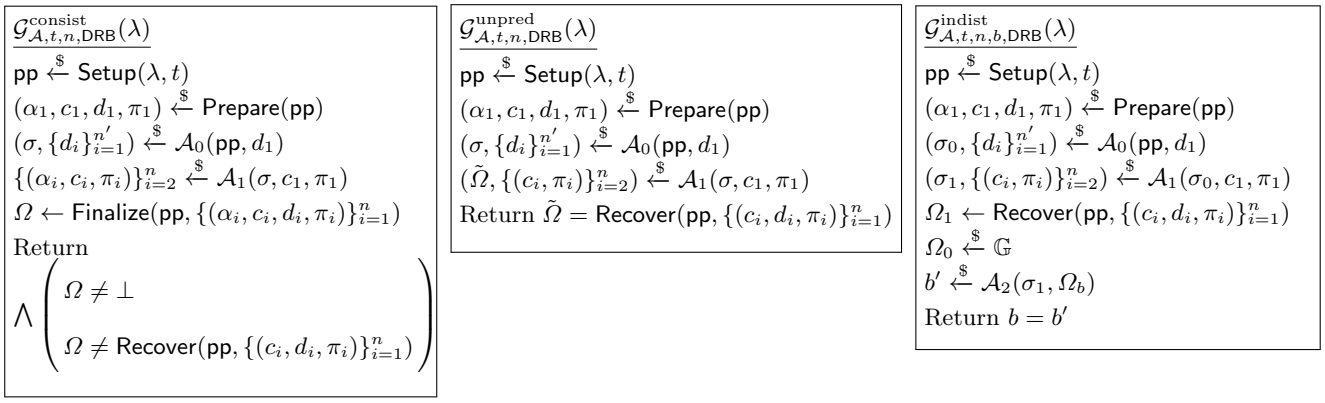

Fig. 2: Security games for our three main security properties: consistency (left), t-unpredictability (center), and t-indistinguishability (right).

We require $\mathsf{Finalize}$ to be a deterministic algorithm running in time $\mathsf{polylog}(t)$ (the fast optimistic case), and $\mathsf{Recover}$ to be a deterministic algorithm running in time $(1+\epsilon)t$ for some small ϵ. We also require the following security properties of a timed DRB (given in pseudocode in Figure 2):

Consistency. Our first security property is a form of correctness. We require that it is not possible for the optimistic and pessimistic paths to return different outputs. The adversary is tasked with providing an accepting set of contributions that results in different outputs from $\mathsf{Finalize}$ and $\mathsf{Recover}$. We define the advantage of an adversary as $\mathsf{Adv}^{\text{consist}}_{\mathcal{A},t,n,\mathsf{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{consist}}_{\mathcal{A},t,n,\mathsf{DRB}}(\lambda) = 1\right]$.

t-Unpredictability. The t-unpredictability game tasks an adversary with predicting the final output Ω exactly, allowing it control of all but a single honest protocol participant (which publishes first). We define the advantage of an adversary as $\mathsf{Adv}^{\text{unpred}}_{\mathcal{A},t,n,\mathsf{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,n,\mathsf{DRB}}(\lambda) = 1\right]$.

t-Indistinguishability. The t-unpredictability property does not guarantee the output is indistinguishable from random. For that, we provide a stronger t-indistinguishability property in which the adversary must distinguish an honest output from a random output, again allowing the adversary control of all but one participant. We define the advantage of an adversary as: $\mathsf{Adv}^{\text{indist}}_{\mathcal{A},t,n,\mathsf{DRB}}(\lambda) = \left|\Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,n,1,\mathsf{DRB}}(\lambda) = 1\right] - \Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,n,0,\mathsf{DRB}}(\lambda) = 1\right]\right|$. A timed DRB that satisfies t-unpredictability can be transformed generically into one with t-indistinguishability by applying a suitable randomness extractor [39, 40] or hash function (modeled as a random oracle) to the output. A nice feature of our DRBs is that they satisfy t-indistinguishability with respect to the group output space (without applying a randomness extractor) under the suitable decisional RSW assumption.

