Cornucopia: Distributed Randomness at Scale

Miranda Christ
Columbia University, New York, NY, USA

Kevin Choi
New York University, NY, USA

Joseph Bonneau
New York University, NY, USA
a16z crypto research, New York, NY, USA

Abstract

We propose Cornucopia, a protocol framework for distributed randomness beacons combining accumulators and verifiable delay functions. Cornucopia generalizes the Unicorn protocol, using an accumulator to enable efficient verification by each participant that their contribution has been included. The output is unpredictable as long as at least one participant is honest, yielding a scalable distributed randomness beacon with strong security properties. Proving this approach secure requires developing a novel property of accumulators, *insertion security*, which we show is both necessary and sufficient for Cornucopia-style protocols. We show that not all accumulators are insertion-secure, then prove that common constructions (Merkle trees, RSA accumulators, and bilinear accumulators) are either naturally insertion-secure or can be made so with trivial modifications.

2012 ACM Subject Classification Security and privacy → Cryptography

Keywords and phrases Randomness beacons, accumulators

Digital Object Identifier 10.4230/LIPIcs.AFT.2024.17

Related Version *Full Version*: `https://eprint.iacr.org/2023/1554` [20]

Funding 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, Andreessen Horowitz, or any other supporting organization.
Miranda Christ: Supported by NSF grants CCF-2107187 and CCF-2212233, by LexisNexis Risk Solutions, and by the Algorand Centres of Excellence programme managed by Algorand Foundation.
Kevin Choi: Supported by DARPA Agreement HR00112020022 and NSF grant CNS-2239975.
Joseph Bonneau: Supported by DARPA Agreement HR00112020022, NSF grant CNS-2239975, and a16z crypto research.

Acknowledgements The authors thank Noemi Glaeser for suggesting the name Cornucopia.

1 Introduction

The goal of distributed randomness beacons (DRBs) is to enable n participants to jointly compute a random output (which we denote Ω) that cannot be predicted or biased by a malicious subset of the participants. Among many important applications of DRBs are cryptographically verifiable lotteries and leader election in consensus protocols.

A classic approach to constructing DRBs is *commit-reveal* [8]. First, all participants publish a cryptographic commitment to a random contribution r_i. Participants then 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 any one participant chooses a random r_i and all participants open their commitments. However, the output can be biased via a so-called *last-revealer* *attack*, in which a participant observes all other r_i values during the reveal phase and drops out if the impending value of Ω is not to their liking. The protocol must either finish without the missing r_i, or restart completely. Either way, the attacker obtains 1 bit of bias on Ω.

Most approaches to avoiding last-revealer attacks enable a majority coalition to recover a withholding participant's contribution. However, this downgrades the security model from requiring only honest participant to requiring an honest majority (to prevent a malicious coalition from privately computing Ω early). Such protocols also typically require communication and computation superlinear in n (though some amortize this over multiple rounds).

A fundamentally different approach constructs DRBs uses time-based cryptography, specifically *delay functions*, to prevent manipulation. The simplest example is Unicorn [38], a one-round protocol in which each participant directly publishes (within a fixed time window) their contribution r_i to a public bulletin board. The result is computed as $\Omega = \mathsf{Delay}(\mathsf{Combine}(r_1, \ldots, r_n))$. By assumption, a participant cannot compute the Delay function before the deadline to publish their r_i and therefore cannot choose r_i in such a way as to manipulate the output Ω. This protocol retains the strong security model of commit-reveal, but with no last-revealer attacks. It is remarkably simple and, using modern verifiable delay functions [9], the result can be efficiently verified. The downside is that $\Theta(n)$ contributions must be posted to the public bulletin board per protocol run.

Improving efficiency with accumulators. Unicorn is simple and robust, but requires publishing $\Theta(n)$ data (one contribution per participant) on the public bulletin board. To reduce this cost to $O(1)$, we can instead publish a succinct commitment to all users' contributions using a cryptographic accumulator (for example, a Merkle tree). We formalize this approach as *Cornucopia*:

- Each participant sends their contribution r_i to a *coordinator* before a time deadline T_0.
- The coordinator accumulates all contributions into a succinct commitment R and publishes it to the bulletin board. It sends each user a proof π_i that their value r_i is included in R.
- After time t passes, the result $\Omega = \mathsf{Delay}(R)$ is published as well as a proof π_Ω.
- Each user i checks both that their contribution r_i was included in R and that Ω was properly computed from R.

While this is a small change to Unicorn, it is powerful. Since security requires only one honest participant there is no risk to allowing more participants. Honest participants need only verify that *they themselves participated in the protocol* (assuming they trust that their own device has not been compromised) and need not know about the full set of participants. The only downside to additional participants is performance, and Cornucopia's sub-linear verification cost means the approach is feasible for *open-participation* randomness protocols at planetary scale (i.e. millions or billions of participants). For example, every user buying a lottery ticket or every player in a massively multi-player online (MMO) game might contribute randomness and be convinced the process was fair.

A malicious coordinator and any number of other malicious participants in the protocol cannot manipulate the DRB output. A malicious coordinator might exclude all honest users from participating, but these users can easily see that they have been excluded and know not to trust the DRB output. For this reason, the coordinator can be viewed as *semi-trusted*; it is trusted for liveness but not for security. We could also consider the coordinator *malicious-but-cautious* [49], in that undermining liveness would be publicly detectable but biasing the DRB output would not be. In Section 7.1 we discuss extending Cornucopia to a multi-coordinator model with stronger liveness guarantees.

Performance-wise, the coordinator does face at least linear costs ($\Omega(n)$) to compute the accumulator and per-user proofs, but for certain accumulators [52, 55], the coordinator can efficiently batch compute all users' witnesses.

Related work. There is a large and growing literature on randomness beacons, dating to the seminal proposal by Rabin [47] and foundational work on *distributed coin tossing* [21, 3, 4, 31, 23, 34, 33]. Several recent surveys cover modern DRBs [48, 19, 35]. Most of this work is orthogonal, as protocols without delay functions either require an honest majorities [54, 17, 14, 7, 30, 32, 51, 22, 6, 24, 2] or offer only economic security [1, 46, 57].

Unicorn [38] introduced delay-based DRBs. Several extensions to Unicorn work in a similar model. Bicorn [18] extends Unicorn with a fast optimistic case, avoiding the delay function if *all* participants are honest. RandRunner [50] also enables avoiding a delay function per beacon output although it does not support flexible participation and allows a withholding leader to affect the protocol.

HeadStart [37] is the most conceptually similar approach to Cornucopia, using Merkle trees to scale Unicorn by combining many users' contributions in a succinct commitment in a multi-round, pipelined protocol. Cornucopia can be seen as a generalization of HeadStart, offering flexibility to use any accumulator and formalizing precise security notions required of accumulators for use in DRBs.

Our contributions.

- We formalize combining a VDF with an accumulator as Cornucopia (Section 3).
- We prove (in Section 4) that this approach is secure when instantiated with *any* VDF and *any* accumulator satisfying a natural security notion that we develop, *insertion security*.
- We prove (in Section 5) that the most common accumulator constructions either naturally feature insertion security (Merkle trees) or achieve it with trivial modifications (RSA accumulators, bilinear accumulators, and accumulators from vector commitments), meaning Cornucopia is practical to build from standard cryptographic assumptions and implementations. We also show that we can construct an insertion-secure accumulator generically from any universal accumulator (Section 5.6).
- We compare performance of different accumulators which can be used to instantiate Cornucopia in Section 6. No accumulator is clearly best in all settings, as different options offer different trade-offs of communication and computation cost.
- Finally, we discuss several natural extensions, including the multi-coordinator model to ensure liveness (Section 7.1) and a notarized model to provide verifiability to passive observers (Section 7.2).

2 Preliminaries

We use λ to denote a security parameter, and $\mathsf{poly}(\lambda)$ and $\mathsf{negl}(\lambda)$ to denote polynomial and negligible functions of λ, respectively. We let $[k]$ denote the set $\{1, \ldots, k\}$. We use $\overset{\$}{\leftarrow}$ (or $\overset{\$}{\rightarrow}$) to denote the output of a randomized algorithm, or sampling uniformly at random from a range. We use α to denote an advice string passed from a precomputation algorithm to a later online algorithm. We assume all adversaries are limited to running in probabilistic polynomial time (PPT) in the security parameter λ; some adversaries are further limited to running in $\sigma(t)$ steps on at most $p(t)$ parallel processors, as defined for VDF sequentiality [9]. Both VDFs [9] and accumulators [5] rely on public parameters pp which all functions require implicitly, though we will typically omit this for brevity.

```
$\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,t,\mathsf{VDF}}(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{VDF.Setup}(\lambda, t)$
$\alpha \xleftarrow{\$} \mathcal{A}_0(\mathsf{pp})$
$x \xleftarrow{\$} U$
$\tilde{y} \xleftarrow{\$} \mathcal{A}_1(\alpha, x)$
$y, \pi \leftarrow \mathsf{VDF.Eval}(\mathsf{pp}, x)$

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

Figure 1 VDF sequentiality game.

2.1 Verifiable delay functions

▶ **Definition 1** (Verifiable delay function [9]). *A* verifiable delay function *(VDF) is a tuple of algorithms* (Setup, Eval, Verify) *where:*

VDF.Setup$(\lambda, t) \to$ **pp** *takes as input λ and a time parameter t and outputs public parameters* pp.

VDF.Eval$(\mathsf{pp}, x) \to (y, \pi)$ *takes as input x and produces an output y and optional proof π. This function should run in t sequential steps.*

VDF.Verify$(\mathsf{pp}, x, y, \pi) \to \{\mathsf{true}, \mathsf{false}\}$ *takes an input x, output y, and optional proof π, and returns* true *if (y, π) is a genuine output of* Eval.

VDFs must satisfy the following three properties:

Verifiability. The verification algorithm is efficient (at most polylogarithmic in t and λ) and always accepts when given a genuine output from VDF.Eval.

Uniqueness. VDF evaluation must be a function, meaning that VDF.Eval is a deterministic algorithm and it is computationally infeasible to find two pairs $(x, y), (x, y')$ with $y \neq y'$ that VDF.Verify will accept.

Sequentiality. VDFs must impose a computational delay. Roughly speaking, computing a VDF successfully with non-negligible probability over a uniformly distributed challenge x should be impossible without executing t sequential steps. Formally (adapted from [9]):

▶ **Definition 2** (VDF sequentiality [9]). *A VDF is (p, σ)-*sequential *if for all randomized algorithms $\mathcal{A}_0$ which run in total time $O(\mathsf{poly}(t, \lambda))$, and $\mathcal{A}_1$ which run in parallel time $\sigma(t)$ on at most $p(t)$ processors:*

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

where $\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,t,\mathsf{VDF}}(\lambda)$ is defined in Figure 1.

2.2 Accumulators

▶ **Definition 3** (Accumulator [5, 13]). *Given a data universe U, an* accumulator *is a tuple of algorithms* (Setup, Accumulate, GetMemWit, MemVer) *where:*

Acc.Setup$(\lambda) \to$ **pp** *takes as input λ and outputs public parameters* pp.

Acc.Accumulate$(S) \to A$ *takes as input a set $S \subseteq U$ to be accumulated. It outputs A, an accumulator value for S.*

Acc.GetMemWit$(S, A, x) \to w$ *takes as input a set $S \subseteq U$, an accumulator value A for S, and an element $x \in S$. It outputs a membership witness w for x.*

Acc.MemVer$(A, x, w) \to \{\mathsf{true}, \mathsf{false}\}$ *takes as input an accumulator value A, an element x, and a membership proof (membership witness) w. It outputs* true *if x is included in the accumulated set represented by A and* false *otherwise.*

