

# Cornucopia: Distributed Randomness Beacons at Scale

Miranda Christ  
Columbia University  
mchrist@cs.columbia.edu

Kevin Choi  
New York University  
kevin.choi@cs.nyu.edu

Joseph Bonneau  
New York University  
a16z crypto research  
jcb@cs.nyu.edu

## ABSTRACT

We propose Cornucopia, a distributed randomness beacon protocol combining accumulators and verifiable delay functions. Cornucopia extends the Unicorn protocol of Lenstra and Wesolowski, utilizing an accumulator to enable efficient verification by each participant that their randomness contribution has been included in the beacon output. The output is unpredictable as long as at least one participant is honest, yielding a highly scalable distributed randomness beacon with strong security properties. The security of this construction reduces to a novel property of accumulators, *insertion security*. We first show that not all accumulators are insertion-secure. We 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. Finally, we give two generic constructions for insertion-secure accumulators, from universal accumulators and vector commitments respectively.

## 1 INTRODUCTION

The goal of distributed randomness beacons (DRBs) is to enable a group of  $n$  mutually untrusting participants to jointly compute a random output (which we denote  $\Omega$ ) which cannot be predicted or biased by any participant or any coalition of participants. Among many important applications of DRBs are cryptographically verifiable lotteries and leader election in distributed consensus protocols.

A classic approach to constructing DRBs is *commit-reveal* [8]. First, all participants publish a cryptographic commitment to a random value  $r_i$ . Participants then reveal their  $r_i$  values, and the result is  $\Omega = \text{Combine}(r_1, \dots, r_n)$  for some suitable combination function (such as exclusive-or or a cryptographic hash). Commit-reveal protocols are simple and efficient, and remain secure as long as at least one participant chooses a random  $r_i$  value and all participants always 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  $\Omega$  is not to their liking. The protocol can either finish without the missing  $r_i$ , or restart and remove them from future protocol runs. Either way, the attacker obtains 1 bit of bias on  $\Omega$ .

Most approaches to avoiding last-revealer attacks relax the security model by assuming most participants are honest and enabling a majority coalition to recover a withholding participant's contribution. However, this means that a malicious majority can privately compute  $\Omega$  early and potentially bias it. Protocols of this type also typically require communication and computation superlinear in  $n$  (though some amortize this over multiple rounds).

A fundamentally different approach to constructing DRBs uses time-based cryptography, specifically using *delay functions* to prevent manipulation. The simplest example is Unicorn [37], a one-round protocol in which each participant directly publishes (within a fixed time window) a random input  $r_i$  to a public bulletin board. The result is computed as  $\Omega = \text{Delay}(\text{Combine}(r_1, \dots, r_n))$ . By assumption, a participant cannot compute the Delay function before the deadline to publish their contribution  $r_i$  and therefore cannot predict  $\Omega$  or choose  $r_i$  in such a way as to manipulate it. This protocol retains the strong  $n - 1$  (dishonest majority) 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.

**Our approach.** We formalize the approach of using a cryptographic accumulator (for example, a Merkle tree) to publish a succinct commitment to all users' contributions, retaining the security advantages of Unicorn while reducing the storage overhead (on the public bulletin board) from  $\Theta(n)$  to  $O(1)$ . We call this general protocol Cornucopia, with a general structure as follows:

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

While this is a small change to Unicorn, it is powerful: individual users can now be convinced that  $\Omega$  is truly random with sublinear verification costs. Observe that since security requires only one honest participant, individuals only need to verify that *they themselves participated in the protocol* (assuming they trust that their own device has not been compromised). 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 availability but not for security.

This approach opens the door to massive *open-participation* randomness protocols. For example, every user buying a lottery ticket might contribute randomness, or every user in a massively multiplayer online (MMO) game might contribute randomness to seed

the game engine. These applications might include millions of participants, which would not be feasible with an honest majority requirement or linear verification costs per user. Cornucopia, by contrast, can offer constant or logarithmic verification costs (depending on the choice of accumulator) thus making planet-scale distributed randomness generation possible. The coordinator does face at least linear costs to compute the accumulator and per-user proofs, but for certain accumulators [50, 53], 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 [46] and foundational work on *distributed coin tossing* [3, 4, 20, 22, 30, 32, 33]. Several recent surveys cover modern DRBs [19, 34, 47]. Most of this work is orthogonal, with protocols operating without the benefit of delay functions and hence assuming honest majorities [2, 6, 7, 14, 17, 21, 23, 29, 31, 49, 52] or economic incentives [1, 45, 55].

Unicorn [37] 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 [48] 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 [36] is the most similar DRB construction to Cornucopia, also using Merkle trees and a multi-round pipelined protocol to scale up Unicorn by combining many users’ contributions in a succinct commitment. We adopt the same conceptual approach as HeadStart, but our approach differs in offering a generic construction from any accumulator and developing precise security notions required of accumulators for use with DRBs.

#### Our contributions.

- We formalize the concept of 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 that satisfies a natural security notion that we develop, called *insertion security*.
- We prove (in Section 5) that the most commonly used accumulator constructions either naturally feature insertion security (Merkle trees) or need only trivial modifications to achieve it (RSA accumulators, bilinear accumulators, and accumulators from vector commitments), meaning Cornucopia is practical to build from standard cryptographic assumptions and implementations. Furthermore, the efficiency of Cornucopia can take likely take advantage of future accumulator schemes (assuming insertion security can be proven).
- We compare performance implications of different accumulators (Section 6). Since Cornucopia can be instantiated with any insertion-secure accumulator, the protocol can be tailored to different settings by choosing an accumulator to optimally trade off communication and computation.

Finally, we conclude in Section 7 with discussion about some protocol extensions and open problems.