Discussion. In t-unpredictability and t-indistinguishability, the adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ are restricted to run in fewer than t sequential steps. This is a slight simplification of the (p, σ)-sequentiality assumption in VDFs [10], which is suitable for working in the AGM in which parallelism is not helpful in computing group operations.

Note that our syntax and security definitions encompass all three of our protocol variants. Except for Bicorn-ZK, the proofs π_i can be set to $\perp$ and are ignored; except for Bicorn-PC, the precommitment values d_i can be set to $\perp$ and are ignored. Also note that there are n' ($\geq n$) values of d_i output by the adversary; they have the option in Bicorn-PC to choose which to use in later steps. The implementation of $\mathsf{Recover}$ is unique to each protocol.

We observe that the consistency property holds unconditionally for all Bicorn variants, as $\mathsf{Finalize}$ and $\mathsf{Recover}$ are deterministic and algebraically equivalent. It remains to prove unpredictability and indistinguishability for each variant.

5 Security of Bicorn-RX

We present a proof of t-unpredictability for Bicorn-RX here, as it is representative of the techniques used for all other proofs.

Theorem 1 (t-Unpredictability of Bicorn-RX). *Let $\mathcal{A}_{\mathrm{brx}} = (\mathcal{A}_{\mathrm{brx},0}, \mathcal{A}_{\mathrm{brx},1})$ be an algebraic adversary against the t-unpredictability of BRX with random exponent space $\mathcal{B} = [2^{2\lambda} \cdot B]$ where hash function H is modeled as a random oracle. Then we construct an adversary $\mathcal{A}_{\mathrm{rsw}} = (\mathcal{A}_{\mathrm{rsw},0}, \mathcal{A}_{\mathrm{rsw},1})$ such that*

$$\mathsf{Adv}^{\mathrm{unpred}}_{\mathcal{A}_{\mathrm{brx}},t,n,\mathsf{BRX}}(\lambda) \leq \mathsf{Adv}^{\text{C-RSW}^e}_{\mathcal{A}_{\mathrm{rsw}},t,\mathsf{GGen}}(\lambda) + \frac{2(\mathsf{q}_{\mathrm{ro}}^2 + n) + 1}{2^{2\lambda+1}} + \prod_{i=1}^{\ell} I_{\frac{1}{p_i}}(r_i, n)\,,$$

and where $\mathsf{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ generates the group of unknown order ($|\mathbb{G}| = \prod_{i=1}^{\ell} p_i^{r_i}$ for distinct primes $p_1, \ldots, p_\ell$) used by BRX, q_{ro} is the number of queries made to the random oracle, n is the number of participants, and $I_{\frac{1}{p}}(r, n) = (1 - \frac{1}{p})^n \sum_{j=r}^{\infty} \binom{n+r-1}{r} p^{-j}$ is the regularized beta function. The running time of $T(\mathcal{A}_{\mathrm{rsw},0}) \approx T(\mathcal{A}_{\mathrm{brx},0}) + 2t$ and $T(\mathcal{A}_{\mathrm{rsw},1}) \approx T(\mathcal{A}_{\mathrm{brx},1})$.

Proof. At a high level, our proof strategy will be to replace the initial commitment c_1 provided by the single honest participant with a random group element. If $\mathcal{A}_{\mathrm{brx}}$ can win with non-negligible probability, then we show that due to unpredictability of the random exponents applied in Bicorn-RX, it must be that a nontrivial large exponent of c_1 was computed which we can use to win the computational power-of-RSW game.

More specifically, we bound the advantage of $\mathcal{A}_{\mathrm{brx}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [6]. We define $\mathcal{G} = \mathcal{G}^{\mathrm{unpred}}_{\mathcal{A}_{\mathrm{brx}},t,n,\mathsf{BRX}}(\lambda)$ and hybrids $\mathcal{G}_1$,$\mathcal{G}_2$,$\mathcal{G}_3$ for which we justify the following claims leading to the inequality above:

- $|\Pr[\mathcal{G}(\lambda) = 1] - \Pr[\mathcal{G}_1(\lambda) = 1]| \leq \frac{1}{2^{2\lambda+1}}$

