Cornucopia: Distributed randomness beacons at scale

Miranda Christ[1], Kevin Choi[2], and Joseph Bonneau[2,3]

[1] Columbia University
`mchrist@cs.columbia.edu`
[2] New York University
`{kevin.choi,jcb}@cs.nyu.edu`
[3] a16z crypto research

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 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 and RSA accumulators) are naturally insertion-secure. Finally, we give a generic transformation from any universal accumulator (supporting non-membership proofs) to an insertion-secure accumulator, albeit with an efficiency loss proportional to the security parameter.

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 (denoted Ω) 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 distributed randomness is *commit-reveal* [7]. First, all participants publish a cryptographic commitment to a random value r_i. Participants then reveal their r_i values, and the result is $\Omega = \mathsf{Combine}(r_1, \ldots, r_n)$ for some suitable combination function (such as exclusive-or or a cryptographic hash). Commit-reveal protocols are simple, efficient, and secure as long as 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 the last participant to reveal observes all other r_i values and refuses to open if the impending value of Ω 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 Ω.

Most approaches to avoiding last-revealer attacks relax the security model by assuming an honest majority of participants, giving the majority a means to recover a withholding participant's contribution. However, this means that a malicious majority can privately compute Ω early and potentially bias it. Protocols of this type also often add significant overhead, with communication and computation costs superlinear in n (though some amortize this over multiple rounds). For details, we refer the reader to several surveys on DRBs [38,16,28].

A fundamentally different approach to constructing DRBs uses time-based cryptography, specifically using *delay functions* to prevent manipulation. The simplest example is Unicorn [30], 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 $\Omega = \mathsf{Delay}(\mathsf{Combine}(r_1, \ldots, r_n))$. By assumption, a participant cannot compute the Delay function before the deadline to publish their contribution r_i and therefore cannot predict Ω or choose r_i in such a way as to influence it. This protocol retains the strong $n-1$ (dishonest majority) security model of commit-reveal, but with no last-revealer attacks. It is remarkably simple and, using modern verifiable delay functions [8], the result can be efficiently verified. The downside is that $\Theta(n)$ elements must be posted to the public bulletin board per protocol run.

Our approach. We introduce Cornucopia, which retains the security advantages of Unicorn while reducing the overhead of storage (on the public bulletin board) from $\Theta(n)$ to $O(1)$. The essential idea is to use a cryptographic accumulator (for example, a Merkle tree) to publish a succinct commitment to all users' contributions. The overall structure of the protocol is as follows:

– Each participants 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 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.
– Users should check both that their contribution was included in R and that Ω was properly computed from R.

While this is a small change to Unicorn, it is powerful: individual users can now be convinced that Ω is truly random with only 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). Any number of other malicious participants in the protocol cannot undermine security.

This opens the door, for the first time, to massive *open-participation* randomness protocols. For example, every user buying a lottery ticket might contribute randomness, or every user in a massively multi-player 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.

To prove Cornucopia secure, we must develop a new security notion for accumulators called *insertion security*, which may be of independent interest. Fortunately, we prove that the most commonly used accumulator constructions either naturally feature insertion security (Merkle trees) or need only trivial modifications to achieve it (RSA accumulators), meaning Cornucopia is practical today.

2 Preliminaries

To define Cornucopia, we first need to define verifiable delay functions (VDFs) [8] and accumulators [4]. Both rely on public parameters pp which all functions take implicitly, though we will typically omit this for brevity. 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 use $\xleftarrow{\$}$ (or $\xrightarrow{\$}$) 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 where noted.

2.1 Verifiable Delay Functions

Definition 1 (Verifiable delay function [8]) *A* verifiable delay function *(VDF) [8] is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Eval}, \mathsf{Verify})$ *where:*

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

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

$\mathsf{VDF.Verify}(\mathsf{pp}, x, y, \pi) \to \{\mathsf{true}, \mathsf{false}\}$ *takes 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 $\mathsf{VDF.Eval}$.

Uniqueness. VDF evaluation must be a function, meaning that $\mathsf{VDF.Eval}$ is a deterministic algorithm and it is computationally infeasible to find two pairs $(x, y), (x, y')$ with $y \neq y'$ that $\mathsf{VDF.Verify}$ 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 [8]):

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

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

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

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

Fig. 1. VDF sequentiality (left) and accumulator security game (right).