```
$\mathcal{G}^{\mathrm{acc}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Acc.Setup}(\lambda)$
$S, x, w \xleftarrow{\$} \mathcal{A}(\mathsf{pp})$
$A \leftarrow \mathsf{Acc.Accumulate}(S)$

return
$\mathsf{Acc.MemVer}(A, x, w) \wedge x \notin S$
```

Figure 2 Accumulator security game.

We describe here only the accumulator functionality necessary for our purposes. Accumulators generally also support an incremental Update function to add additional elements to the accumulated set and *dynamic* accumulators support a Delete function to remove elements [13]. Cornucopia does not require either capability; we assume in each run of the protocol the coordinator collects all randomness contributions (the set being accumulated), accumulates them in one batch operation and never deletes.

An accumulator is *correct* if MemVer always accepts for elements included in honestly accumulated sets. An accumulator is *computationally correct* if it is computationally infeasible to find a set such that an honestly generated inclusion proof for an element in that set does not verify. The key security property of an accumulator is that for an honestly generated accumulator value for some set S, it is infeasible to find a membership proof for an element not in S:

▶ **Definition 4** (Accumulator security [13]). *An accumulator* Acc *is* secure *if no PPT adversary* $\mathcal{A}$ *can succeed with non-negligible probability in* $\mathcal{G}^{\mathrm{acc}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$ *as defined in Figure 2.*

A *universal accumulator* [39] also supports non-membership proofs; that is, it supports two additional functions:

$\mathsf{Acc.GetNonMemWit}(S, A, x') \rightarrow w'$ takes as input a set $S \subseteq U$, an accumulator value A for S, and an element $x' \notin S$. It outputs a non-membership witness w' for x'.

$\mathsf{Acc.NonMemVer}(A, x', w') \rightarrow \{\mathsf{true}, \mathsf{false}\}$ takes as input an accumulator value A, an element x', and a non-membership proof (non-membership witness) w'. It outputs true if x' is *not* included in the accumulated set represented by A and false otherwise.

For Cornucopia itself, a universal accumulator is not required as there is no reason for the coordinator to prove to that any value is *not* included. However, in Section 5.6 we show a generic transformation from any universal accumulator to an insertion-secure accumulator.

A universal accumulator is *correct* if, in addition to MemVer accepting for all included elements, NonMemVer accepts for all non-included elements. Security requires that no adversary can find valid membership and non-membership proofs for the same element:

▶ **Definition 5** (Universal accumulator security [39]). *A universal accumulator* Acc *is* secure *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \xleftarrow{\$} \mathsf{Acc.Setup}(\lambda) \\ A, x, w, w' \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\ \mathsf{Acc.MemVer}(A, x, w) \wedge \mathsf{Acc.NonMemVer}(A, x, w') \end{array}\right] \leq \mathsf{negl}(\lambda)$$

2.3 Vector commitments

We present only the functionality of vector commitments necessary for our applications.

▶ **Definition 6** (Vector commitment [15]). *Given a message space $\mathcal{M}$, a* vector commitment *is a tuple of algorithms including:*

KeyGen$(\lambda, s) \rightarrow$ **pp** *takes in the security parameter λ and the size s of the committed vector, and outputs public parameters* pp.

Com$(m_1, \ldots, m_s) \rightarrow C,$ **aux** *takes as input a vector of s messages in $\mathcal{M}$, and outputs a commitment C and some auxiliary information* aux.

Open$(m, i,$ **aux**$) \rightarrow \pi_i$ *takes as input a message $m \in \mathcal{M}$, an index i, and some auxiliary information* aux. *It outputs a proof π_i that the i^{th} component of the committed vector is m.*

Ver$(C, m, i, \pi_i) \rightarrow \{$**true**, **false**$\}$ *takes as input a commitment, a message m, an index i, and a proof that the i^{th} component of the committed vector is m. It outputs* true *if and only if the proof verifies.*

A vector commitment must satisfy *correctness*, which requires that honestly generated proofs for correct components of honestly generated vector commitments verify, as well as *position binding*, which requires that an adversary cannot produce a (possibly maliciously formed) commitment and two proofs of distinct values for the same component.

▶ **Definition 7** (Position binding [15]). *A vector commitment satisfies* position binding *if for all $i \in [s]$ and for all PPT adversaries $\mathcal{A}$:*

$$\Pr\left[\begin{array}{l} \mathsf{pp} \xleftarrow{\$} \mathsf{Acc.Setup}(\lambda) \\ C, m, m', i, \pi_i, \pi_i' \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\ \mathsf{Ver}(C, m, i, \pi_i) \wedge \mathsf{Ver}(C, m', i, \pi_i') \wedge m \neq m' \end{array}\right] \leq \mathsf{negl}(\lambda)$$

3 Timed DRBs: Definitions and Constructions

We first define timed DRBs using a generalized syntax, building on the definitions of [18].[1]

▶ **Definition 8** (Timed DRBs). *A timed DRB protocol is a tuple of algorithms* (Setup, Prepare, Post, Finalize, Verify)*:*

Setup$(\lambda, t) \xrightarrow{\$}$ **pp**: *The setup algorithm can be run once and outputs public parameters* pp *used for multiple protocol runs.*

Prepare(**pp**) $\xrightarrow{\$} r_i$: *The prepare algorithm is run by each participant to produce a randomness contribution r_i. This contribution is submitted during the* contribution phase*, which is bounded in length by the time parameter t.*

Post$(\{r_i\}) \rightarrow (R, \{\pi_i\})$: *The post algorithm is run by a coordinator immediately after the end of the contribution phase, producing a commitment R to all users' contributions and (optionally) a list of user-specific proofs π_i. Typically, this value R will be posted to a public bulletin board, whereas π_i will be made privately available.*

Finalize(**pp**$, R) \rightarrow (\Omega, \pi_\Omega)$: *The finalize algorithm is run after the post algorithm, evaluating a delay function on R to produce a final DRB output Ω and (optionally) a proof π_Ω. It is a deterministic algorithm running in time $(1 + \epsilon)t$ for some small ϵ.*

Verify(**pp**$, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{$**true**, **false**$\}$: *Individual users should verify both the final DRB output Ω as well as that their contribution r_i was correctly included, possibly with the help of an auxiliary user-specific proof π_i.*

[1] Note that our syntax here is specific to one-round timed DRBs. Some timed DRBs such as Bicorn [18] have an optional second communication round.

```
$\mathcal{G}^{\text{indist}}_{\mathcal{A},t,b,\mathsf{DRB}}(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda, t)$
$r_1 \xleftarrow{\$} \mathsf{Prepare}(\mathsf{pp})$
$\alpha_0 \xleftarrow{\$} \mathcal{A}_0(\mathsf{pp})$
$\alpha_1, R, \pi_1 \xleftarrow{\$} \mathcal{A}_1(\alpha_0, r_1)$
$\Omega_0, \pi_0 \leftarrow \mathsf{Finalize}(\mathsf{pp}, R)$
$\Omega_1 \xleftarrow{\$} U$
$b' \xleftarrow{\$} \mathcal{A}_2(\alpha_1, \Omega_b)$

return $b = b'$
$\wedge \mathsf{Verify}(\mathsf{pp}, R, \Omega_0, \pi_0, r_1, \pi_1)$
```