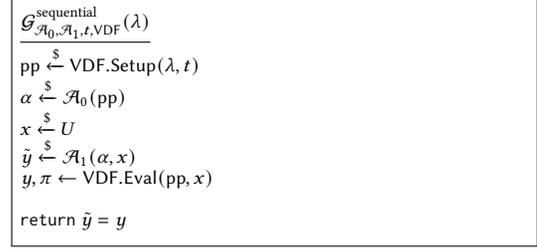


Figure 1: VDF sequentiality game

## 2 PRELIMINARIES

To define Cornucopia, we first need to define verifiable delay functions (VDFs) [9] and accumulators [5]. Both rely on public parameters  $\text{pp}$  which all functions take implicitly, though we will typically omit this for brevity. We use  $\lambda$  to denote a security parameter, and  $\text{poly}(\lambda)$  and  $\text{negl}(\lambda)$  to denote polynomial and negligible functions of  $\lambda$ , respectively. We use  $\stackrel{\$}{\leftarrow}$  (or  $\stackrel{\$}{\rightarrow}$ ) to denote the output of a randomized algorithm, or sampling uniformly at random from a range. We use  $\alpha$  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  $\lambda$ ; some adversaries are further limited to running in  $\sigma(t)$  steps on at most  $p(t)$  parallel processors where noted. We let  $[k]$  denote the set  $\{1, \dots, k\}$ .

### 2.1 Verifiable delay functions

**DEFINITION 1 (VERIFIABLE DELAY FUNCTION [9]).** A verifiable delay function (VDF) [9] is a tuple of algorithms (Setup, Eval, Verify) where:

$\text{VDF.Setup}(\lambda, t) \rightarrow \text{pp}$  takes as input  $\lambda$  and a time parameter  $t$  and outputs public parameters  $\text{pp}$ .

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

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

VDFs must satisfy the following three properties:

**Verifiability.** The verification algorithm is efficient (at most poly-logarithmic in  $t$  and  $\lambda$ ) 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  $(p, \sigma)$ -sequential if for all randomized algorithms  $\mathcal{A}_0$  which run in total time  $O(\text{poly}(t, \lambda))$ ,

$$\begin{array}{l}
\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{acc}}(\lambda) \\
\text{pp} \stackrel{\$}{\leftarrow} \text{Acc.Setup}(\lambda) \\
S, x, w \stackrel{\$}{\leftarrow} \mathcal{A}(\text{pp}) \\
A \leftarrow \text{Acc.Accumulate}(S) \\
\\
\text{return} \\
\text{Acc.MemVer}(A, x, w) \wedge x \notin S
\end{array}$$

Figure 2: Accumulator security game

and  $\mathcal{A}_1$  which run in parallel time  $\sigma(t)$  on at most  $p(t)$  processors:

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

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

## 2.2 Accumulators

DEFINITION 3 (ACCUMULATOR [5, 13]). Given a data universe  $U$ , an accumulator [5] is a tuple of algorithms (Setup, Accumulate, GetMemWit, MemVer) where:

$\text{Acc.Setup}(\lambda) \rightarrow \text{pp}$  takes as input  $\lambda$  and outputs public parameters pp.

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

$\text{Acc.GetMemWit}(S, A, x) \rightarrow w$  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  $x$ .

$\text{Acc.MemVer}(A, x, w) \rightarrow \{\text{true}, \text{false}\}$  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.

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  $\text{Acc}$  is secure if no PPT adversary  $\mathcal{A}$  can succeed with non-negligible probability in  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{acc}}(\lambda)$  as defined in Figure 2.

A universal accumulator [38] also supports non-membership proofs; that is, it supports two additional functions:

$\text{Acc.GetNonMemWit}(S, A, x') \rightarrow w'$  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  $x'$ .

$\text{Acc.NonMemVer}(A, x', w') \rightarrow \{\text{true}, \text{false}\}$  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 any user that their contribution is *not* included. However, in Section 5.5 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 (in addition to basic accumulator security) that no adversary can find valid membership and non-membership proofs for the same element:

DEFINITION 5 (UNIVERSAL ACCUMULATOR SECURITY [38]). A universal accumulator  $\text{Acc}$  is secure if for all PPT adversaries  $\mathcal{A}$ :

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

## 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:

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

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

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

$\text{Ver}(C, m, i, \pi_i) \rightarrow \{\text{true}, \text{false}\}$  takes as input a commitment, a message  $m$ , an index  $i$ , and a proof that the  $i^{\text{th}}$  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. A vector commitment must also satisfy *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  $\mathcal{A}$ :

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

```


$$\mathcal{G}_{\mathcal{A},t,b,\text{DRB}}^{\text{indist}}(\lambda)$$

pp  $\xleftarrow{\$}$  Setup( $\lambda, t$ )
 $r_1 \xleftarrow{\$}$  Prepare(pp)
 $\alpha_0 \xleftarrow{\$}$   $\mathcal{A}_0$ (pp)
 $\alpha_1, R, \pi_1 \xleftarrow{\$}$   $\mathcal{A}_1(\alpha_0, r_1)$ 
 $\Omega_0, \pi_0 \xleftarrow{\$}$  Finalize(pp, R)
 $\Omega_1 \xleftarrow{\$}$   $U$ 
 $b' \xleftarrow{\$}$   $\mathcal{A}_2(\alpha_1, \Omega_b)$ 

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

```

Figure 3: Security game for  $(p, \sigma)$ -indistinguishability

### 3 TIMED DRBS: DEFINITIONS AND CONSTRUCTIONS

We first define timed DRBs using a generalized syntax.<sup>1</sup>

**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  $\pi_i$ . Typically, this value  $R$  will be posted to a public bulletin board, whereas  $\pi_i$  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  $\Omega$  and (optionally) a proof  $\pi_\Omega$ . It is a deterministic algorithm running in time  $(1 + \epsilon)t$  for some small  $\epsilon$ .