- $|\Pr[\mathcal{G}_1(\lambda)=1] - \Pr[\mathcal{G}_2(\lambda)=1]| \leq \frac{\mathsf{q}_{\mathrm{ro}}^2}{2^{2\lambda}}$
- $|\Pr[\mathcal{G}_2(\lambda)=1] - \Pr[\mathcal{G}_3(\lambda)=1]| \leq \frac{n}{2^{2\lambda}} + \prod_{i=1}^{\ell} I_{\frac{1}{p_i}}(r_i, n)$
- $\Pr[\mathcal{G}_3(\lambda)=1] = \mathsf{Adv}^{\text{C-RSW}^e}_{\mathcal{A}_{\mathrm{rsw}},t,\mathsf{GGen}}(\lambda)$

$\underline{\mathcal{G} \to \mathcal{G}_1}$**.** Hybrid $\mathcal{G}_1$ is defined the same as $\mathcal{G}$ except $\mathcal{G}_1$ samples c_1 in $\mathsf{Prepare}$ at random from $\mathbb{G}$ instead of through an exponent sampled from $\mathcal{B}$. By Lemma 1, the statistical distance between $\mathcal{G}$ and $\mathcal{G}_1$ is at most $1/2^{2\lambda+1}$.

We can view $\mathcal{G}_1$ as computing the beacon output Ω using the representations of $\{c_i\}_{i=2}^n$ provided by the algebraic adversary. Since $\mathcal{A}_{\mathrm{brx}}$ is algebraic, it will provide a representation for each c_i in terms of elements (c_1, g, h). That is, the adversary outputs $[(e_{i,0}, e_{i,1}, e_{i,2})]_{i=2}^n$ such that $c_i = c_1^{e_{i,0}} g^{e_{i,1}} h^{e_{i,2}}$.

Given a value $\hat{h} = h^{2^t}$, we can compute Ω as follows. Consider the random exponents $b_i = H(c_i \parallel b_*)$ where $b_* = H(c_1 || \ldots || c_n)$, and let $\mathbf{b} = (b_1, \ldots, b_n)$. Using these, we have:

$$\begin{aligned}
\Omega &= \left(\prod_{i=1}^{n} c_i^{b_i}\right)^{2^t} = \left(c_1^{b_1} \cdot \prod_{i=2}^{n} \left(c_1^{e_{i,0}} g^{e_{i,1}} h^{e_{i,2}}\right)^{b_i}\right)^{2^t} \\
&= \left(c_1^{b_1 + \sum_{i=2}^n b_i e_{i,0}} g^{\sum_{i=2}^n b_i e_{i,1}} h^{\sum_{i=2}^n b_i e_{i,2}}\right)^{2^t} \\
&\text{By letting } \mathbf{e} = (1, e_{2,0}, \ldots, e_{n,0}),\ m_1 = \sum_{i=2}^{n} b_i e_{i,1}, \text{ and } m_2 = \sum_{i=2}^{n} b_i e_{i,2}, \\
&= \left(c_1^{\langle \mathbf{b}, \mathbf{e} \rangle} g^{m_1} h^{m_2}\right)^{2^t} = (c_1^{2^t})^{\langle \mathbf{b}, \mathbf{e} \rangle} \cdot h^{m_1} \cdot \hat{h}^{m_2}
\end{aligned}$$

Thus if $\mathcal{A}_{\mathrm{brx}}$ wins, i.e., $\tilde{\Omega} = \Omega$, then we have

$$(c_1^{2^t})^{\langle \mathbf{b}, \mathbf{e} \rangle} = \tilde{\Omega} \cdot h^{-m_1} \cdot \hat{h}^{-m_2}$$

and we build $\mathcal{A}_{\mathrm{rsw}}$ to win the computational power-of-RSW game by setting c_1 equal to challenge element x and returning this value along with $\langle \mathbf{b}, \mathbf{e} \rangle$. All that is left to show is that $\langle \mathbf{b}, \mathbf{e} \rangle \neq 0$ which we can do through an application of the Schwartz-Zippel lemma modulo a composite [17, 36, 43]. Define a non-zero polynomial $f(x_1, \ldots, x_n) = x_1 + \sum_{i=2}^n x_i e_{i,0}$. Note that $f(\mathbf{b}) = \langle \mathbf{b}, \mathbf{e} \rangle$.