```
$\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda)$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda, t)$
$r_1 \xleftarrow{\$} \mathsf{Prepare}(\mathsf{pp})$
$\alpha_0 \xleftarrow{\$} \mathcal{A}_0(\mathsf{pp})$
$\tilde{\Omega}, \pi_{\tilde{\Omega}}, R, \pi_1 \xleftarrow{\$} \mathcal{A}_1(\alpha_0, r_1)$

return $\mathsf{Verify}(\mathsf{pp}, R, \tilde{\Omega}, \pi_{\tilde{\Omega}}, r_1, \pi_1)$
```

Figure 3 Security games for (p, σ)-indistinguishability (left) and (p, σ)-unpredictability (right).

A timed DRB has the following security properties (shown in Figure 3):

▶ **Definition 9** ((p, σ)-unpredictability)**.** *The (p, σ)-unpredictability game tasks an adversary with predicting the final output Ω exactly, allowing it control of all but a single honest participant (which publishes first). This adversary's computation is broken into two phases. In the precomputation phase, before the adversary sees the honest contribution r_1, it may run an algorithm $\mathcal{A}_0$ that runs in time $\mathsf{poly}(\lambda, t)$. This algorithm outputs some advice string. After seeing r_1, the adversary is limited to running for $\sigma(t)$ steps on at most $p(t)$ parallel processors, exactly like the adversary for VDF sequentiality (Definition 2). The adversary's advantage is:* $\mathsf{Adv}^{\text{unpred}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda) = 1\right]$.

The (p, σ)-unpredictability property only guarantees the DRB output cannot be predicted exactly. We can define a stronger (p, σ)-indistinguishability property in which the adversary must distinguish a DRB output from random, again allowing the adversary control of all-but-one participants:

▶ **Definition 10** ((p, σ)-indistinguishability)**.** *The (p, σ)-indistinguishability game is exactly like the (p, σ)-unpredictability game, except with an extra input bit b. The challenger provides the adversary the genuine output of* Finalize *if $b = 0$ and a random output if $b = 1$. The adversary must, after running for at most $\sigma(t)$ steps on at most $p(t)$ parallel processors, output a guess b' for which output it received. We define the adversary's advantage as:*

$$\mathsf{Adv}^{\text{indist}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda) = \left|\Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,1,\mathsf{DRB}}(\lambda) = 1\right] - \Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,0,\mathsf{DRB}}(\lambda) = 1\right]\right|$$

As observed by Boneh et al. [9], we can convert any timed DRB which satisfies (p, σ)-unpredictability into one with (p, σ)-indistinguishability by applying a random oracle to the output. Our main result (Theorem 14) shows Cornucopia is unpredictable, indistinguishability thus immediately follows in the random oracle model (Corollary 15).

3.1 Unicorn

As a warm-up, we succinctly describe Unicorn [38] as a timed DRB in our framework in Figure 4.[2] Intuitively, Unicorn is secure because every user can check that their value is included in the posted set $\{r_i\}$. A VDF is evaluated on a hash of this set. A single honest

[2] Note that the the original Unicorn proposal used the delay function Sloth, which computes modular square roots modulo a prime. We describe Unicorn here using a modern VDF instead [9].

```
$\underline{\mathsf{Setup}(\lambda, t) \xrightarrow{\$} \mathsf{pp}}$
$\mathsf{pp} \leftarrow \mathsf{VDF.Setup}(\lambda, t)$

$\underline{\mathsf{Prepare}() \xrightarrow{\$} r_i}$
$r_i \overset{\$}{\leftarrow} U$

$\underline{\mathsf{Post}(\{r_i\}) \rightarrow (R, \varnothing)}$
$R \leftarrow \{r_i\}$

$\underline{\mathsf{Finalize}(R) \rightarrow (\Omega, \pi_\Omega)}$
$\Omega, \pi_\Omega \leftarrow \mathsf{VDF.Eval}(H(R))$

$\underline{\mathsf{Verify}(\mathsf{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{\mathsf{true}, \mathsf{false}\}}$
return $r_i \in R \land \mathsf{VDF.Verify}(H(R), \Omega, \pi_\Omega)$
```

```
$\underline{\mathsf{Setup}(\lambda, t) \xrightarrow{\$} \mathsf{pp}}$
$\mathsf{pp} \leftarrow (\mathsf{VDF.Setup}(\lambda, t), \mathsf{Acc.Setup}(\lambda))$

$\underline{\mathsf{Prepare}() \xrightarrow{\$} r_i}$
$r_i \overset{\$}{\leftarrow} U$

$\underline{\mathsf{Post}(\{r_i\}) \rightarrow (R, \{\pi_i\})}$
$R \leftarrow \mathsf{Acc.Accumulate}(\{r_i\})$
$\pi_i \leftarrow \mathsf{Acc.GetMemWit}(\{r_j\}, R, r_i)$

$\underline{\mathsf{Finalize}(R) \rightarrow (\Omega, \pi_\Omega)}$
$\Omega, \pi_\Omega \leftarrow \mathsf{VDF.Eval}(H(R))$

$\underline{\mathsf{Verify}(\mathsf{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{\mathsf{true}, \mathsf{false}\}}$
return $\mathsf{VDF.Verify}(H(R), \Omega, \pi_\Omega)$
$\land$ $\mathsf{Acc.MemVer}(R, r_i, \pi_i)$
```

Figure 4 The Unicorn timed DRB protocol [38] (left) and the Cornucopia protocol (right).

user is enough to ensure this hashed value cannot have been precomputed by the adversary. Unicorn's security is directly implied by our security proof for Cornucopia in Theorem 14, as Unicorn is a special case using the trivial "concatenation accumulator".[3] The primary downside of Unicorn is the fact that $|R| = \Theta(n)$. The goal of Cornucopia is to achieve the same security as Unicorn while storing only $\Theta(1)$ data on the public bulletin board.

3.2 Cornucopia

Cornucopia, shown in Figure 4, improves on Unicorn by having the coordinator accumulate all user contributions into a succinct commitment R using a cryptographic accumulator scheme (see Section 2). Because $|R|$ does not grow with the number of participants, Cornucopia easily scales to many users with constant publishing costs. Our indistinguishability and unpredictability definitions ensure that the protocol is secure as long as a single honest user contributes, so any honest user can be convinced the final result is random as long as they are convinced that their contribution was included.

Note that our Cornucopia presentation and security definitions focus on security against manipulation and not on *liveness*; the coordinator can trivially block individual participants or even prevent the protocol from running at all. In Section 7.1 we revisit this and introduce the multi-coordinator model to ensure liveness even if all-but-one coordinators act maliciously.

4 Cornucopia Security

The security of Cornucopia relies on the adversary's inability to predict the output of the VDF. This also requires that the adversary cannot produce an accumulator value satisfying an honest participant before seeing that participant's randomness contribution. If it were able to do so, it could precompute the output of the VDF applied to this accumulator value and predict the output of the randomness beacon. However, the participant would still receive a valid proof that their contribution was included and believe that the randomness beacon was unpredictable. A trivial attack would be to accumulate the entire data universe, ensuring any user contribution could be proven "included." To formalize this requirement, we define a novel security property for accumulators, called *insertion security*. We then prove that Cornucopia is secure when instantiated with any insertion-secure accumulator.

[3] Lenstra and Wesolowski prove security of Unicorn in a slightly different model [38].

```
$\underline{\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)}$
$\mathsf{pp} \xleftarrow{\$} \mathsf{Acc.Setup}(\lambda)$
$A \leftarrow \mathcal{A}(\mathsf{pp})$
$x \xleftarrow{\$} U$
$w \leftarrow \mathcal{A}(\mathsf{pp}, A, x)$