Definition 2 (VDF sequentiality [8]) *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:*

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

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 [4,11]) *Given a data universe* U*, an* accumulator *[4] is a tuple of algorithms* ($\mathsf{Setup}, \mathsf{Accumulate}, \mathsf{GetMemWit}, \mathsf{MemVer}$) *where:*

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

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

$\mathsf{Acc.GetMemWit}(S, A, x) \to w$ *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*.*

$\mathsf{Acc.MemVer}(A, x, w) \to \{\mathsf{true}, \mathsf{false}\}$ *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.*

We describe here only the functionality necessary for our purposes; accumulators generally also support an Update function to add additional elements to the accumulated set and *dynamic* accumulators support a Delete function to remove elements [11]. An accumulator is *correct* if MemVer always accepts for elements included in honestly accumulated sets. 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 [11]) *An accumulator* Acc *is* secure *if no PPT adversary* $\mathcal{A}$ *can succeed with non-negligible probability in* $\mathcal{G}^{\text{acc}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$ *as defined in Figure 1.*

A *universal accumulator* [31] also supports non-membership proofs; that is, it supports two additional functions:

$\mathsf{Acc.GetNonMemWit}(S, A, x') \to w'$ 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'.

$\mathsf{Acc.NonMemVer}(A, x', w') \to \{\mathsf{true}, \mathsf{false}\}$ 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.

A universal accumulator is *correct* if, in addition to MemVer accepting for all included elements, $\mathsf{NonMemVer}$ 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 [31]) *A universal accumulator* Acc *is* secure *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\begin{bmatrix}\mathsf{pp} \xleftarrow{\$} \mathsf{Acc.Setup}(\lambda) \\ A, x, w, w' \xleftarrow{\$} \mathcal{A}(\mathsf{pp}) \\ \mathsf{Acc.MemVer}(A, x, w) \wedge \mathsf{Acc.NonMemVer}(A, x, w')\end{bmatrix} \leq \mathsf{negl}(\lambda)$$

3 Timed DRBs: Definitions and Constructions

We first define timed DRBs using a generalized syntax.[4]

Definition 6 (Timed DRBs) *A timed DRB protocol is a tuple of algorithms* $(\mathsf{Setup}, \mathsf{Prepare}, \mathsf{Post}, \mathsf{Finalize}, \mathsf{Verify})$ *as follows:*

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

$\mathsf{Prepare}(\mathsf{pp}) \xrightarrow{\$} r_i$*: 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*.*

$\mathsf{Post}(\{r_i\}) \to (R, \{\pi_i\})$*: 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.*

$\mathsf{Finalize}(\mathsf{pp}, R) \to (\Omega, \pi_\Omega)$*: 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*.*

[4] Note that our syntax here is specific to one-round timed DRBs. Some timed DRBs such as Bicorn [15] have an optional second communication round.

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

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

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

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

Fig. 2. Security games for (p, σ)-unpredictability and (p, σ)-indistinguishability.

$\mathsf{Verify}(\mathsf{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{\mathsf{true}, \mathsf{false}\}$*: 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.*

A timed DRB has the following security properties (shown in Figure 2):

Definition 7 ((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:* $\mathsf{Adv}^{\text{unpred}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{DRB}}(\lambda) = 1\right]$.

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

Definition 8 ((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* $\mathsf{Finalize}$ *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. [8], there is a generic transformation in the random oracle model in which a timed DRB which satisfies (p, σ)-unpredictability

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

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

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

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

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

Fig. 3. The Unicorn timed DRB protocol [30]

can be transformed generically into one with (p, σ)-indistinguishability by applying the random oracle to the output.

3.1 Unicorn

As a warm-up, we describe Unicorn [30] succinctly as a timed DRB in our framework in Figure 3. 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 [8].

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 provide a formal security proof for Unicorn (in a slightly different model) [30]. We omit one here but note that its security is implied by our security proof for Cornucopia in Theorem 1, as Unicorn is a special case using the "concatenation accumulator" which simply concatenates all accumulated values together.

The primary downside of Unicorn is communication efficiency. As every user's value is posted, we have $|R| = \Theta(n)$. 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 improves on Unicorn by having the coordinator accumulate all user contributions into a succinct commitment R using a cryptographic accumulator scheme (see Section 2.2). Because $|R|$ does not grow with the number of participants, Cornucopia makes it easy to scale to many users with low publishing

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

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

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

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

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

Fig. 4. The Cornucopia protocol

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

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

Fig. 5. Insertion security game

costs and low costs for users to verify that the beacon output Ω 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 prove a novel security property for accumulators: *insertion security*:

Definition 9 *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:*

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

To win the insertion security game (Figure 5), 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 $\mathsf{Accumulate}$ 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. This is because 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 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. In this proof, we allow the reduction to program the random oracle as is standard in the random oracle model.