$\underline{\mathcal{G}_1 \to \mathcal{G}_2}$**.** To apply the Schwartz-Zippel lemma modulo a composite, we must first have that the evaluation point $\mathbf{b}$ does not coincide with values precomputed by the adversary. To do this, we step through $\mathcal{G}_2$ in which we disallow the output of the random oracle H from colliding with (the trailing substring of) any previous inputs to the random oracle. This ensures that the adversary has not made any previous queries that include b_* and ultimately ensures that the b_i values are chosen randomly *after* the polynomial is decided. We can apply a standard birthday analysis to bound the probability of collision among the q_{ro} queries made to $\mathsf{q}_{\mathrm{ro}}^2/2^{2\lambda}$, to bound the distinguishing advantage between $\mathcal{G}_1$ and $\mathcal{G}_2$.

$\underline{\mathcal{G}_2 \to \mathcal{G}_3}$**.** After we have that the evaluation point $\mathbf{b}$ does not coincide with precomputed values, we transition to $\mathcal{G}_3$ which is identical to $\mathcal{G}_2$ except it

		Gas Costs ($\times 10^3$), Operations Involved						
		Commit/user		**Reveal**/user		**Recover**		
	Commit-Reveal	50	$\mathsf{store}_{2\lambda}$	60	xor, hash	-		
[28]	**Unicorn**	55	$\mathsf{store}_{2\lambda}$		-	$30n$	n·hash	+2,330 poe.v
§2.2	**Bicorn-ZK**	2,950	zk-poke.v, $\mathsf{store}_{\mathbb{G}}$	300	exp, mul	(*negligible*)		
§2.3	**Bicorn-PC**	155; 180	mul, $\mathsf{store}_{\mathbb{G}}$	300	exp, mul	(*negligible*)		
§2.4	**Bicorn-RX**	145	mul, $\mathsf{store}_{\mathbb{G}}$	425	2·exp, mul	170n	n·exp	

Table 2: Ethereum gas costs and main operations involved for each Bicorn variant as well as Unicorn and Commit-Reveal DRBs. For Bicorn-PC, the Commit cost is split to show Precommit and Commit costs. The operations are: $\mathsf{store}_{\mathbb{G}/2\lambda}$, storing a group element or 2λ-bit value; mul, multiplication of two group elements; exp, raising a group element to a power of size 2λ bits; poe.v and zk-poke.v, verifying a proof of exponentiation and proof of knowledge of exponent, respectively. Concrete costs are given with $\mathbb{G} = \mathbb{QR}_N^+$ within an RSA-2048 group and $\lambda = 128$.

aborts if $f(\mathbf{b}) = 0$. We bound the distinguishing advantage to probability $\frac{n}{2^{2\lambda}} + \prod_{i=1}^{\ell} I_{\frac{1}{p_i}}(r_i, n)$ by applying Schwartz-Zippel modulo a composite [17]. Adversary $\mathcal{A}_{\text{rsw}}$ can simulate $\mathcal{G}_3$ perfectly, simulating the setup and computing $\hat{h}$ with $2t$ work, and wins the RSW game with the same advantage as $\mathcal{G}_3$. ■

6 Implementation

We implemented all three variants of Bicorn in Solidity and measured the associated gas costs in Ethereum [42]. Our results are presented in Table 2. We instantiate $\mathbb{G}$ as an RSA group with a 2048-bit modulus (specifically, it is the quadratic residue subgroup $\mathbb{QR}_N^+$ [30]). Multiplying two group elements costs ∼90,000 gas and raising a group element to a power of size 32 bytes costs ∼150,000 gas. As mentioned in Section 3, we use the short exponent indistinguishability (SEI) assumption [23] to reduce the size of the exponent required in practice from 288 to 32 bytes. The largest costs for each protocol are verifying a proof of exponentiation (PoE) for the VDF computation in the pessimistic Recover case and verifying a zero-knowledge proof of knowledge of exponent needed for each commitment in Bicorn-ZK. We implemented both proofs using non-interactive variants of Wesolowski proofs (ZKPoKRep from [11] for the latter), which requires a prime challenge to be sampled. Verifying this "hash-to-prime" operation costs between 2.3–4 million gas.[5]

[5] Verifying "hash-to-prime" involves testing the primality of a number on-chain using Pocklington certificates. This costs between 2.3–4 million gas, depending on the size of the certificate. Table 2 reports costs with the smallest possible certificate.