return $\mathsf{Acc.MemVer}(A, x, w)$
```

Figure 5 Insertion security game.

4.1 Insertion Security

Intuitively, an accumulator is insertion-secure if it is infeasible for any efficient adversary to accumulate a non-negligible fraction of the data universe. We formalize this property using an insertion security game, shown in Figure 5. To win the insertion security game, the adversary must produce an accumulator value A such that it can supply a membership proof for a randomly chosen element with non-negligible probability. Note that the adversary is not limited to producing A via the normal Accumulate function; it can compute A using any procedure at all. Using this game, insertion security is defined as follows:

▶ **Definition 11** (Insertion Security). *An accumulator is* insertion-secure *if for any PPT algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ winning the insertion security game (Figure 5) is negligible:*

$$\Pr\left[\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda) = 1\right] \leq \mathsf{negl}(\lambda)$$

Although insertion security is (to our knowledge) a novel property of accumulators, it turns out that many constructions are naturally insertion-secure, as we will show in Section 5.

Necessity of insertion security. We will show that insecurity security is sufficient for Cornucopia in Theorem 14. We can also show insertion security is *necessary*. To see why, suppose that the underlying accumulator is not insertion-secure. The adversary is therefore able to produce some A such that with noticeable probability, it can efficiently compute a membership proof for a random element with respect to A. The adversarial coordinator precomputes $\Omega = \mathsf{VDF.Eval}(H(A))$ and predicts that this will be the beacon output. The coordinator then accepts randomness contributions from the participants, and in the Post protocol outputs A regardless of the values of these contributions. Now, consider some honest participant. With noticeable probability, the adversary is able to produce a membership proof with respect to A for their randomness contribution. Therefore, this honest participant accepts. However, this breaks security, as the adversary correctly predicted the output Ω. Combined with our proof of Theorem 14, this shows that our definition is tight – insertion security is both necessary and sufficient.

Incomparability with standard accumulator security. We can show that insertion security is incomparable to standard accumulator security (Definition 4). Given any secure accumulator scheme Acc, one can construct an accumulator Acc' which is not insertion-secure, but otherwise satisfies the standard security definitions of an accumulator. One approach is to add a special symbol ϵ which is defined as the accumulation of the entire data universe U. $\mathsf{Acc'.MemVer}(A, x, w)$ is defined to be 1 if $A = \epsilon$ (regardless of the value of x or w), and otherwise is equal to $\mathsf{Acc.MemVer}(A, x, w)$. The scheme Acc' can be used exactly as Acc in normal operation, with the extra property that ϵ is a "shortcut" to computing an accumulation of the entire data universe. We show later in Section 5 that some common schemes such as RSA and bilinear accumulators naturally feature this shortcut.

On the other hand, insertion security does not imply standard accumulator security. Recall that an accumulator is secure if an adversary cannot produce an honestly computed commitment A to a set S, an element $x \notin S$, and a valid membership proof for x with respect to A. Now, consider modifying an insertion-secure accumulator so that for a special element x^*, any witness is accepted; that is, $\mathsf{MemVer}(A, x^*, w)$ outputs true for all A and w. This resulting accumulator is still insertion-secure, as x^* is chosen as the challenge element with only negligible probability; however, it does not satisfy standard accumulator security as it is possible to provide a valid proof for x^* even if it was not in the genuinely accumulated set.

4.2 Security of Cornucopia

Before proving our main result (Theorem 14), we first prove two useful lemmas. The first is that if Cornucopia is constructed using an insertion-secure accumulator, an adversary cannot guess a satisfactory R before seeing the contribution r_1 of the sole honest participant. Insertion security implies that it is difficult to precompute an accumulator value for which one can provide a membership proof of a random element. The second lemma states that if the adversary does not query R to the random oracle in its precomputation phase, it cannot output $\tilde{\Omega} = \mathsf{VDF.Eval}(H(R))$. This is because after the precomputation phase, the adversary is (p, σ)-sequential and therefore cannot evaluate the VDF; thus, to prove this lemma we invoke VDF sequentiality. Together, these lemmas make it straightforward to prove that Cornucopia (CC for short) is secure given any insertion-secure accumulator and secure VDF.

▶ **Lemma 12.** *Let $\mathcal{E}_1$ be the event that $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1$ and $\mathcal{A}_0$ queried R to the random oracle. If CC is instantiated with an insertion-secure accumulator, then* $\Pr[\mathcal{E}_1] \le \mathsf{negl}(\lambda)$.

Proof. Suppose for the sake of contradiction that for some constant $c > 0$,

$$\Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1 \wedge \mathcal{A}_0 \text{ queried } R \text{ to the random oracle}\right] \ge \frac{1}{\lambda^c}$$

We define an adversary $\mathcal{B}$ that breaks insertion security of the accumulator scheme by simulating the challenger in $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}$ and using $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. $\mathcal{B}$ first receives $\mathsf{Acc.pp}$ in $\mathcal{G}^{\text{insert}}_{\mathcal{B},\mathsf{Acc}}(\lambda)$. It samples $\mathsf{VDF.pp} \leftarrow \mathsf{VDF.Setup}(\lambda, t)$ and passes $\mathsf{pp} = (\mathsf{Acc.pp}, \mathsf{VDF.pp})$ to $\mathcal{A}_0$. $\mathcal{B}$ simulates the challenger in $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda)$ and records the queries $q_1, \dots, q_k$ that $\mathcal{A}_0$ makes to the random oracle. $\mathcal{B}$ also receives α_0 as the output of $\mathcal{A}_0$. $\mathcal{B}$ then chooses some query q_i uniformly at random from the queries made by $\mathcal{A}_0$ and outputs $A = q_i$ as its accumulator value in $\mathcal{G}^{\text{insert}}_{\mathcal{B},\mathsf{Acc}}(\lambda)$. $\mathcal{B}$ then receives x from the challenger in $\mathcal{G}^{\text{insert}}_{\mathcal{B},\mathsf{Acc}}(\lambda)$, and it continues simulating the $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda)$ challenger by passing α_0 and $r_1 = x$ to $\mathcal{A}_1$. $\mathcal{B}$ receives $(\tilde{\Omega}, R, w_1)$ as the output of $\mathcal{A}_1$.

Since $\mathcal{A}$ succeeds with at least probability $\frac{1}{\lambda^c}$,

$$\Pr[\mathsf{MemVer}(R, x, w_1) = \mathsf{true} \wedge \mathcal{A}_0 \text{ queried } R \text{ to the random oracle}] \ge \frac{1}{\lambda^c}$$

Let $q(\lambda)$ be some polynomial upper bounding the number of queries that $\mathcal{A}_0$ makes to the random oracle; this polynomial must exist since $\mathcal{A}_0$ runs in polynomial time. Since $\mathcal{B}$'s random choice of q_i is independent of $\mathcal{A}$, $\Pr[\mathsf{MemVer}(R, x, w_1) = \mathsf{true} \wedge A = R] \ge \frac{1}{\lambda^c} \cdot \frac{1}{q(\lambda)}$ which is non-negligible. Thus, with non-negligible probability, $\mathcal{G}^{\text{insert}}_{\mathcal{B},\mathsf{Acc}}(\lambda) = 1$. ◀

▶ **Lemma 13.** *Let $\mathcal{E}_2$ be the event that $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1$ and $\mathcal{A}_0$ did not query R to the random oracle. If CC is instantiated with an insertion-secure accumulator and a (p, σ)-sequential VDF, then* $\Pr[\mathcal{E}_2] \le \mathsf{negl}(\lambda)$.

Proof. Suppose for the sake of contradiction that for some constant $c > 0$,

$$\Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1 \wedge \mathcal{A}_0 \text{ did not query } R \text{ to the random oracle}\right] \geq \frac{1}{\lambda^c}$$

We define an adversary $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$ that breaks (p, σ)-sequentiality of the VDF by simulating the challenger and random oracle in $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}$ and using $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$. When $\mathcal{A}$ evaluates the hash function it must query $\mathcal{B}$. $\mathcal{B}$ responds in a way that is indistinguishable (to $\mathcal{A}$) from a random function.

$\mathcal{B}_0$ first receives $(\lambda, \mathsf{VDF.pp}, t)$ from the VDF challenger in $\mathcal{G}^{\text{sequential}}_{\mathcal{B}_0,\mathcal{B}_1,t,\mathsf{VDF}}(\lambda)$. $\mathcal{B}_0$ samples $\mathsf{Acc.pp} \leftarrow \mathsf{Acc.Setup}(\lambda)$ and passes $\mathsf{pp} = (\mathsf{VDF.pp}, \mathsf{Acc.pp})$ to $\mathcal{A}_0$. $\mathcal{B}_0$ answers $\mathcal{A}_0$'s random oracle queries using uniformly random values. It records these queries and their responses in a list Q. If any query is repeated, $\mathcal{B}_0$ answers consistently with its previous response in Q. $\mathcal{A}_0$ outputs an advice string α_0, which $\mathcal{B}_0$ outputs as part of its advice string $\alpha = (\alpha_0, Q)$.

Now, the VDF challenger samples a random input x which is passed to $\mathcal{B}_1$ along with $\mathsf{VDF.pp}$ and α. $\mathcal{B}_1$ passes α_0 and a randomly-generated value $r_1 \xleftarrow{\$} \mathsf{Prepare}(\mathsf{pp})$ to $\mathcal{A}_1$. $\mathcal{B}_1$ then simulates the random oracle for $\mathcal{A}_1$, with one key modification: $\mathcal{B}_1$ chooses an index $i \leq p(t) \cdot t$ uniformly at random[4] and answers $\mathcal{A}_1$'s i^{th} random oracle query q_i with x (provided that q_i has not been previously queried, otherwise it responds with the appropriate value from Q). It answers any future repeated queries q_i similarly. For all other queries, $\mathcal{B}_1$ answers randomly the first time and then consistent with its stored responses in Q. When $\mathcal{A}_1$ outputs $(\tilde{\Omega}, R, w_1)$, $\mathcal{B}_1$ outputs $\tilde{\Omega}$.

$\mathcal{B}$ properly simulates the random oracle. Since x is a uniformly random value and all other queries receive random responses, $\mathcal{B}_1$ does not change the output distribution of the random oracle and hence does not affect $\mathcal{A}_1$'s behavior.

If $\mathcal{A}$ succeeds, $\mathcal{B}$ succeeds with non-negligible probability. We now argue that if $\mathcal{A}$ wins $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}$, $\mathcal{B}$ wins $\mathcal{G}^{\text{sequential}}_{\mathcal{B}_0,\mathcal{B}_1,t,\mathsf{VDF}}(\lambda)$ with non-negligible probability. First, recall that if $\mathcal{A}$ wins, $\mathsf{DRB.Verify}$ holds. By uniqueness of the VDF, the probability that $\mathcal{A}_1$ outputs a proof π_Ω such that $\mathsf{VDF.Verify}(\mathsf{VDF.pp}, H(R), \tilde{\Omega}, \pi_\Omega) = 1$ yet $\tilde{\Omega} \neq \mathsf{VDF.Eval}(H(R))$ is negligible. Thus, since $\mathsf{DRB.Verify}$ holds, $\mathcal{A}_1$ must have output $\tilde{\Omega} = \mathsf{VDF.Eval}(H(R))$.