Lemma 1. *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 Cornucopia (CC) is instantiated with an insertion-secure accumulator, then* $\Pr[\mathcal{E}_1] \leq \mathsf{negl}(\lambda)$.

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

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

We define an adversary $\mathcal{B}$ that breaks insertion security of the accumulator scheme by simulating the challenger in $\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\mathsf{CC}}$ 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 $q_1, \ldots, q_k$ 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 $\frac{1}{\lambda^c}$, $\Pr[\mathsf{MemVer}(R, x, w_1) = \mathsf{true} \wedge \mathcal{A}_0 \text{ queried } R \text{ to the random oracle}] \geq \frac{1}{\lambda^c}$. 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 $\mathcal{A}$, $\Pr[\mathsf{MemVer}(R, x, w_1) = \mathsf{true} \wedge A = R] \geq \frac{1}{\lambda^c} \cdot \frac{1}{q(\lambda)}$ which is non-negligible. Thus, with non-negligible probability, $\mathcal{G}^{\text{insert}}_{\mathcal{B},\mathsf{Acc}}(\lambda) = 1$. □

Lemma 2. *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* $\Pr[\mathcal{E}_2] \leq \mathsf{negl}(\lambda)$.

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[5] 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

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

[5] 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.

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

Corollary 1. *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 9). We consider Merkle trees, RSA accumulators and a generic scheme here, we discuss other accumulators in Section 6.

5.1 Accumulators without insertion security

Given any secure accumulator scheme Acc, it is trivial to 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. $\mathsf{Acc}'.\mathsf{MemVer}(A, x, w)$ is defined to be 1 if $A = \epsilon$ (regardless of the value of x or w), and otherwise is equal to $\mathsf{Acc}.\mathsf{MemVer}(A, x, w)$. 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. 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.

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 $w = \mathsf{Acc}.\mathsf{GetMemWit}(S, A, H(x, y))$ 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.[6]

5.2 Merkle trees

Lemma 3. *Any Merkle tree of bounded depth $k = \mathsf{poly}(n)$ using a hash function modeled as a random oracle outputting $\geq \lambda$ bits is insertion-secure.*

Proof. We work in the random oracle model, supposing that the Merkle tree uses a random oracle $\mathcal{O} : \{0,1\}^{2n} \to \{0,1\}^n$. Suppose for the sake of contradiction that an adversary $\mathcal{A}$ succeeds in constructing A such that with probability at least $\frac{1}{\mathsf{poly}(\lambda)}$, for a random $x \in \{0,1\}^n$, the adversary can provide a verifying witness $w = (w_1, \ldots, w_k)$ for x. That is, $\mathcal{O}(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x))) = A$. 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 $\mathcal{G}^{\text{insert}}_{\mathcal{A},\mathsf{Acc}}(\lambda)$.

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, $b_i = w_i || \mathcal{O}(w_{i-1} || \ldots)$. Observe first that $\Pr[b_1 = q_j \text{ for some j}] = \frac{\ell}{2^\lambda}$ since x is chosen at random. Assume that the probability that b_i is equal to any q_j is at most $\frac{i \cdot \ell}{2^\lambda}$. If this event does not occur, then $b_{i+1} = \mathcal{O}(b_i || \mathcal{O}(b_{i-1}))$ is a freshly random value, and the probability that $b_{i+1} = q_j$ for any j is $\frac{\ell}{2^\lambda}$.

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

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

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, $\mathcal{O}(w_k || \ldots \mathcal{O}(w_2 || \mathcal{O}(w_1 || x)))$ is a uniformly random value independent of A and equals A with only negligible probability. □

5.3 RSA accumulators

In a standard RSA accumulator [11,32], $\mathsf{Setup}(\lambda)$ generates a random group of unknown order and a generator g for this group using some group generation

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

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 $A^a B^x = g$.

We make a small modification to make RSA accumulators insertion-secure: $\mathsf{MemVer}(A, x, w)$ now outputs 1 if and only if $w^x = A$ and $A \neq 1$. This requirement that $A \neq 1$ is necessary for our proof of insertion security; observe that if the adversary outputs an accumulator value $A = 1$, then for all x it is easy to find w such that $w^x = A$: just let $w = 1$. This matches the requirement in the Adaptive Root Assumption that $u^l = v \neq 1$.