Comparison to other DRBs. *Per-user Costs:* We find that the user operations for Bicorn-RX are practical on Ethereum with them costing 3× for Commit and 7× for Reveal when compared to the standard Commit-Reveal and Unicorn protocols. In total, the sum of these operations per user per run comes to under 600,000 gas, or $6 USD when 1 Eth = $1,000 USD and 1 gas = 10 Gwei.
Pessimistic Costs: In the pessimistic case, a single call to Recover is required in all versions of Bicorn, costing millions of gas. This pessimistic case is roughly equivalent to *every* run of Unicorn. As the number of users grows large and the chances of Bicorn's optimistic case occurring decrease though, at some point it may make more sense to switch to Unicorn and avoid the overheads of Commit and Reveal that Bicorn protocols incur.

7 Discussion

Last revealer prediction. All Bicorn variants come with a fundamental security caveat: if participant j withholds their α_j value, but all others publish, then participant j will be able to simulate the optimistic case and learn Ω quickly, while the honest participants will need to execute the pessimistic case and compute the delay function to complete before learning Ω. Similarly, a coalition of malicious participants can share their α values and privately compute Ω. This issue appears fundamental; in any protocol with a fast optimistic case and a slow pessimistic case, a unified malicious coalition can simulate the optimistic case.

This does not undermine t-unpredictability or t-indistinguishability and does not allow an adversary to manipulate the outcome. As a result, any protocol built on top of Bicorn should consider the output Ω to be potentially available to adversaries as of the deadline T_1, even if the result is not publicly known until $T_1 + t$ if the pessimistic case is triggered. For example, in a lottery application all wagers must be locked in before time T_1.

Incentives and punishment. While all Bicorn variants ensure malicious participants cannot manipulate the output, they can waste resources by forcing the protocol into the more-expensive recovery mode. The protocol provides accountability as to which nodes published an incorrect α_i value or other minor deviations which lead to removal (i.e. publishing an incorrect c_i such that $H(c_i) \neq d_i$ in Bicorn-PC or publishing an incorrect π_i in Bicorn-ZK). If signatures are added to each message, efficient fraud proofs are possible. In a blockchain setting, financial penalties can be used to punish incorrect behavior.

Batch verification optimization. In the optimistic case, the n exponentiations required to verify that $c_i = g^{\tilde{\alpha}_i}$ for each participant can be streamlined via batch verification [5, 16]. The general idea is that $g^x = 1 \wedge g^y = 1$ can be verified more efficiently by checking $g^{r \cdot x + y} = 1$ for a random $r \xleftarrow{\$} \mathcal{R}$, as the latter equation implies the former with high probability given a large enough $\mathcal{R}$. In our case, to verify that $c_1 = g^{\tilde{\alpha}_1} \wedge c_2 = g^{\tilde{\alpha}_2} \wedge \ldots \wedge c_n = g^{\tilde{\alpha}_n}$, we generate random values $r_i \xleftarrow{\$} \mathcal{R}$ and verify that $g^{\sum r_i \cdot \tilde{\alpha}_i} = \prod c_i^{r_i}$. Thus, instead of

computing n exponentiations each with an exponent of size $|\mathcal{B}|$, verification requires only one exponentiation with an exponent of size $n|\mathcal{B}||\mathcal{R}|$ and one n-way multi-exponentiation [31].