We now show that the fact that $\mathcal{A}_1$ outputs $\mathsf{VDF.Eval}(H(R))$ implies that $\mathcal{B}$ breaks (p, σ)-sequentiality of the VDF. Because the index i of the query to be replaced was chosen uniformly and independently of $\mathcal{A}_1$, q_i was chosen to be the first instance that R was queried by $\mathcal{A}_1$ with probability at least $\frac{1}{p(t) \cdot t}$. Since $\mathcal{A}_0$ did not query R, we can indeed make this replacement. Therefore, with non-negligible probability $\mathcal{B}_1$ simulates the random oracle to answer R with x, and $\tilde{\Omega} = \mathsf{VDF.Eval}(x)$ as desired.

Thus, for $(\tilde{\Omega}, R, w_1)$ output by $\mathcal{A}_1$, it holds that

$$\Pr\left[\tilde{\Omega} = \mathsf{VDF.Eval}(H(R)) \wedge \mathcal{A}_0 \text{ did not query } R \text{ to the RO}\right] \geq \frac{1}{\lambda^c}$$

In the above, we assumed that $\mathcal{A}_1$ queried R to the random oracle. If $\mathcal{A}_1$ did not query R to the random oracle, it has anyways succeeded in computing the VDF output on $H(R)$ which is a random value and identically distributed to x. ◀

Given these lemmas, we can now succinctly prove our main result:

[4] We use $p(t) \cdot t$ as a generous upper bound on the number of random oracle queries made by $\mathcal{A}_1$, if every processor queries the oracle in every time step.

▶ **Theorem 14** (Unpredictability of Cornucopia)**.** *Cornucopia is (p, σ)-unpredictable when instantiated with an insertion-secure accumulator, a (p, σ)-sequential VDF, and a hash function modeled as a random oracle.*

Proof. Let $\mathcal{E}_1$ be the event that $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1$ and $\mathcal{A}_0$ queried R to the random oracle. Let $\mathcal{E}_2$ be the event that $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1$ and $\mathcal{A}_0$ did not query R to the random oracle.

Observe that $\Pr[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1] = \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2]$. By Lemma 12, $\Pr[\mathcal{E}_1] \leq \mathsf{negl}(\lambda)$. By Lemma 13, $\Pr[\mathcal{E}_2] \leq \mathsf{negl}(\lambda)$. Therefore, $\Pr[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}(\lambda) = 1] \leq \mathsf{negl}(\lambda)$. ◀

▶ **Corollary 15.** *Cornucopia is (p, σ)-indistinguishable when a random oracle is applied to its output.*

5 Insertion-secure accumulators

We now turn to the question of instantiating accumulators satisfying insertion security (Definition 11).

5.1 Accumulators without insertion security

Recall from Section 4.1 that one can construct accumulators that have a shortcut ϵ that accumulates the entire data universe. RSA accumulators naturally feature such a shortcut: $\epsilon = 1$. A valid membership witness for any x is $w = 1$, since $w^x = 1^x = 1$. Although we will prove RSA accumulators can easily be made insertion-secure by disallowing an accumulator value of 1, technically they are not insertion-secure as commonly specified. Bilinear accumulators have the same shortcut, which we remove with the same modification.

A second example, potentially of practical interest, is a *range accumulator*. A range accumulator can be defined from any accumulator scheme and for any data universe with a known total ordering (for example, any fixed subset of the integers such as $\{0, 1\}^k$). With a range accumulator, the value $H(x, y)$ can be accumulated, which is interpreted as adding a range $[x, y]$ (the value $H(x, x)$ can be accumulated to add a single element x). Given any value z, proving membership can be achieved by providing a witness $w' = (w, x, y)$ where $w = \mathsf{Acc.GetMemWit}(S, A, H(x, y))$ for $x \leq z \leq y$. This concept is quite natural and efficient, though it is also trivially not insertion-secure: an adversary can win $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$ with probability 1 by accumulating the value $H(x_{\min}, x_{\max})$ for the smallest and largest data elements in U, effectively accumulating the entire data universe in constant time.[5]

5.2 Merkle trees

▶ **Lemma 16.** *A Merkle tree of bounded depth $k = \mathsf{poly}(n)$ is insertion-secure in the random oracle model.*

Proof. We work in the random oracle model, supposing that the Merkle tree uses a random oracle $\mathcal{O} : \{0, 1\}^{2n} \to \{0, 1\}^n$. Let A be the accumulator output by an adversary $\mathcal{A}$ in $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$. We show that for a uniform $x \in \{0, 1\}^n$, the adversary can provide a verifying witness $w = (w_1, \ldots, w_k)$ for x with only negligible probability. For a verifying witness, it must hold that $\mathcal{O}(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x))) = A$. We'll show that with overwhelming probability (over choice of x), no query to $\mathcal{O}$ involved in the witness verification was made by the adversary in step 2 of $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$.

[5] The adversary can in fact win with non-negligible probability by accumulating any range whose size is a constant fraction of $|U|$.

This can be shown by induction. Let $a_1, \ldots, a_\ell$ be the adversary's queries to the random oracle in step 2. Let $b_1, \ldots, b_k$ be the queries to the random oracle in the Merkle membership proof verification; that is, $b_i = w_i || \mathcal{O}(w_{i-1} || \ldots)$. Let $p(\lambda)$ be a polynomial upper bound on the total number of queries made by the adversary to the random oracle throughout the game. Observe first that $\Pr[b_1 = a_j \text{ for some j}] = \frac{\ell}{2^\lambda}$ since $b_1 = w_1 || x$ and x is chosen at random. Assume that the probability that b_i is equal to any a_j is at most $\frac{i\ell \cdot p(\lambda)}{2^\lambda}$. If this event does not occur, then $\mathcal{O}(b_{i+1}) = \mathcal{O}(w_{i+1} || \mathcal{O}(b_i))$ is a freshly random value, and the probability that $b_{i+1} = a_j$ for any j is at most $\frac{\ell \cdot p(\lambda)}{2^\lambda}$ (since $\mathcal{A}$ can try up to $p(\lambda)$ values for w_{i+1}).

$$\begin{aligned}
\Pr\left[b_{i+1} = a_j \text{ for some } j\right] &\leq \frac{\ell \cdot p(\lambda)}{2^\lambda} \Pr\left[b_i \neq a_j \text{ for all } j\right] \\
&\quad + \Pr\left[b_i = a_j \text{ for some } j\right] \\
&\leq \frac{\ell \cdot p(\lambda)}{2^\lambda} + \frac{i\ell \cdot p(\lambda)}{2^\lambda} \\
&= \frac{(i+1)\ell \cdot p(\lambda)}{2^\lambda}
\end{aligned}$$

since $\Pr[b_i = a_j \text{ for some } j] \leq \frac{i\ell \cdot p(\lambda)}{2^\lambda}$ by assumption. Therefore, the probability that any of the (polynomially bounded) k queries involved in witness verification was queried in step 2 is at most $\frac{k\ell \cdot p(\lambda)}{2^\lambda} \leq \mathsf{negl}(\lambda)$.

In order for witness verification to pass, the last query must match the root; that is, $\mathcal{O}(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x))) = A$. Since the above argument shows that $(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x)))$ was never queried in step 2, at the end of which $\mathcal{A}$ outputs A, for each choice of w_k, $\mathcal{O}(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x)))$ is a uniformly random value independent of A and equals A with only negligible probability. ◀

5.3 RSA accumulators

In a standard RSA accumulator [13, 40], $\mathsf{Setup}(\lambda)$ generates a random group of unknown order and a generator g for this group using some group generation algorithm $\mathsf{GenGroup}$. The data universe is Π_λ, the set of all λ-bit primes. The accumulator value for a set S is $A = g^{\prod_{x \in S} x}$, and the witness w for an element x for the value A is $w = g^{\prod_{x' \in S \setminus \{x\}} x'} = A^{1/x}$. $\mathsf{Add}(A_t, x)$ outputs $A_{t+1} = A_t^x$. Thus, the accumulator value for a set S can be obtained by starting with the value $A_0 = 1$ and adding each $x_i \in S$ to $A_{i=1}$ to obtain A_i, repeating until we reach $A_{|S|}$. $\mathsf{UpdWit}(A_t, x, w_t')$ outputs $w_{t+1}' = (w_t')^x$. $\mathsf{MemVer}(A, x, w)$ outputs 1 if and only if $w^x = A$. A non-membership witness for x with respect to $A = g^{\prod_{s \in S} s}$ is $\{a, B\}$ where a and b are Bézout coefficients for $(x, \prod_{s \in S} s)$, and $B = g^b$. $\mathsf{NonMemVer}(A, \{a, B\}, x)$ outputs 1 if and only if $A^a B^x = g$.

To make RSA accumulators insertion-secure, we add a second condition to $\mathsf{MemVer}(A, x, w)$: It now outputs 1 if and only if $w^x = A$ *and* $A \neq 1$. Note that our requirement that $A \neq 1$ is necessary to reduce insertion security to the Adaptive Root Assumption.

▶ **Assumption 17** (Adaptive Root Assumption [10]).

$$\Pr\left[\begin{array}{cr@{\;}l} & \mathbb{G} & \xleftarrow{\$} \mathsf{GenGroup}(\lambda) \\ & (v, st) & \leftarrow \mathcal{A}_0(\mathbb{G}) \\ u^l = v \neq 1 : & l & \xleftarrow{\$} \Pi_\lambda = \mathit{Primes}(\lambda) \\ & u & \leftarrow \mathcal{A}_1(v, l, st) \end{array}\right] \leq \mathsf{negl}(\lambda)$$

▶ **Lemma 18.** *Suppose a standard RSA accumulator is modified so that the algorithm* $\mathsf{MemVer}(A, x, w)$ *outputs 1 if and only if* $w^x = A$ *and* $A \neq 1$. *The modified RSA accumulator is insertion-secure if the Adaptive Root Assumption holds for the group generation algorithm* $\mathsf{GenGroup}$.

Proof. Suppose that there exists a PPT adversary $\mathcal{A}$ that wins $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$ with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$ when the data universe is Π_λ, the set of all λ-bit primes. We construct a pair of adversaries $\mathcal{B}_0, \mathcal{B}_1$ that uses $\mathcal{A}$ to break the Adaptive Root Assumption. $\mathcal{B}_0$ draws $\mathbb{G} \xleftarrow{\$} \mathsf{GenGroup}(\lambda)$. $\mathcal{B}_0$ passes $\mathbb{G}$ to $\mathcal{A}$ and obtains an accumulator value A. $\mathcal{B}_0$ outputs $v = A$ and st as its current state. $\mathcal{B}_1$ draws a random $l \xleftarrow{\$} \Pi_\lambda$ and passes $x = l$ to $\mathcal{A}$. $\mathcal{A}$ outputs an alleged witness w_x which $\mathcal{B}_1$ outputs directly as u in the Adaptive Root Game.

Recall that if $\mathcal{A}$ wins $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$, it means that $\mathsf{MemVer}(A, x, w_x) = \mathsf{true}$. For RSA accumulators, $\mathsf{MemVer}(A, x, w_x) = \mathsf{true}$ if and only if $(w_x)^x = A$ and $A \neq 1$. This implies that $u^l = v$ where $v \neq 1$, and $(\mathcal{B}_0, \mathcal{B}_1)$ win the Adaptive Root Game. Since $\mathcal{A}$ wins with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, $(\mathcal{B}_0, \mathcal{B}_1)$ win with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, violating the Adaptive Root Assumption. ◀