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

A timed DRB has the following security properties (shown in Figures 4 and 3):

**DEFINITION 9 ( $(p, \sigma)$ -UNPREDICTABILITY).** The  $(p, \sigma)$ -unpredictability game tasks an adversary with predicting the final output  $\Omega$  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  $\text{poly}(\lambda, t)$ . 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,

```


$$\mathcal{G}_{\mathcal{A},t,\text{DRB}}^{\text{unpred}}(\lambda)$$

pp  $\xleftarrow{\$}$  Setup( $\lambda, t$ )
 $r_1 \xleftarrow{\$}$  Prepare(pp)
 $\alpha_0 \xleftarrow{\$}$   $\mathcal{A}_0$ (pp)
 $\tilde{\Omega}, \pi_{\tilde{\Omega}}, R, \pi_1 \xleftarrow{\$}$   $\mathcal{A}_1(\alpha_0, r_1)$ 

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

```

Figure 4: Security game for  $(p, \sigma)$ -unpredictability

```

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

Prepare()  $\xrightarrow{\$}$   $r_i$ 
 $r_i \xleftarrow{\$}$   $U$ 

Post( $\{r_i\}$ )  $\rightarrow (R, \emptyset)$ 
 $R \leftarrow \{r_i\}$ 

Finalize(R)  $\rightarrow (\Omega, \pi_\Omega)$ 
 $\Omega, \pi_\Omega \leftarrow$  VDF.Eval( $H(R)$ )

Verify(pp, R,  $\Omega, \pi_\Omega, r_i, \pi_i$ )  $\rightarrow \{\text{true}, \text{false}\}$ 
return  $r_i \in R \wedge$  VDF.Verify( $H(R), \Omega, \pi_\Omega$ )

```

Figure 5: The Unicorn timed DRB protocol [37]

exactly like the adversary for VDF sequentiality (Definition 2). The adversary's advantage is:  $\text{Adv}_{\mathcal{A},t,\text{DRB}}^{\text{unpred}}(\lambda) = \Pr \left[ \mathcal{G}_{\mathcal{A},t,\text{DRB}}^{\text{unpred}}(\lambda) = 1 \right]$ .

As the  $(p, \sigma)$ -unpredictability property does not guarantee the DRB output is indistinguishable from random, we define a stronger  $(p, \sigma)$ -indistinguishability property in which the adversary must distinguish a DRB output from a random output, again allowing the adversary control of all but one participant.

**DEFINITION 10 ( $(p, \sigma)$ -INDISTINGUISHABILITY).** The  $(p, \sigma)$ -indistinguishability game is exactly like the  $(p, \sigma)$ -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:

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

As observed by Boneh et al. [9], there is a generic transformation in the random oracle model in which a timed DRB which satisfies  $(p, \sigma)$ -unpredictability can be transformed generically into one with  $(p, \sigma)$ -indistinguishability by applying the random oracle to the output.

#### 3.1 Unicorn

As a warm-up, we describe Unicorn [37] succinctly as a timed DRB in our framework in Figure 5. 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].

<sup>1</sup>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.

|  |
|--|
| $\text{Setup}(\lambda, t) \xrightarrow{\$} \text{pp}$ $\text{pp} \leftarrow (\text{VDF.Setup}(\lambda, t), \text{Acc.Setup}(\lambda))$   |
| $\text{Prepare}() \xrightarrow{\$} r_i$ $r_i \xleftarrow{\$} U$  |
| $\text{Post}(\{r_i\}) \rightarrow (R, \{\pi_i\})$ $R \leftarrow \text{Acc.Accumulate}(\{r_i\})$ $\pi_i \leftarrow \text{Acc.GetMemWit}(\{r_j\}, R, r_i)$   |
| $\text{Finalize}(R) \rightarrow (\Omega, \pi_\Omega)$ $\Omega, \pi_\Omega \leftarrow \text{VDF.Eval}(H(R))$  |
| $\text{Verify}(\text{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{\text{true}, \text{false}\}$ $\text{return VDF.Verify}(H(R), \Omega, \pi_\Omega) \wedge \text{Acc.MemVer}(R, r_i, \pi_i)$ |

Figure 6: The Cornucopia protocol

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 user is enough to ensure this hashed value cannot have been predicted and precomputed by the adversary. Lenstra and Wesolowski prove security of Unicorn in a slightly different model [37]. We note that its security is also implied by our security proof for Cornucopia in Theorem 4.1, as Unicorn is a special case using the “concatenation accumulator” which simply concatenates all accumulated values.

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 6, 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 makes it easy to scale to many users with low publishing costs and low costs for users to verify that the beacon output  $\Omega$  incorporates their contribution  $r_i$ . 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.

## 4 CORNUCOPIA SECURITY

Towards proving that Cornucopia is a secure timed DRB, we first must define a novel security property for accumulators, *insertion security*:

**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 7) is negligible:*

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

To win the insertion security game (Figure 7), 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  $A$

|  |
|--|
| $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$ $\text{pp} \xleftarrow{\$} \text{Acc.Setup}(\lambda)$ $A \leftarrow \mathcal{A}(\text{pp})$ $x \xleftarrow{\$} U$ $w \leftarrow \mathcal{A}(\text{pp}, A, x)$ $\text{return Acc.MemVer}(A, x, w)$ |
|--|

Figure 7: Insertion security game

via the normal Accumulate function; it can produce  $A$  using any procedure at all. We will prove this property holds for concrete accumulators in Section 5, for now we will assume we have access to an accumulator which satisfies this property.

We next 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 randomness contribution  $r_1$ . Insertion security implies that it is difficult to precompute an accumulator value for which one can provide a membership proof of a random element revealed later. The second states that if the adversary does not query  $R$  to the random oracle in its pre-computation phase, it cannot output  $\tilde{\Omega} = \text{VDF.Eval}(H(R))$ . This is because after the precomputation phase, the adversary is  $(p, \sigma)$ -sequential and therefore cannot evaluate the VDF; thus, to prove this lemma we invoke VDF sequentiality.