Lowering costs with rollup proofs. Practical costs can become significant if all users must post data to the blockchain to participate. For example, each run of Bicorn-RX costs about $6 USD per user even in the optimistic case. An alternative solution is to perform Bicorn mediated via a *rollup server* (Rollup-Bicorn) which gathers every participant's c_i value and publishes:

- A commitment $s = \mathsf{SetCommitment}(C)$ to the set $C = \{c_1, \ldots, c_n\}$ of all participant contributions. For example, s might be a Merkle Tree root.
- The value $c_* = \prod_{i \in [n]} c_i$, the product of all participants' commitments.
  - For Bicorn-RX, c_* will be adjusted with each party's exponent $H(c_i||b_*)$.
- A succinct proof (SNARK) $\pi_{\text{rollup-commit}}$ that c_* has been computed consistently with the set S. This proof does not need to be zero-knowledge.
  - For Bicorn-ZK, the proof must recursively check each proof π_i.
  - For Bicorn-PC, the proof must check c_i was correctly precommitted.
  - For Bicorn-RX, the proof must check c_i was raised to the power b_i.

In the optimistic case, if all participants reveal their private value α_i, then the rollup server can finalize the protocol by posting:

- The output Ω and a succinct proof (SNARK) $\pi_{\text{rollup-finalize}}$ that states that:
  - The prover knows a set $A = \{\alpha_1, \ldots, \alpha_n\}$
  - For each $c_i \in C$, it holds that $c_i = g^{\alpha_i}$
  - The output Ω was computed correctly given the set A.

In the pessimistic case, if the rollup server goes offline without supplying the second proof (or some participants don't publish α_i), anybody can still compute $\Omega = c_*^{(2^t)}$. A single proof could be used which is a disjunction of verifying the rollup server's proof $\pi_{\text{rollup-finalize}}$ or verifying a PoE proof that $\Omega = c_*^{2^t}$. The end result is that Bicorn can be run with $O(1)$ cost for any number of participants.

Lowering cost with delegation. While the rollup approach requires only constant overhead on the blockchain regardless of the number of participants, the primary downside (in common with most rollup systems) is that the rollup server can *censor* by refusing to include any participant's c_i in the protocol. In the worst case, a malicious rollup server might only allow participants from a known cabal to participate, who are then able to manipulate the DRB output.

To achieve the best of both worlds (the efficiency of rollup servers for large protocol runs as well as robustness against censorship), we might design a *delegated* Bicorn protocol. In a delegated protocol, users can choose between multiple rollup servers or directly participate as an untrusted (possibly singleton) rollup server. This works like delegated proof-of-stake protocols: participants can delegate for efficiency if they want or participate individually if no server is considered trustworthy. This is straightforward for Bicorn-PC and Bicorn-ZK, as each

rollup server can simply compute a partial product c_* which are multiplied together to obtain the final output Ω. Such a protocol for Bicorn-RX would require additional rounds of exponent randomization, to ensure each user's exponent is randomized by contributions from users at other rollup servers.

Acknowledgments

Kevin Choi, Arasu Arun and Joseph Bonneau were supported by DARPA under Agreement No. HR00112020022. Nirvan Tyagi was supported via a Facebook Graduate Fellowship, and part of this work was done while he was a visiting student at Stanford University. Joseph Bonneau and Arasu Arun were also supported by a16z crypto research. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the United States Government, DARPA, a16z, Facebook or any other supporting organization.