▶ **Corollary 19.** *The modified RSA accumulator is insertion-secure in the Algebraic Group Model (AGM), since the Adaptive Root Assumption holds in the AGM [25].*

5.4 Bilinear accumulators

We show that bilinear accumulators [42, 53] with a small modification are insertion-secure in the AGM, under the Bilinear q-Strong Diffie-Hellman Assumption. The standard bilinear accumulator was defined by Nguyen [42], and we follow [44] in its presentation. Let $\mathbb{G}, \mathcal{G}$ be cyclic multiplicative groups of prime order p, and let $e : \mathbb{G} \times \mathbb{G} \to \mathcal{G}$ be a bilinear pairing. Let $s \xleftarrow{\$} \mathbb{Z}_p^*$, and let g be a generator of $\mathbb{G}$. Let $\mathsf{srs} = [g, g^s, \ldots, g^{s^q}]$ be the structured reference string, where q is an (polynomial in λ) upper bound on the number of accumulated elements. The public parameters are $(p, \mathbb{G}, \mathcal{G}, e, g, \mathsf{srs})$. Note that s must be kept secret even to the coordinator, and therefore a trusted setup is required.

This accumulator has data universe $U = \mathbb{Z}_p^* \setminus \{-s\}$. To accumulate a set $X \subset U$, where $|X| \leq q$, one computes $A = g^{\prod_{x_i \in X}(x_i+s)}$. The witness for an element $x \in X$ is $W = g^{\prod_{x_i \in (X \setminus \{x\})}(x_i+s)}$. To verify a witness, one checks that $e(W, g^{s+x}) = e(A, g)$. To make this accumulator insertion-secure, we also check that $A \neq 1$.

In the Algebraic Group Model (AGM) [28], the adversary is constrained to perform only algebraic operations within the given group. That is, the adversary is given some group elements as input, and for any element that it outputs, it must provide a description of the operations used to obtain that element. In our setting, the algebraic adversary is given as input $[1, g, g^s, \ldots, g^{s^q}]$. For any group element h that the adversary outputs, it must provide a scalar vector $v \in \mathbb{Z}_p^*$ such that $h = \prod_{i=0}^{q} g^{v_i \cdot s^i}$. We refer the reader to [28, 29] for a more formal definition. Observe that the v_i's can be interpreted as the coefficients of a polynomial of degree q evaluated at s. We use this interpretation in the following proof.

▶ **Assumption 20** (q-Discrete Logarithm Assumption (q-DLOG) [28]). *The q-DLOG assumption holds in a group* $\mathbb{G}$ *if for every p.p.t. adversary* $\mathcal{A}$,

$$\Pr_{s \leftarrow \mathbb{Z}_p^*}\left[\mathcal{A}\left(g, g^s, \ldots, g^{s^q}\right) \to s\right] \leq \mathsf{negl}(\lambda).$$

▶ **Lemma 21.** *The bilinear accumulator of [42] is insertion-secure in the AGM, under the q-DLOG Assumption.*

Proof. Let $\mathcal{A}$ be an algebraic adversary that takes srs as input and outputs A such that with non-negligible probability, $\mathcal{A}$ can produce a verifying witness W for a randomly chosen $x \in \mathbb{Z}_p^*$. Since $\mathcal{A}$ is algebraic, it must output vectors which we interpret as polynomials $\alpha(S), w(S)$ of degree at most q such that $A = g^{\alpha(s)}$ and $W = g^{w(s)}$. Since the witness verifies, $e(W, g)^{(s+x)} = e(g^{\alpha(s)}, g)$; that is, $e(g,g)^{w(s)(s+x)} = e(g,g)^{\alpha(s)}$. Furthermore, $\alpha(S)$ is a nonzero polynomial since verification requires that $A \neq 1$.

Observe that since x is chosen randomly from an exponentially large set, and α is a nonzero polynomial of polynomially bounded degree, $(S+x)$ divides $\alpha(S)$ with only negligible probability by the Schwartz-Zippel lemma. Therefore, $w(S)(S+x) - \alpha(S)$ is a nonzero polynomial that has s as a root. The adversary can factor $w(S)(S+x) - \alpha(S)$ in polynomial time to find s. ◀

5.5 From vector commitments

Vector commitments (VCs) [15] can be used to construct an insertion-secure accumulator for sets of bounded size $\leq k$ for any k polynomial in λ. Let the message space $\mathcal{M}$ underlying our VC have size exponential in λ, and assume there is some total ordering over $\mathcal{M}$. To accumulate a set $S \subseteq \mathcal{M}$, we order this set to obtain a vector and commit to this vector. The witness for an element $x \in S$ is an index $i \leq k$ and a VC opening proof for that index. To verify this witness, one verifies the opening proof. This scheme is detailed below:

Setup(λ): Output $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(\lambda)$.
Accumulate(S): Interpret S as an ordered list $s_1, \ldots, s_{|S|}$, and let $v = [s_1, \ldots, s_{|S|}, 0, \ldots, 0]$ be a vector of length k. Compute $C, \mathsf{aux} \leftarrow \mathsf{VC.Commit}(v)$.
GetMemWit(S, A, x): Compute C, aux from S as above. Let i be such that $x = s_i$. Compute $\pi_i \leftarrow \mathsf{VC.Open}(x, i, \mathsf{aux})$ and output (i, π_i).
MemVer($A, x, (i, \pi_i)$): Output $\mathsf{VC.Ver}(A, x, i, \pi_i)$.

Position binding of vector commitments says that it is infeasible for a PPT adversary to produce *any* (possibly maliciously-generated) A, distinct values x, x', an index i, and accepting proofs π_i, π_i' that the vector committed to by A has x and x' respectively as its i^{th} component. We prove insertion security by showing that an adversary that breaks insertion security of this accumulator can be used to break position binding of the underlying VC scheme.

▶ **Theorem 22.** *When constructed with a vector commitment over an exponentially large data universe, this accumulator scheme is insertion-secure.*

Proof. Suppose that $\Pr\left[\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda) = 1\right]$ is non-negligible. Let $\mathcal{E}_i$ denote the event that $\mathcal{A}$ outputs a proof for index i. Then there must be some accumulator A and index i such that

$$\Pr_{\substack{\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda) \\ A \leftarrow \mathcal{A}(\mathsf{pp})}}\left[\Pr\left[\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda) = 1 \wedge \mathcal{E}_i \;\middle|\; \mathsf{pp}, A\right] \geq \frac{1}{\lambda^{c_1}}\right] \geq \frac{1}{\lambda^{c_2}}$$

for some constants $c_1, c_2 > 0$.

Consider drawing $\mathsf{pp} \leftarrow \mathsf{Setup}(\lambda)$ and running $\mathcal{A}(\mathsf{pp})$ to obtain A. As stated above, with non-negligible probability, there exists some i such that with non-negligible probability given this choice of pp, A the adversary produces a verifying proof for index i. Consider running $\mathcal{A}$ twice from this point, for two independently drawn $x_1, x_2 \leftarrow U$. With probability at least $\frac{1}{\lambda^{2c_1}}$, $\mathcal{A}$ produces verifying opening proofs π_1, π_2 that the i^{th} index of the committed vector equals x_1 and x_2 respectively. Since U is exponentially large, $x_1 \neq x_2$ with overwhelming probability. Therefore, we have found a vector commitment A and proofs π_1, π_2 that the same component takes on two distinct values, contradicting position binding of the vector commitment. ◀

5.6 From generic universal accumulators

Finally, we show how to construct an insertion-secure accumulator Acc' from any universal accumulator Acc. The core idea is to map each element x to two pseudorandom sets (S_x^+, S_x^-), each a subset of the data universe U. Proving membership of x for Acc' requires showing *inclusion* of all elements of S_x^+ in Acc and *exclusion* of all elements of S_x^- in Acc. Intuitively, breaking insertion security by accumulating the entire data universe in Acc does not work because it will make the required non-membership proofs impossible. The best attacker strategy is to accumulate a random subset of half the elements of U, but this will mean that each item in S_x^+ is wrongly excluded with probability $\frac{1}{2}$ and each item in S_x^- is wrongly included with probability $\frac{1}{2}$. By setting ensuring the sizes of S_x^+, S_x^-, we can amplify security to ensure such an adversary has only a negligible probability of correctly showing inclusion of a random element.

In more detail, let Acc be a universal accumulator scheme for data universe U. Here, we let the data universe for Acc' be $U' = \{0,1\}^\lambda$. Let $H : [\lambda] \times U' \to U$ be a hash function that we will model as a random oracle. For any $x \in U'$, let $S_x^+ := \{y \ : \ H(i,x) = y \text{ for } i \in [\frac{\lambda}{2}]\}$, and let $S_x^- := \{y \ : \ H(i,x) = y \text{ for } i \in \{(\frac{\lambda}{2}+1), \ldots, \lambda\}\}$ (assume for convenience that λ is even). We specify the functions of Acc' as follows:

Setup: uses the same setup function as Acc.
Accumulate(S′): Let $S = \bigcup_{x \in S'} S_x^+$. Outputs $A = \mathsf{Acc.Accumulate}(S)$.
GetMemWit(S′, A, x): Outputs a vector of witnesses $\mathbf{w}$ of length λ where:
- For $i \leq \frac{\lambda}{2}$, $w_i = \mathsf{Acc.GetMemWit}(S, A, H(i,x))$ is a membership proof for $H(i,x)$
- For $i > \frac{\lambda}{2}$, $w_i = \mathsf{Acc.GetNonMemWit}(S, A, H(i,x))$ is a non-membership proof for $H(i,x)$

MemVer(A, x, w): Outputs true if and only if the following holds for all $i \in [\lambda]$:
- For $i \leq \frac{\lambda}{2}$, $\mathsf{Acc.MemVer}(A, H(i,x), w_i) = \mathsf{true}$.
- For $i > \frac{\lambda}{2}$, $\mathsf{Acc.NonMemVer}(A, H(i,x), w_i) = \mathsf{true}$.

▶ **Lemma 23.** *If* Acc *is a secure universal accumulator and* H *is modeled as a random oracle,* Acc' *is insertion-secure.*

Proof. Suppose for the sake of contradiction that Acc' is not insertion-secure, and let $\mathcal{A}$ be an adversary that wins the insertion game with probability at least $\frac{1}{\lambda^c}$ for some constant $c > 0$, conditioned on the event that it does not query x before it outputs A. (Since $\mathcal{A}$ is polynomially-bounded, this event fails to occur with only negligible probability). Thus, treating H as a random oracle, $H(x)$ is a λ-length tuple of truly random independent values $y_i \in U$, where $y_1, \ldots, y_{\frac{\lambda}{2}}$ should be included, and $y_{\frac{\lambda}{2}+1}, \ldots, y_\lambda$ should be excluded.

Equivalently, we can think of drawing $\mathbf{y} = y_1, \ldots, y_\lambda$ (uniform and i.i.d. from U) and subsequently drawing a uniformly random vector $\mathbf{b}$ of Hamming weight $\frac{\lambda}{2}$, where y_i should be included if and only if $b_i = 1$.

By an averaging argument, we must have that for a non-negligible fraction of $\mathbf{y} \in X$, $\mathcal{A}$ succeeds with non-negligible probability over subsequent choice of $\mathbf{b} \in \{0,1\}^\lambda$. Let $\mathcal{E}[A, \mathbf{y}, \mathbf{b}, \mathbf{w}]$ denote the event that $\mathsf{Acc.MemVer}(A, y_i, w_i) = \mathsf{true}$ for all i such that $b_i = 1$, and $\mathsf{Acc.NonMemVer}(A, y_i, w_i) = \mathsf{true}$ for all i such that $b_i = 0$. The success of $\mathcal{A}$ in $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}'}(\lambda)$ implies that $\mathcal{E}[A, \mathbf{y}, \mathbf{b}, \mathbf{w}]$ occurs for its choice of A and $\mathbf{w}$, and the random choice of $\mathbf{y}, \mathbf{b}$. Thus,