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

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

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

We define an adversary  $\mathcal{B}$  that breaks insertion security of the accumulator scheme by simulating the challenger in  $\mathcal{G}_{\mathcal{A}, t, \text{CC}}^{\text{unpred}}(\lambda)$  and using  $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ .  $\mathcal{B}$  first receives  $\text{Acc.pp}$  in  $\mathcal{G}_{\mathcal{B}, \text{Acc}}^{\text{insert}}(\lambda)$ . It samples  $\text{VDF.pp} \leftarrow \text{VDF.Setup}(\lambda, t)$  and passes  $\text{pp} = (\text{Acc.pp}, \text{VDF.pp})$  to  $\mathcal{A}_0$ .  $\mathcal{B}$  simulates the challenger in  $\mathcal{G}_{\mathcal{A}, t, \text{CC}}^{\text{unpred}}(\lambda)$  and records the queries  $q_1, \dots, q_k$  that  $\mathcal{A}_0$  makes to the random oracle.  $\mathcal{B}$  also receives  $\alpha_0$  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  $\mathcal{G}_{\mathcal{B}, \text{Acc}}^{\text{insert}}(\lambda)$ .  $\mathcal{B}$  then receives  $x$  from the challenger in  $\mathcal{G}_{\mathcal{B}, \text{Acc}}^{\text{insert}}(\lambda)$ , and it continues simulating the  $\mathcal{G}_{\mathcal{A}, t, \text{CC}}^{\text{unpred}}(\lambda)$  challenger by passing  $\alpha_0$  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  $\frac{1}{\lambda^c}$ ,  $\Pr[\text{MemVer}(R, x, w_1) = \text{true} \wedge \mathcal{A}_0 \text{ queried } R \text{ to the random oracle}] \geq \frac{1}{\lambda^c}$ . 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  $\mathcal{A}$ ,  $\Pr[\text{MemVer}(R, x, w_1) = \text{true} \wedge A = R] \geq \frac{1}{\lambda^c} \cdot \frac{1}{q(\lambda)}$  which is non-negligible. Thus, with non-negligible probability,  $\mathcal{G}_{\mathcal{B}, \text{Acc}}^{\text{insert}}(\lambda) = 1$ .  $\square$

LEMMA 2. Let  $\mathcal{E}_2$  be the event that  $\mathcal{G}_{\mathcal{A},t,CC}^{\text{unpred}}(\lambda) = 1$  and  $\mathcal{A}_0$  did not query  $R$  to the random oracle. If CC is instantiated with an insertion-secure accumulator and a  $(p, \sigma)$ -sequential VDF, then  $\Pr[\mathcal{E}_2] \leq \text{negl}(\lambda)$ .

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

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

We define an adversary  $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1)$  that breaks  $(p, \sigma)$ -sequentiality of the VDF by simulating the challenger and random oracle in  $\mathcal{G}_{\mathcal{A},t,CC}^{\text{unpred}}$  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  $(\lambda, \text{VDF.pp}, t)$  from the VDF challenger in  $\mathcal{G}_{\mathcal{B}_0, \mathcal{B}_1, t, \text{VDF}}^{\text{sequential}}(\lambda)$ .  $\mathcal{B}_0$  samples  $\text{Acc.pp} \leftarrow \text{Acc.Setup}(\lambda)$  and passes  $\text{pp} = (\text{VDF.pp}, \text{Acc.pp})$  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  $\alpha_0$ , 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  $\text{VDF.pp}$  and  $\alpha$ .  $\mathcal{B}_1$  passes  $\alpha_0$  and a randomly-generated value  $r_1 \xrightarrow{\$} \text{Prepare}(\text{pp})$  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<sup>2</sup> and answers  $\mathcal{A}_1$ 's  $i^{\text{th}}$  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  $\mathcal{G}_{\mathcal{A},t,CC}^{\text{unpred}}$ ,  $\mathcal{B}$  wins  $\mathcal{G}_{\mathcal{B}_0, \mathcal{B}_1, t, \text{VDF}}^{\text{sequential}}(\lambda)$  with non-negligible probability. First, recall that if  $\mathcal{A}$  wins,  $\text{DRB.Verify}$  holds. By uniqueness of the VDF, the probability that  $\mathcal{A}_1$  outputs a proof  $\pi_{\tilde{\Omega}}$  such that  $\text{VDF.Verify}(\text{VDF.pp}, H(R), \tilde{\Omega}, \pi_{\tilde{\Omega}}) = 1$  yet  $\tilde{\Omega} \neq \text{VDF.Eval}(H(R))$  is negligible. Thus, since  $\text{DRB.Verify}$  holds,  $\mathcal{A}_1$  must have output  $\tilde{\Omega} = \text{VDF.Eval}(H(R))$ .

We now show that the fact that  $\mathcal{A}_1$  outputs  $\text{VDF.Eval}(H(R))$  implies that  $\mathcal{B}$  breaks  $(p, \sigma)$ -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  $\tilde{\Omega} = \text{VDF.Eval}(x)$  as desired.

<sup>2</sup>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.

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

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

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  $x$ .  $\square$

THEOREM 4.1 (UNPREDICTABILITY OF CORNUCOPIA). *Cornucopia is  $(p, \sigma)$ -unpredictable when instantiated with an insertion-secure accumulator, a  $(p, \sigma)$ -sequential VDF, and a hash function modeled as a random oracle.*

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

Observe that  $\Pr[\mathcal{G}_{\mathcal{A},t,CC}^{\text{unpred}}(\lambda) = 1] = \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2]$ . By Lemma 1,  $\Pr[\mathcal{E}_1] \leq \text{negl}(\lambda)$ . By Lemma 2,  $\Pr[\mathcal{E}_2] \leq \text{negl}(\lambda)$ . Therefore,  $\Pr[\mathcal{G}_{\mathcal{A},t,CC}^{\text{unpred}}(\lambda) = 1] \leq \text{negl}(\lambda)$ .  $\square$

COROLLARY 12. *Cornucopia is  $(p, \sigma)$ -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

Given any secure accumulator scheme  $\text{Acc}$ , it is trivial to construct an accumulator  $\text{Acc}'$  which is not insertion-secure, but otherwise satisfies the standard security definitions of an accumulator. One approach is to add a special symbol  $\epsilon$  which is defined as the accumulation of the entire data universe  $U$ .  $\text{Acc}'.\text{MemVer}(A, x, w)$  is defined to be 1 if  $A = \epsilon$  (regardless of the value of  $x$  or  $w$ ), and otherwise is equal to  $\text{Acc.MemVer}(A, x, w)$ . The scheme  $\text{Acc}'$  can be used exactly as  $\text{Acc}$  in normal operation, with the extra property that  $\epsilon$  is a ‘‘shortcut’’ to computing an accumulation of 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 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  $w = \text{Acc.GetMemWit}(S, A, H(x, y))$  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  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$  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.<sup>3</sup>

## 5.2 Merkle trees

LEMMA 3. *A Merkle tree of bounded depth  $k = \text{poly}(n)$  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  $\mathcal{O} : \{0, 1\}^{2^n} \rightarrow \{0, 1\}^n$ . Let  $A$  be the accumulator output by an adversary  $\mathcal{A}$  in  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$ . We show that for a uniform  $x \in \{0, 1\}^n$ , the adversary can provide a verifying witness  $w = (w_1, \dots, w_k)$  for  $x$  with only negligible probability. For a verifying witness, it must hold that  $\mathcal{O}(w_k || \dots || \mathcal{O}(w_2 || \mathcal{O}(w_1 || x))) = A$ . 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  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$ .