Assumption 1 (Adaptive Root Assumption [9])

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

Lemma 4. *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 $\mathsf{InsertGame}$ 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 $\mathsf{InsertGame}$, 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. $\square$

Corollary 2. *The modified RSA accumulator is insertion-secure in the Algebraic Group Model (AGM), since the Adaptive Root Assumption holds in the AGM [21].*

5.4 From generic universal accumulators

Finally, we show how to construct an insertion-secure accumulator scheme Acc' from any universal accumulator scheme Acc, albeit one only offering computational correctness. The core idea 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' in 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 S_x^+ is wrongly excluded with probability $\frac{1}{2}$ and each item 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 $U' = \{0,1\}^\lambda$, although one can use any superpolynomially-large set that supports the following hash function H. Let $H : [\lambda] \times U' \to U$ (where $[\lambda]$ denotes $\{1, \dots, \lambda\}$) be a hash function that we will model as a random oracle. For any $x \in U'$, let $S_x^+ := \left\{y \ : \ H(i,x) = y \text{ for } i \in [\frac{\lambda}{2}]\right\}$, and let $S_x^- := \left\{y \ : \ H(i,x) = y \text{ for } i \in \left\{(\frac{\lambda}{2}+1), \dots, \lambda\right\}\right\}$ (assume for convenience that λ is even).

Setup: Acc' uses the same setup function as Acc.

$\mathsf{Accumulate}$: Let $S' \subseteq U'$. Let $S = \bigcup_{x \in S'} S_x^+$. We define $\mathsf{Acc}'.\mathsf{Accumulate}(S')$ to output $A = \mathsf{Acc}.\mathsf{Accumulate}(S)$.

$\mathsf{GetMemWit}$: $\mathsf{Acc}'.\mathsf{GetMemWit}(S', A, x)$ outputs a vector of witnesses $\mathbf{w}$ of length λ where:
- For $i \leq \frac{\lambda}{2}$, $w_i = \mathsf{Acc}.\mathsf{GetMemWit}(S, A, H(i,x))$ is a membership proof for $H(i,x)$
- For $i > \frac{\lambda}{2}$, $w_i = \mathsf{Acc}.\mathsf{GetNonMemWit}(S, A, H(i,x))$ is a non-membership proof for $H(i,x)$

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

Lemma 5. *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 $y_1, \dots, y_{\frac{\lambda}{2}}$ should be included, and $y_{\frac{\lambda}{2}+1}, \dots, y_\lambda$ should be excluded.

Equivalently, we can think of drawing $\mathbf{y} = y_1, \ldots, y_\lambda$ (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, an honestly-generated membership proofs for elements in that set should verify for MemVer; similarly, honestly-generated non-membership proofs for elements not in that set should verify for $\mathsf{NonMemVer}$. 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. This might be problematic for Cornucopia if, for example, one user chooses x_1 and expects a membership proof for y and another user chooses x_2 and expects a non-membership proof for y. 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 Other accumulators

Accumulators can also be constructed from bilinear pairings [34,41]. While we conjecture these are insertion-secure, they require public parameters linear in the maximum number of items that can be accumulated. Hence we consider them a poor fit for Cornucopia, where we hope to support millions of participants. Similarly, we might try to build accumulators from polynomial commitments [27], but the popular KZG construction also requires linear-sized public parameters.

Vector commitments for exponentially large message spaces [13] can be used to construct an insertion-secure accumulator for sets of bounded size $\leq s$. The data universe consists of index-message pairs (i, m). To accumulate a list of index-message pairs (i, m) for distinct indices i, we generate a commitment to the vector that sets index i to m for each pair in the list. The membership proof for (i, m) is an opening proof for (i, m). The accumulator value A is the vector commitment value.

Note that binding property of vector commitments[7] ensures that for any i', it is computationally feasible to find only one m^* such that a proof of opening for (i', m') holds with respect to the commitment value A. For a randomly chosen m', the probability that $m' = m^*$ is negligible (since the message space is exponentially large). Therefore, the adversary can provide a membership proof for a random (i', m') with only negligible probability.

However, this accumulator from vector commitments does not satisfy correctness, since users can request (i, m) and (i, m') for $m \neq m'$. This issue can be easily solved for use in Cornucopia by assigning each user a unique index. It is tempting to use the hash function trick from Section 5.4 to obtain computational correctness by letting (i, m) be computed as the output of a hash function on a λ-length binary string. However, the space of possible indices i has polynomial size and it is not hard to find two inputs to the hash function yielding (i, m) and (i, m') for different m, m'.