$$\Pr_{\substack{\mathsf{pp} \xleftarrow{\$} \mathsf{Setup}(\lambda) \\ \mathbf{y}}} \left[\begin{array}{c} \mathcal{A} \text{ outputs } A \text{ such that} \\ \Pr_{\mathbf{b}}\left[\mathbf{w} \leftarrow \mathcal{A} \wedge \mathcal{E}[A, \mathbf{y}, \mathbf{b}, \mathbf{w}]\right] \geq \frac{1}{\lambda^c} \end{array} \right] \geq \frac{1}{\lambda^c}$$

We now construct an adversary $\mathcal{B}$ that breaks universal security of Acc by producing an accumulator value, an element, and both membership and non-membership proofs for that element. Let $\mathcal{B}$ first generate setup parameters and run $\mathcal{A}$ on these parameters to obtain an accumulator value A. Let $\mathcal{B}$ choose $\mathbf{y}$ as above and $\mathbf{b_1}, \mathbf{b_2}$ uniformly random vectors of Hamming weight $\frac{\lambda}{2}$. $\mathcal{B}$ runs $\mathcal{A}$ on inputs $(\mathbf{y}, \mathbf{b_1})$ and $(\mathbf{y}, \mathbf{b_2})$ to obtain $\mathbf{w_1}$ and $\mathbf{w_2}$ respectively. With probability at least $\frac{1}{\lambda^c}$, $\mathcal{B}$ chose pp and $\mathbf{y}$ such that $\Pr_{\mathbf{b}}\left[\mathbf{w} \leftarrow \mathcal{A} \wedge \mathcal{E}[A, \mathbf{y}, \mathbf{b}, \mathbf{w}]\right] \geq \frac{1}{\lambda^c}$. In this event, the probability that both $\mathbf{w_1}$ and $\mathbf{w_2}$ verify is at least $\frac{1}{\lambda^{2c}}$. As $\mathbf{b_1} = \mathbf{b_2}$ with only negligible probability (since $\binom{n}{n/2} \geq 2^{n/2}$), with overwhelming probability there is some i such that $(b_1)_i \neq (b_2)_i$. However, we have (without loss of generality) both that $\mathsf{Acc.MemVer}(A, y_i, (w_1)_i) = \mathsf{true}$ and $\mathsf{Acc.NonMemVer}(A, y_i, (w_2)_i) = \mathsf{true}$. This happens with probability at least $\frac{1}{\lambda^c} \cdot \frac{1}{\lambda^{2c}} \cdot \left(1 - \frac{1}{2^\lambda}\right)$, which is non-negligible. This contradicts universal security of Acc. ◀

Correctness. Accumulators typically require *correctness*, which says that given an honestly-generated accumulator value for a set, honestly-generated membership proofs for elements in that set should verify under MemVer; similarly, honestly-generated non-membership proofs for elements not in that set should verify under $\mathsf{NonMemVer}$. We note that Acc' has only *computational* correctness, since there may be some x_1, x_2 for which the same y is included in $S^+_{x_1}$ and $S^-_{x_2}$. This is problematic, since the membership proofs for x_1, x_2 would require a membership proof *and* a non-membership proof for y (with respect to Acc), which should be difficult by security of Acc, and hence x_1 and x_2 cannot both be included in the accumulator. In Cornucopia, if one user chose x_1 and another user chose x_2, the coordinator could not satisfy both users.

Fortunately, collision resistance of H ensures that actually finding such x_1, x_2 is computationally hard: finding x_1, x_2 such that $y \in S^+_{x_1}$ and $y \in S^-_{x_2}$ would involve finding $i_1 \neq i_2$ such that $y = H(i_1, x_1) = H(i_2, x_2)$, which yields a collision of H. Computational correctness is sufficient for use in Cornucopia (and most other applications), as polynomially-bounded users would not be able to find x_1 and x_2 resulting in the above issue.

6 Efficiency comparison of accumulator constructions

Cornucopia can be constructed from any insertion-secure accumulator. In Table 1 we compare efficiency trade-offs between Merkle trees, RSA accumulators, bilinear accumulators, and a construction from a vector commitment called Hyperproofs. All of these schemes require only $O(1)$ space on the public bulletin board, regardless of the number of participants, though the concrete size varies. No accumulator construction offers obviously superior performance, each offers different trade-offs which might be attractive for different practical applications. For very large deployments (e.g. millions or billions of users) the performance bottleneck is likely inclusion proof generation by the coordinator.

Table 1 Comparison of accumulator options for Cornucopia, at a security level of $\lambda = 128$ bits. Witness generation time is the time required to compute all n witnesses.
†RSA accumulators can be instantiated using class groups [41], which do not require trusted setup. We report numbers here for the classic RSA group $\mathbb{Z}_N^*$.

Scheme	Trusted setup?	|commitment| (bytes)	Witness size (asymp.)	Witness size (bytes)	|Public params| (asymp.)	Witness gen. time (asymp.)
Merkle tree	no	32	$O(\log n)$	$32 \cdot \lceil \log n \rceil$	$O(1)$	$O(n \log n)$
RSA Accumulator	yes†	384	$O(1)$	384	$O(1)$	$O(n^2)$
Bilinear Accumulator	yes	48	$O(1)$	48	$O(n)$	$O(n \log n)$
Hyperproofs [52]	yes	48	$O(\log n)$	$48 \cdot \lceil \log n \rceil$	$O(n)$	$O(n \log n)$

Merkle trees. Merkle trees are optimal in terms of the commitment size (32 bytes), require no trusted setup or public parameters and are naturally post-quantum secure. They are also the most efficient for the coordinator to compute witnesses, both in asymptotic and concrete terms. The only downside of Merkle trees is logarithmic witness sizes. Overall, we expect this to be the simplest and best approach for many applications, unless clients are extremely bandwidth-limited or the number of users is very large.

RSA accumulators. By contrast, RSA accumulators offer constant witness sizes, potentially offering the capability to scale to more users without imposing extra bandwidth requirements on clients. However, we note that the large size of RSA groups considered to offer 128-bit security (3072 bit moduli) means that Merkle proofs are shorter in practice with fewer than $\approx 2^{12}$ users participating. RSA proofs also require computing modular exponentiation on large integers. This is relatively poorly supported by today's smart contract platforms like EVM, but we observe that these only ever need to be verified off-chain by users. Still, proof verification is expected to be roughly an order of magnitude slower than Merkle proofs which only require hashing (though both are very efficient in concrete terms).

Furthermore, the size of the public commitment is over 10 times larger than for Merkle trees. This cost can be significant if the public bulletin board is an L1 blockchain such as Ethereum, where every 32-byte word stored on-chain costs over US$2 at today's gas prices. RSA accumulators also impose the highest costs on the coordinator ($O(n^2)$) to compute witnesses, which may limit scalability.

RSA accumulators also require a trusted setup. This can be done for traditional RSA groups $\mathbb{Z}_N^*$ as a multiparty ceremony [16]. Deployments may also use class groups of imaginary quadratic order [12, 41], which avoid the need for trusted setup but have higher concrete overhead and lack well-understood security parameters.

Finally, we note that there may be interesting optimizations when combining RSA accumulators with RSA-based VDFs [45, 56], such as offering a combined proof of inclusion and VDF evaluation.

Bilinear accumulators. Bilinear accumulators can offer the combination of small (48 byte) commitments and constant-sized membership proofs (48 bytes) along with the same asymptotic efficiency as Merkle trees for computing membership proofs ($O(n \log n)$). Bilinear accumulators offer higher concrete overhead than for Merkle trees. In particular, they require pairing operations which are relatively expensive compared to hashing (though still cheap in concrete terms). However, only a single pairing operation by verifiers is required.

The downside is that bilinear accumulators require a trusted setup of an $O(n)$-sized structured reference string. This powers-of-tau string is common to many protocols and there are many approaches to generating it in a distributed manner [36, 43]. For example, the Filecoin setup generated 2^{27} powers of tau which can be used in a bilinear accumulator with up to $2^{27} \approx 130$ million participants [26]. Ethereum generated a smaller string with 2^{12} powers of tau in a community setup [27]. While the coordinator must store this entire string, participants need only access $O(1)$ terms to verify that their contributions were included.

Hyperproofs. Finally, Hyperproofs [52] is a vector commitment scheme with the feature that witnesses can be generated in batch very efficiently – generating all n witnesses takes $O(n \log n)$ time. Concretely, computing all n witnesses takes 0.7 hours for $n = 2^{22}$ and 2.7 hours for $n = 2^{24}$ as implemented in [52]. Verifying witnesses takes on the order of milliseconds. This efficiency is immediately inherited by the accumulator constructed using our approach in Subsection 5.5. The drawback of Hyperproofs is that it requires linear-sized public parameters that must be generated using a trusted setup. Merkle trees and bilinear accumulators also allow all witnesses to be batch computed in $O(n \log n)$ time.

7 Concluding Discussion

Cornucopia is a simple but powerful framework for VDF-based DRBs, using accumulators to construct open-participation randomness beacon protocols at massive scale. Our work shows that this paradigm is secure, and it can be instantiated with efficient accumulators which are already in common practical use (see Section 6). We discuss important practical extensions to the Cornucopia framework, leaving a complete analysis to future work.

7.1 The multi-coordinator model

Basic Cornucopia is entirely dependent on the (single) coordinator to achieve liveness; a malicious coordinator could prevent targeted individuals from contributing to the protocol (censorship), or even withhold the commitment R and prevent the protocol from finishing at all. This does not undermine our DRB security definitions (Section 3) since the coordinator cannot do so conditionally based on the impending outcome, but they can arbitrarily bias the outcome if they successfully block all honest participants.

A natural way to ensure liveness is to allow $k > 1$ coordinators, each of which posts an accumulator value R_i. The final beacon output is then computed as $\Omega = \mathsf{VDF.Eval}(H(R_1||\ldots||R_k))$. In the limit, every user might be their own coordinator ($k = n$), in which case the protocol is exactly the original Unicorn proposal [38]. Any number of malicious coordinators cannot undermine security of the protocol as long as least one honest contributor submits a value to one honest coordinator.

If *any* coordinator is honest, the protocol will finish, hence we can achieve liveness if any of k coordinators is honest. Users can submit contributions to multiple coordinators and trust the final output Ω as long as at least one coordinator includes their contribution. Combined, we can achieve 1-out-of-k liveness and 1-out-of-n security (for n contributors and k coordinators) for $k \leq n$.