References

1. Drand. `https://drand.love/`
2. Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure Multiparty Computations on Bitcoin. In: IEEE Security & Privacy (2014)
3. Beaver, D., Chalkias, K., Kelkar, M., Kogias, L.K., Lewi, K., de Naurois, L., Nicolaenko, V., Roy, A., Sonnino, A.: Strobe: Stake-based threshold random beacons. Cryptology ePrint Archive (2021)
4. Beaver, D., So, N.: Global, unpredictable bit generation without broadcast. In: Eurocrypt (1993)
5. Bellare, M., Garay, J.A., Rabin, T.: Fast batch verification for modular exponentiation and digital signatures. In: Eurocrypt (1998)
6. Bellare, M., Rogaway, P.: The security of triple encryption and a framework for code-based game-playing proofs. In: EUROCRYPT. Lecture Notes in Computer Science, vol. 4004, pp. 409–426. Springer (2006)
7. Bhat, A., Kate, A., Nayak, K., Shrestha, N.: OptRand: Optimistically responsive distributed random beacons. Cryptology ePrint Archive, Paper 2022/193 (2022)
8. Bhat, A., Shrestha, N., Kate, A., Nayak, K.: RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication. Cryptology ePrint Archive, Paper 2020/1590 (2020)
9. Blum, M.: Coin flipping by telephone a protocol for solving impossible problems. ACM SIGACT News (1983)
10. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (2018)
11. Boneh, D., Bünz, B., Fisch, B.: Batching techniques for accumulators with applications to IOPs and stateless blockchains. In: CRYPTO (2019)
12. Boneh, D., Bünz, B., Fisch, B.: A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Paper 2018/712 (2018)
13. Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Asiacrypt (2018)
14. Boneh, D., Naor, M.: Timed commitments. In: Annual international cryptology conference (2000)
15. Buchmann, J., Hamdy, S.: A survey on IQ cryptography. In: Public-Key Cryptography and Computational Number Theory (2011)
16. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Short proofs for confidential transactions and more. In: IEEE Security & Privacy (2018)
17. Bünz, B., Fisch, B.: Schwartz-zippel for multilinear polynomials mod n. Cryptology ePrint Archive, Paper 2022/458 (2022)
18. Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.A., Shoup, V., Williams, D.: Internet computer consensus. In: ACM PODC (2022)
19. Cascudo, I., David, B.: Scrape: Scalable randomness attested by public entities. In: ACNS (2017)
20. Cascudo, I., David, B.: Albatross: publicly attestable batched randomness based on secret sharing. In: Asiacrypt (2020)
21. Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., Wang, R.: Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. In: IEEE Security & Privacy (2021)
22. Cherniaeva, A., Shirobokov, I., Shlomovits, O.: Homomorphic encryption random beacon. Cryptology ePrint Archive, Paper 2019/1320 (2019)

23. Couteau, G., Klooß, M., Lin, H., Reichle, M.: Efficient range proofs with transparent setup from bounded integer commitments. In: Eurocrypt (2021)
24. Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. Cryptology ePrint Archive, Paper 2021/100 (2021)
25. Fuchsbauer, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO (2018)
26. Guo, Z., Shi, L., Xu, M.: SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability. IEEE Access (2020)
27. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: CRYPTO (2017)
28. Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366 (2015)
29. Micciancio, D.: The RSA group is pseudo-free. In: CRYPTO (2005)
30. Pietrzak, K.: Simple Verifiable Delay Functions. In: ITCS (2018)
31. Pippenger, N.: On the evaluation of powers and monomials. SIAM Journal on Computing **9**(2), 230–250 (1980)
32. Qian, Y.: Randao: Verifiable random number generation (2017), `https://randao.org/whitepaper/Randao_v0.85_en.pdf`
33. Rivest, R.L., Shamir, A., Wagner, D.A.: Time-lock puzzles and timed-release crypto (1996)
34. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Efficient continuous distributed randomness. In: IEEE Security & Privacy (2020)
35. Schoenmakers, B.: A simple publicly verifiable secret sharing scheme and its application to electronic voting. In: CRYPTO (1999)
36. Schwartz, J.T.: Fast probabilistic algorithms for verification of polynomial identities. Journal of the ACM (JACM) **27**(4), 701–717 (1980)
37. Syta, E., Jovanovic, P., Kogias, E.K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M.J., Ford, B.: Scalable bias-resistant distributed randomness. In: IEEE Security & Privacy (2017)
38. Thyagarajan, S.A.K., Castagnos, G., Laguillaumie, F., Malavolta, G.: Efficient CCA Timed Commitments in Class Groups. Cryptology ePrint Archive, Report 2021/1272 (2021)
39. Trevisan, L.: Extractors and pseudorandom generators. Journal of the ACM **48**(4) (2001)
40. Trevisan, L., Vadhan, S.: Extracting randomness from samplable distributions. In: FOCS (2000)
41. Wesolowski, B.: Efficient Verifiable Delay Functions. In: Eurocrypt (2019)
42. Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper (2014)
43. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Symbolic and algebraic manipulation (1979)