This can be shown by induction. Let  $a_1, \dots, a_\ell$  be the adversary's queries to the random oracle in step 2. Let  $b_1, \dots, b_k$  be the queries to the random oracle in the Merkle membership proof verification; that is,  $b_i = w_i || \mathcal{O}(w_{i-1} || \dots)$ . 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  $\frac{i \cdot p(\lambda)}{2^\lambda}$ . 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  $w_{i+1}$ ).

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

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  $\frac{k \ell \cdot p(\lambda)}{2^\lambda} \leq \text{negl}(\lambda)$ .

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

## 5.3 RSA accumulators

In a standard RSA accumulator [13, 39],  $\text{Setup}(\lambda)$  generates a random group of unknown order and a generator  $g$  for this group using some group generation algorithm  $\text{GenGroup}$ . The data universe is  $\Pi_\lambda$ , the set of all  $\lambda$ -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

<sup>3</sup>The adversary can in fact win with non-negligible probability by accumulating any range whose size is a constant fraction of  $|U|$ .

value  $A$  is  $w = g^{\prod_{x' \in S \setminus \{x\}} x'} = A^{1/x}$ .  $\text{Add}(A, x)$  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  $A_{|S|}$ .  $\text{UpdWit}(A_t, x, w'_t)$  outputs  $w'_{t+1} = (w'_t)^x$ .  $\text{MemVer}(A, x, w)$  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  $B = g^b$ .  $\text{NonMemVer}(A, \{a, B\}, x)$  outputs 1 if and only if  $A^a B^x = g$ .

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

ASSUMPTION 1 (ADAPTIVE ROOT ASSUMPTION [10]).

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

LEMMA 4. *Suppose a standard RSA accumulator is modified so that the algorithm  $\text{MemVer}(A, x, w)$  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  $\text{GenGroup}$ .*

PROOF. Suppose that there exists a PPT adversary  $\mathcal{A}$  that wins  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$  with probability at least  $\frac{1}{\text{poly}(\lambda)}$  when the data universe is  $\Pi_\lambda$ , the set of all  $\lambda$ -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  $\mathbb{G} \xleftarrow{\$} \text{GenGroup}(\lambda)$ .  $\mathcal{B}_0$  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  $\mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda)$ , it means that  $\text{MemVer}(A, x, w_x) = \text{true}$ . For RSA accumulators,  $\text{MemVer}(A, x, w_x) = \text{true}$  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  $\frac{1}{\text{poly}(\lambda)}$ ,  $(\mathcal{B}_0, \mathcal{B}_1)$  win with probability at least  $\frac{1}{\text{poly}(\lambda)}$ , violating the Adaptive Root Assumption.  $\square$

COROLLARY 13. *The modified RSA accumulator is insertion-secure in the Algebraic Group Model (AGM), since the Adaptive Root Assumption holds in the AGM [24].*

## 5.4 Bilinear accumulators

We show that bilinear accumulators [41, 51] 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 [41], and we follow [43] in its presentation. Let  $\mathbb{G}, \mathcal{G}$  be cyclic multiplicative groups of prime order  $p$ , and let  $e : \mathbb{G} \times \mathbb{G} \rightarrow \mathcal{G}$  be a bilinear pairing. Let  $s \xleftarrow{\$} \mathbb{Z}_p^*$ , and let  $g$  be a generator of  $\mathbb{G}$ . Let  $\text{srs} = [g, g^s, \dots, g^{s^q}]$  be the structured reference string, where  $q$  is an (polynomial in  $\lambda$ ) upper bound on

the number of accumulated elements. The public parameters are  $(p, \mathbb{G}, \mathcal{G}, e, g, s, \text{srs})$ . 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) [27], 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  $[1, g, g^s, \dots, g^{s^q}]$ . 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 [27, 28] 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 2** (*q-DISCRETE LOGARITHM ASSUMPTION* (*q-DLOG*) [27]). *The q-DLOG assumption holds in a group  $\mathbb{G}$  if for every p.p.t. adversary  $\mathcal{A}$ ,*

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

**LEMMA 5.** *The bilinear accumulator of [41] 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  $\alpha$  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  $s$ .  $\square$

## 5.5 From generic universal accumulators

Finally, we show how to construct an insertion-secure accumulator  $\text{Acc}'$  from any universal accumulator  $\text{Acc}$ . 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  $\text{Acc}'$  in requires showing *inclusion* of all elements of  $S_x^+$  in  $\text{Acc}$  and *exclusion* of all elements of  $S_x^-$  in  $\text{Acc}$ . Intuitively, breaking insertion security by accumulating the entire data universe in  $\text{Acc}$  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  $\text{Acc}$  be a universal accumulator scheme for data universe  $U$ . Here, we let the data universe for  $\text{Acc}'$  be  $U' = \{0, 1\}^\lambda$ . Let  $H : [\lambda] \times U' \rightarrow U$  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  $S_x^- := \{y : H(i, x) = y \text{ for } i \in [(\frac{\lambda}{2} + 1), \dots, \lambda]\}$  (assume for convenience that  $\lambda$  is even). We specify the functions of  $\text{Acc}'$  as follows:

**Setup:** uses the same setup function as  $\text{Acc}$ .

**Accumulate( $S'$ ):** Let  $S = \bigcup_{x \in S'} S_x^+$ . Outputs  $A = \text{Acc.Accumulate}(S)$ .

**GetMemWit( $S', A, x$ ):** Outputs a vector of witnesses  $w$  of length  $\lambda$  where:

- For  $i \leq \frac{\lambda}{2}$ ,  $w_i = \text{Acc.GetMemWit}(S, A, H(i, x))$  is a membership proof for  $H(i, x)$
- For  $i > \frac{\lambda}{2}$ ,  $w_i = \text{Acc.GetNonMemWit}(S, A, H(i, x))$  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  $i \leq \frac{\lambda}{2}$ ,  $\text{Acc.MemVer}(A, H(i, x), w_i) = \text{true}$ .
- For  $i > \frac{\lambda}{2}$ ,  $\text{Acc.NonMemVer}(A, H(i, x), w_i) = \text{true}$ .