While security and liveness are maximal with $k = n$, we note that in blockchain deployments the on-chain cost is $O(k)$, hence choosing $k \ll n$ is likely required for efficiency considerations. Furthermore, the consequences of a security failure are more severe than a liveness failure, and a liveness failure will be visible on-chain whereas manipulating a randomness beacon is typically impossible to detect.

In a blockchain setting, there is no need to fix the set of coordinators; any party can act as a coordinator as long as they are willing to pay the cost (e.g. gas) of posting their accumulation R_i to the bulletin board. Coordinators can even use different accumulators with different efficiency trade-offs. For example, a user participating across many epochs may prioritize shorter witnesses and prefer a coordinator using a bilinear accumulator with constant-sized witnesses but a trusted setup. Another user who participates only once may opt for a coordinator using a Merkle tree, requiring an $O(\log n)$-sized witness but avoiding the need for a trusted setup.

7.2 Public verifiability with notaries

As proposed, Cornucopia only offers meaningful security guarantees to participants who have contributed randomness to the protocol. Passive observers will have no idea if the coordinator actually included any honest participants' values in the published commitment. We can provide a notion of verifiability to purely passive observers by introducing a subset of *notarized participants* with some public reputation for honesty.

Notaries may be organizations such as nonprofits or government bodies who commit to participating in the protocol regularly. Each notary, after verifying its inclusion proof from the coordinator, signs the accumulator value. These signatures might be published by the coordinator or posted to the public bulletin board. To save space, they can be compressed using using a succinct *multi-signature* scheme such as BLS [11], resulting in only $O(1)$ additional overhead.

Observers can now verify the set of notarized participants who have contributed to the beacon output. As long as *one* of an observer's trusted notaries is honest and has signed the accumulator value, the final output Ω can be trusted. In practice, using BLS multi-signatures, this would be about as efficient to verify as a threshold-signature-based protocol like drand [24], while offering much stronger security (any honest notarized participant vs. a majority of honest nodes in drand).

7.3 Incentivizing participation

Analyzing incentives in public randomness generation is an important open problem for DRBs in general. For Cornucopia specifically, we must incentivize coordinator(s) to provide a highly reliable service and expend non-trivial effort computing inclusion proofs. This is somewhat similar to incentivizing nodes to participate in an honest-majority DRB such as drand. In general, randomness beacons are a *public good* in that they are non-rivalrous (their value is not decreased as more users rely on them) and non-excludable (it is difficult to prevent anybody from utilizing them for their own purposes). Standard economic theory predicts that public goods are susceptible to free-riding: users may not want to contribute to funding a coordinator if they can rely on the efforts of others to do so and still utilize the randomness beacon. We hope that the relatively low costs of running a coordinator means it might attract corporate sponsorship for publicity, be run by a foundation, or receive government support.

Second, in Cornucopia we must incentivize users to regularly contribute randomness and to ensure their local machine is uncompromised and generating randomness correctly. The potentially large scale of Cornucopia instances might paradoxically decrease user motivation: if the protocol is secure as long as at least one other user is honest, why expend the effort to contribute at all? This is a version of the *bystander effect*, whereby opening participation to more parties which can contribute security means all of them may figure somebody else will do it. Hopefully, the open nature of Cornucopia may provide a new type of incentive, as by participating users themselves gain trust that the result is secure.

References

1 Marcin Andrychowicz, Stefan Dziembowski, Daniel Malinowski, and Lukasz Mazurek. Secure Multiparty Computations on Bitcoin. In *IEEE Security & Privacy*, 2014.

2 Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. GRandLine: First Adaptively Secure DKG and Randomness Beacon with (Almost) Quadratic Communication Complexity. Cryptology ePrint Archive, Paper 2023/1887, 2023.

3 Michael Ben-Or and Nathan Linial. Collective coin flipping, robust voting schemes and minima of banzhaf values. In *FOCS*, 1985.

4 Michael Ben-Or and Nathan Linial. Collective coin flipping. *Advances in Computing Research*, 1989.

5 Josh Benaloh and Michael De Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Eurocrypt*, 1993.

6 Adithya Bhat, Aniket Kate, Kartik Nayak, and Nibesh Shrestha. OptRand: Optimistically responsive distributed random beacons. Cryptology ePrint Archive, Paper 2022/193, 2022.

7 Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication. Cryptology ePrint Archive, Paper 2020/1590, 2020.

8 Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 1983.

9 Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable Delay Functions. In *CRYPTO*, 2018.

10 Dan Boneh, Benedikt Bünz, and Ben Fisch. A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Paper 2018/712, 2018.

11 Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *Asiacrypt*, 2018.

12 Johannes Buchmann and Safuat Hamdy. A survey on IQ cryptography. In *Public-Key Cryptography and Computational Number Theory*, 2011.

13 Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *CRYPTO*, 2002.

14 Ignacio Cascudo and Bernardo David. Albatross: publicly attestable batched randomness based on secret sharing. In *Asiacrypt*, 2020.

15 Dario Catalano and Dario Fiore. Vector commitments and their applications. In *PKC*, 2013.

16 Megan Chen, Carmit Hazay, Yuval Ishai, Yuriy Kashnikov, Daniele Micciancio, Tarik Riviere, Abhi Shelat, Muthu Venkitasubramaniam, and Ruihan Wang. Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. In *IEEE Security & Privacy*, 2021.

17 Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic encryption random beacon. Cryptology ePrint Archive, Paper 2019/1320, 2019.

18 Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. Bicorn: An optimistically efficient distributed randomness beacon. In *Financial Crypto*, 2023.

19 Kevin Choi, Aathira Manoj, and Joseph Bonneau. Sok: Distributed randomness beacons. In *IEEE Security & Privacy*, 2023.

20 Miranda Christ, Kevin Choi, and Joseph Bonneau. Cornucopia: Distributed randomness beacons at scale. *Cryptology ePrint Archive*, 2023.

21 Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *TOC*, 1986.

22 Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *IEEE Security & Privacy*, 2022.

23 Yevgeniy Dodis. Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping. In *ECCC*, 2000.

24 Drand. `https://drand.love/`.

25 Dankrad Feist. RSA Assumptions. `rsa.cash/rsa-assumptions/`, 2022.

26 FileCoin. Trusted setup complete!, 2020. URL: `https://filecoin.io/blog/posts/trusted-setup-complete/`.

27 Ethereum Foundation. Proto-danksharding, 2023. URL: `https://www.eip4844.com/`.

28 Georg Fuchsbauer, Eike Kiltz, and Julian Loss. The algebraic group model and its applications. In *CRYPTO*, 2018.

29 Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953, 2019.

30 David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *Euro S&P*, 2021.

31 Shafi Goldwasser, Yael Tauman Kalai, and Sunoo Park. Adaptively secure coin-flipping, revisited. In *ICALP*, 2015.

32 Zhaozhong Guo, Liucheng Shi, and Maozhi Xu. SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability. *IEEE Access*, 2020.

33 Iftach Haitner and Yonatan Karidi-Heller. A tight lower bound on adaptively secure full-information coin flip. In *FOCS*, 2020.

34 Yael Tauman Kalai, Ilan Komargodski, and Ran Raz. A lower bound for adaptively-secure collective coin flipping protocols. *Combinatorica*, 41(1), 2021.

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

36 Thomas Kerber, Aggelos Kiayias, and Markulf Kohlweiss. Mining for Privacy: How to Bootstrap a Snarky Blockchain. In *Financial Crypto*, 2021.

37 Hsun Lee, Yuming Hsu, Jing-Jie Wang, Hao Cheng Yang, Yu-Heng Chen, Yih-Chun Hu, and Hsu-Chun Hsiao. HeadStart: Efficiently Verifiable and Low-Latency Participatory Randomness Generation at Scale. In *NDSS*, 2022.

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

39 Jiangtao Li, Ninghui Li, and Rui Xue. Universal accumulators with efficient nonmembership proofs. In *ACNS*, 2007.

40 Helger Lipmaa. Secure accumulators from Euclidean rings without trusted setup. In *ACNS*, 2012.

41 Lipa Long. Binary quadratic forms. `https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf`, 2018.

42 Lan Nguyen. Accumulators from bilinear pairings and applications. In *CT-RSA*, 2005.

43 Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In *ACNS*, 2024.

44 Charalampos Papamanthou. *Cryptography for efficiency: new directions in authenticated data structures*. PhD thesis, Brown University, 2011.

45 Krzysztof Pietrzak. Simple Verifiable Delay Functions. In *ITCS*, 2018.

46 Youcai Qian. Randao: Verifiable random number generation. `randao.org/whitepaper/Randao_v0.85_en.pdf`, 2017.

47 Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, 1983.

48 Mayank Raikwar and Danilo Gligoroski. SoK: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*, 2022.

49 Mark Ryan. Enhanced Certificate Transparency and End-to-End Encrypted Mail. In *NDSS*, 2014.

50 Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In *NDSS*, 2023.

51 Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Hydrand: Efficient continuous distributed randomness. In *IEEE Security & Privacy*, 2020.

52 Shravan Srinivasan, Alexander Chepurnoy, Charalampos Papamanthou, Alin Tomescu, and Yupeng Zhang. Hyperproofs: Aggregating and maintaining proofs in vector commitments. In *USENIX Security*, 2022.

53 Shravan Srinivasan, Ioanna Karantaidou, Foteini Baldimtsi, and Charalampos Papamanthou. Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *ACM CCS*, 2022.

54 Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *IEEE Security & Privacy*, 2017.

55 Weijie Wang, Annie Ulichney, and Charalampos Papamanthou. {BalanceProofs}: Maintainable Vector Commitments with Fast Aggregation. In *USENIX Security*, 2023.

56 Benjamin Wesolowski. Efficient Verifiable Delay Functions. In *Eurocrypt*, 2019.

57 David Yakira, Avi Asayag, Ido Grayevsky, and Idit Keidar. Economically viable randomness. *CoRR*, 2020.