**LEMMA 6.** *If  $\text{Acc}$  is a secure universal accumulator and  $H$  is modeled as a random oracle,  $\text{Acc}'$  is insertion-secure.*

**PROOF.** Suppose for the sake of contradiction that  $\text{Acc}'$  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  $\lambda$ -length tuple of truly random independent values  $y_i \in U$ , where  $y_1, \dots, y_{\frac{\lambda}{2}}$  should be included, and  $y_{\frac{\lambda}{2}+1}, \dots, y_\lambda$  should be excluded.

Equivalently, we can think of drawing  $y = y_1, \dots, y_\lambda$  (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  $y \in X$ ,  $\mathcal{A}$  succeeds with non-negligible probability over subsequent choice of  $\mathbf{b} \in \{0, 1\}^\lambda$ . Let  $\mathcal{E}[A, y, \mathbf{b}, w]$  denote the event that  $\text{Acc.MemVer}(A, y_i, w_i) = \text{true}$  for all  $i$  such that  $b_i = 1$ , and  $\text{Acc.NonMemVer}(A, y_i, w_i) = \text{true}$  for all  $i$  such that  $b_i = 0$ . The success of  $\mathcal{A}$  in  $\mathcal{G}_{\mathcal{A}, \text{Acc}'}^{\text{insert}}(\lambda)$  implies that  $\mathcal{E}[A, y, \mathbf{b}, w]$  occurs for its choice of  $A$  and  $w$ , and the random choice of  $y, \mathbf{b}$ . Thus,

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

We now construct an adversary  $\mathcal{B}$  that breaks universal security of  $\text{Acc}$  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  $y$  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  $(y, \mathbf{b}_1)$  and  $(y, \mathbf{b}_2)$  to obtain  $\mathbf{w}_1$  and  $\mathbf{w}_2$  respectively. With probability at least  $\frac{1}{\lambda^c}$ ,  $\mathcal{B}$  chose  $\text{pp}$  and  $y$  such that  $\Pr_{\mathbf{b}}[\mathbf{w} \leftarrow \mathcal{A} \wedge \mathcal{E}[A, y, \mathbf{b}, \mathbf{w}]] \geq \frac{1}{\lambda^c}$ . 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  $\text{Acc.MemVer}(A, y_i, (w_1)_i) = \text{true}$  and  $\text{Acc.NonMemVer}(A, y_i, (w_2)_i) = \text{true}$ . 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  $\text{Acc}$ .  $\square$

*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  $\text{MemVer}$ ; similarly, honestly-generated non-membership proofs for elements not in that set should verify under  $\text{NonMemVer}$ . We note that  $\text{Acc}'$  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  $\text{Acc}$ ), which should be difficult by security of  $\text{Acc}$ , 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.

## 5.6 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  $\lambda$ . Let the message space  $\mathcal{M}$  underlying our VC have size exponential in  $\lambda$ , 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:

$\text{Setup}(\lambda)$  : Output  $\text{pp} \leftarrow \text{VC.Setup}(\lambda)$ .

$\text{Accumulate}(S)$  : Interpret  $S$  as an ordered list  $s_1, \dots, s_{|S|}$ , and let  $v = [s_1, \dots, s_{|S|}, 0, \dots, 0]$  be a vector of length  $k$ . Compute  $C, \text{aux} \leftarrow \text{VC.Commit}(v)$ .

$\text{GetMemWit}(S, A, x)$  : Compute  $C, \text{aux}$  from  $S$  as above. Let  $i$  be such that  $x = s_i$ . Compute  $\pi_i \leftarrow \text{VC.Open}(x, i, \text{aux})$  and output  $(i, \pi_i)$ .

$\text{MemVer}(A, x, (i, \pi_i))$  : Output  $\text{VC.Ver}(A, x, i, \pi_i)$ .

*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  $\pi_i, \pi'_i$  that the vector committed to by  $A$  has  $x$  and  $x'$  respectively as its  $i^{\text{th}}$

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

**PROOF.** Suppose that  $\Pr \left[ \mathcal{G}_{\mathcal{A}, \text{Acc}}^{\text{insert}}(\lambda) = 1 \right]$  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

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

for some constants  $c_1, c_2 > 0$ .

Consider drawing  $\text{pp} \leftarrow \text{Setup}(\lambda)$  and running  $\mathcal{A}(\text{pp})$  to obtain  $A$ . As stated above, with non-negligible probability, there exists some  $i$  such that with non-negligible probability given this choice of  $\text{pp}, A$  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  $\pi_1, \pi_2$  that the  $i^{\text{th}}$  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  $\pi_1, \pi_2$  that the same component takes on two distinct values, contradicting position binding of the vector commitment.  $\square$

## 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. In practice, each offers different trade-offs which might be attractive for different applications.

**Merkle trees.** Merkle trees are optimal in terms of the commitment size (32 bytes) and require no trusted setup or public parameters. 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 tree 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

| Scheme               | Trusted setup?   | commitment  (bytes) | Witness size |                                 | Public params  (asyp.) | Witness gen. time (asyp.) |
|----------------------|------------------|---------------------|--------------|---------------------------------|------------------------|---------------------------|
|                      |                  |                     | (asyp.)      | (bytes)                         |                        |                           |
| Merkle tree          | no               | 32                  | $O(\log n)$  | $32 \cdot \lceil \log n \rceil$ | $O(1)$                 | $O(n \log n)$             |
| RSA Accumulator      | yes <sup>†</sup> | 384                 | $O(1)$       | 384                             | $O(1)$                 | $O(n^2)$                  |
| Bilinear Accumulator | yes              | 48                  | $O(1)$       | 48                              | $O(n)$                 | $O(n \log n)$             |
| Hyperproofs [50]     | yes              | 48                  | $O(\log n)$  | $48 \cdot \lceil \log n \rceil$ | $O(n)$                 | $O(n \log n)$             |

**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. <sup>†</sup> RSA accumulators can be instantiated using class groups [40], which do not require trusted setup. We report numbers here for the classic RSA group  $\mathbb{Z}_N^*$ .**

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, 40], 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 [44, 54], 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, the only pairing operation required is a single operation done by the verifier.

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 [35, 42]. 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 [25]. Ethereum generated a smaller string with  $2^{12}$  powers of tau in a community setup [26]. While the coordinator must store this entire structured reference string, participants need only store  $O(1)$  terms from this string to verify that their contributions were included.

**Hyperproofs.** Finally, Hyperproofs [50] 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 [50]. Verifying witnesses takes on the order of milliseconds. This efficiency is immediately inherited by the accumulator constructed using our approach

in Section 5.6. 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

We introduced Cornucopia, a simple but powerful framework for VDF-based DRBs, using accumulators to construct participatory randomness beacon protocols at massive scale. Our work shows that this paradigm is secure, and it can be instantiated with practically efficient accumulators. We discussed the efficiency of common accumulator constructions in Section 6. We note that there is no obvious accumulator construction that is superior performance-wise in all scenarios. We further note that the performance bottleneck in practice for very large deployments (e.g. millions or billions of users) is likely to be inclusion proof generation by the coordinator. Constructing an accumulator of a large set and batch-computing all witnesses appears to be an under-studied problem; our work might serve as motivation to revisit accumulator constructions with this goal in mind.

We discuss two possible extensions to the Cornucopia framework, leaving a complete analysis to future work.

**Public verifiability.** As proposed, Cornucopia only offers meaningful security guarantees to participants who themselves contributed randomness to the protocol. Passive observers will have no idea if the coordinator actually included any honest participant’s values in the published commitment.

We can provide a slightly weaker security guarantee to purely passive participants by introducing a subset of *notarized participants* with some public reputation for honesty. These participants may be organizations such as nonprofits or government bodies who commit to participating in the protocol regularly. Each notarized participant, after verifying its inclusion proof showing that its contribution was included by the coordinator in the accumulator value, signs the accumulator value. These signatures might be collected by the coordinator or posted to the public bulletin board. To save space, they can be compressed using a signature scheme such as BLS that supports succinct multi-signatures [11], resulting in only  $O(1)$  additional overhead.

Any observer 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 the VDF output is valid, the output of Cornucopia must be secure. In practice, using BLS multi-signatures, this would be about as efficient to verify as a

state-of-the-art honest majority protocol like *drand* [23], while offering much stronger security (any honest notarized participant vs. a majority of honest nodes in *drand*).

Note that this discussion assumes that notaries are perfectly reliable and never go offline. Tolerating a minority of offline notaries requires relaxing the assumptions further, but not necessarily to an honest majority assumption. Tolerating  $k$  notaries going offline requires trusting that at least  $k + 1$  notaries are honest.

**Improving liveness with multiple coordinators.** A malicious coordinator can prevent individuals from contributing to the protocol, or even withhold the commitment  $R$  and prevent the protocol from finishing at all. The coordinator can't do so conditionally based on the impending outcome, but they can try to block all honest participants. As noted, the coordinator is trusted for availability, but not for security.

A natural way to mitigate denial-of-service is to introduce multiple coordinators, each of which posts a commitment  $R_i$ . The final beacon output is then computed as  $\Omega = \text{Delay}(\text{Combine}(R_1, \dots, R_n))$ , passing a the concatenation of these commitments to the VDF. Note that in the limit, every user might in fact be their own coordinator, in which case the protocol is exactly the original Unicorn proposal [37]. This makes it easy to see, informally, that extra malicious coordinators cannot undermine security of the protocol as long as the VDF is secure.

Indeed, there is no *security* reason to limit the number of coordinators, only efficiency considerations. It would be possible in a distributed setting, for example, to enable any party 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. Now, as long as at least one coordinator posts a commitment that has at least one honest randomness contribution, the beacon output is unpredictable. Users can submit contributions to multiple coordinators and trust the final output  $\Omega$  as long as at least one coordinator includes their contribution.

Another benefit of this multi-coordinator design is that coordinators can use different accumulators. This allows users to choose their desired efficiency trade-off. For example, a user participating across many epochs may prioritize shorter witnesses and opt for the bilinear accumulator with its constant-sized witnesses. Another user who participates only once may opt for a coordinator using a Merkle tree, requiring an  $O(\log n)$ -sized witness, which is a small one-time cost, and avoiding the need for a trusted setup.

**Incentives.** Finally, while we note that analyzing incentives in public randomness generation is an important open problem, not just for Cornucopia-style protocols but for DRBs in general. First, it is necessary in Cornucopia to incentivize the coordinator(s) to provide a highly reliable service and expend non-trivial effort computing inclusion proofs. This problem 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, it is necessary to 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.

**Acknowledgments.** The authors thank Noemi Glaeser for suggesting the name Cornucopia. Miranda Christ was 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. Joseph Bonneau and Kevin Choi were supported by DARPA Agreement HR00112020022 and NSF grant CNS-2239975. Joseph Bonneau was additionally 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, or any other supporting organization.

## REFERENCES

- [1] Andrychowicz, M., Dziembowski, S., Malinowski, D., Mazurek, L.: Secure Multi-party Computations on Bitcoin. In: IEEE Security & Privacy (2014)
- [2] Bacho, R., Lenzen, C., Loss, J., Ochseneither, S., Papachristoudis, D.: GrandLine: First Adaptively Secure DKG and Randomness Beacon with (Almost) Quadratic Communication Complexity. Cryptology ePrint Archive, Paper 2023/1887 (2023)
- [3] Ben-Or, M., Linnal, N.: Collective coin flipping, robust voting schemes and minima of *banzhaf* values. In: FOCS (1985)
- [4] Ben-Or, M., Linnal, N.: Collective coin flipping. Advances in Computing Research (1989)
- [5] Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Eurocrypt (1993)
- [6] Bhat, A., Kate, A., Nayak, K., Shrestha, N.: OptRand: Optimistically responsive distributed random beacons. Cryptology ePrint Archive, Paper 2022/193 (2022)
- [7] Bhat, A., Shrestha, N., Kate, A., Nayak, K.: RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication. Cryptology ePrint Archive, Paper 2020/1590 (2020)
- [8] Blum, M.: Coin flipping by telephone a protocol for solving impossible problems. ACM SIGACT News (1983)
- [9] Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable Delay Functions. In: CRYPTO (2018)
- [10] Boneh, D., Bünz, B., Fisch, B.: A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Paper 2018/712 (2018)
- [11] Boneh, D., Drijvers, M., Neven, G.: Compact multi-signatures for smaller blockchains. In: Asiacrypt (2018)
- [12] Buchmann, J., Hamdy, S.: A survey on IQ cryptography. In: Public-Key Cryptography and Computational Number Theory (2011)
- [13] Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO (2002)
- [14] Cascudo, I., David, B.: Albatross: publicly attestable batched randomness based on secret sharing. In: Asiacrypt (2020)
- [15] Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC (2013)
- [16] Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkatasubramanian, M., Wang, R.: Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. In: IEEE Security & Privacy (2021)
- [17] Cherniaeva, A., Shirobokov, I., Shlomovits, O.: Homomorphic encryption random beacon. Cryptology ePrint Archive, Paper 2019/1320 (2019)
- [18] Choi, K., Arun, A., Tyagi, N., Bonneau, J.: Bicorn: An optimistically efficient distributed randomness beacon. In: Financial Crypto (2023)

- [19] Choi, K., Manoj, A., Bonneau, J.: SoK: Distributed randomness beacons. In: IEEE Security & Privacy (2023)
- [20] Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: TOC (1986)
- [21] Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. In: IEEE Security & Privacy (2022)
- [22] Dodis, Y.: Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping. In: ECC (2000)
- [23] Drand. <https://drand.love/>
- [24] Feist, D.: RSA Assumptions. [rsa.cash/rsa-assumptions/](https://rsa.cash/rsa-assumptions/) (2022)
- [25] FileCoin: Trusted setup complete! (2020), <https://filecoin.io/blog/posts/trusted-setup-complete/>
- [26] Foundation, E.: Proto-danksharding (2023), <https://www.eip4844.com/>
- [27] Fuchsbaue, G., Kiltz, E., Loss, J.: The algebraic group model and its applications. In: CRYPTO (2018)
- [28] Gabizon, A., Williamson, Z.J., Ciobotaru, O.: PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953 (2019)
- [29] Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: Euro S&P (2021)
- [30] Goldwasser, S., Kalai, Y.T., Park, S.: Adaptively secure coin-flipping, revisited. In: ICALP (2015)
- [31] Guo, Z., Shi, L., Xu, M.: SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability. IEEE Access (2020)
- [32] Haitner, I., Karidi-Heller, Y.: A tight lower bound on adaptively secure full-information coin flip. In: FOCS (2020)
- [33] Kalai, Y.T., Komargodski, I., Raz, R.: A lower bound for adaptively-secure collective coin flipping protocols. *Combinatorica* **41**(1) (2021)
- [34] Kavousi, A., Wang, Z., Jovanovic, P.: SoK: Public Randomness. Cryptology ePrint Archive, Paper 2023/1121 (2023)
- [35] Kerber, T., Kiayias, A., Kohlweiss, M.: Mining for Privacy: How to Bootstrap a Snarky Blockchain. In: Financial Crypto (2021)
- [36] Lee, H., Hsu, Y., Wang, J.J., Yang, H.C., Chen, Y.H., Hu, Y.C., Hsiao, H.C.: HeadStart: Efficiently Verifiable and Low-Latency Participatory Randomness Generation at Scale. In: NDSS (2022)
- [37] Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366 (2015)
- [38] Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: ACNS (2007)
- [39] Lipmaa, H.: Secure accumulators from Euclidean rings without trusted setup. In: ACNS (2012)
- [40] Long, L.: Binary quadratic forms. <https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf> (2018)
- [41] Nguyen, L.: Accumulators from bilinear pairings and applications. In: CT-RSA (2005)
- [42] Nikolaenko, V., Ragsdale, S., Bonneau, J., Boneh, D.: Powers-of-tau to the people: Decentralizing setup ceremonies. In: ACNS (2024)
- [43] Papamanthou, C.: Cryptography for efficiency: new directions in authenticated data structures. Ph.D. thesis, Brown University (2011)
- [44] Pietrzak, K.: Simple Verifiable Delay Functions. In: ITCS (2018)
- [45] Qian, Y.: Randao: Verifiable random number generation. [randao.org/whitepaper/Randao\\_v0.85\\_en.pdf](https://randao.org/whitepaper/Randao_v0.85_en.pdf) (2017)
- [46] Rabin, M.O.: Transaction protection by beacons. *Journal of Computer and System Sciences* (1983)
- [47] Raikwar, M., Gligoroski, D.: SoK: Decentralized randomness beacon protocols. In: Australasian Conference on Information Security and Privacy (2022)
- [48] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.: RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In: NDSS (2023)
- [49] Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Efficient continuous distributed randomness. In: IEEE Security & Privacy (2020)
- [50] Srinivasan, S., Chepur, A., Papamanthou, C., Tomescu, A., Zhang, Y.: Hyperproofs: Aggregating and maintaining proofs in vector commitments. In: USENIX Security (2022)
- [51] Srinivasan, S., Karantaidou, I., Baldimtsi, F., Papamanthou, C.: Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In: ACM CCS (2022)
- [52] 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)
- [53] Wang, W., Ulichney, A., Papamanthou, C.: {BalanceProofs}: Maintainable Vector Commitments with Fast Aggregation. In: USENIX Security (2023)
- [54] Wesolowski, B.: Efficient Verifiable Delay Functions. In: Eurocrypt (2019)
- [55] Yakira, D., Asayag, A., Grayevsky, I., Keidar, I.: Economically viable randomness. CoRR (2020)