7 Related Work

There is a large and growing literature on randomness beacons, dating to the seminal proposal by Rabin [37] and foundational work on *distributed coin tossing* [17,2,3,23,19,26,25]. Several recent surveys cover modern DRBs [38,16,28]. Most of this work is orthogonal, working without the benefit of delay functions and hence assuming honest majorities [42,14,12,6,22,24,40,18,5,20] or economic

[7] Binding says that it is infeasible for a PPT adversary to come up with *any* vector commitment C, distinct messages m, m', an index i, and accepting proofs π, π' for (i, m) and (i, m').

incentives [1,36,44]. Unicorn [30] introduced delay-based DRBs. Several extensions to Unicorn work in a similar model to ours. Bicorn [15] extends Unicorn with a fast optimistic case, avoiding the delay function if *all* participants are honest. RandRunner [39] 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. Finally, HeadStart [29] is perhaps the most similar to ours, using Merkle trees and a multi-round pipelined protocol to scale up Unicorn. Our work primarily differs in offering a generic construction from any accumulator and developing precise security notions required of accumulators for use with DRBs.

8 Concluding Discussion

We introduce Cornucopia, a simple but powerful framework for VDF-based DRBs, using accumulators to scale to millions of users in participatory randomness beacon protocols. Our work shows that this paradigm is secure, but leaves open many practical questions about optimal deployments. First, there is an interesting performance trade-off between the Merkle trees and RSA accumulators. RSA accumulators offer constant-sized proofs which should be shorter than Merkle tree proofs for more than $\approx 2^{10}$ users, a regime we hope to support. However, RSA proofs are much more expensive for the coordinator to generate, which may become prohibitive for very large deployments. RSA proofs are 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. There may also be interesting optimizations when combining RSA accumulators with RSA-based VDFs [35,43], such as offering a single combined proof of membership and correct exponentiation. Finally, practical RSA deployments must consider which group to use [9], with traditional RSA groups $(\mathbb{Z}/N)^*$ offering better concrete performance and better-understood security parameters, but class groups of imaginary quadratic order [10,33] avoiding the need for trusted setup.

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 Multiparty Computations on Bitcoin. In: IEEE Security & Privacy (2014)
2. Ben-Or, M., Linial, N.: Collective coin flipping, robust voting schemes and minima of banzhaf values. In: FOCS (1985)
3. Ben-Or, M., Linial, N.: Collective coin flipping. Advances in Computing Research (1989)
4. Benaloh, J., De Mare, M.: One-way accumulators: A decentralized alternative to digital signatures. In: Eurocrypt (1993)
5. Bhat, A., Kate, A., Nayak, K., Shrestha, N.: OptRand: Optimistically responsive distributed random beacons. Cryptology ePrint Archive, Paper 2022/193 (2022)
6. Bhat, A., Shrestha, N., Kate, A., Nayak, K.: RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication. Cryptology ePrint Archive, Paper 2020/1590 (2020)
7. Blum, M.: Coin flipping by telephone a protocol for solving impossible problems. ACM SIGACT News (1983)
8. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: CRYPTO (2018)
9. Boneh, D., Bünz, B., Fisch, B.: A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Paper 2018/712 (2018)
10. Buchmann, J., Hamdy, S.: A survey on IQ cryptography. In: Public-Key Cryptography and Computational Number Theory (2011)
11. Camenisch, J., Lysyanskaya, A.: Dynamic accumulators and application to efficient revocation of anonymous credentials. In: CRYPTO (2002)
12. Cascudo, I., David, B.: Albatross: publicly attestable batched randomness based on secret sharing. In: Asiacrypt (2020)
13. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public-Key Cryptography–PKC 2013: 16th International Conference on Practice and Theory in Public-Key Cryptography, Nara, Japan, February 26–March 1, 2013. Proceedings 16. pp. 55–72. Springer (2013)
14. Cherniaeva, A., Shirobokov, I., Shlomovits, O.: Homomorphic encryption random beacon. Cryptology ePrint Archive, Paper 2019/1320 (2019)
15. Choi, K., Arun, A., Tyagi, N., Bonneau, J.: Bicorn: An optimistically efficient distributed randomness beacon. In: Financial Crypto (2023)
16. Choi, K., Manoj, A., Bonneau, J.: Sok: Distributed randomness beacons. In: IEEE Security & Privacy (2023)
17. Cleve, R.: Limits on the security of coin flips when half the processors are faulty. In: TOC (1986)
18. Das, S., Krishnan, V., Isaac, I.M., Ren, L.: Spurt: Scalable distributed randomness beacon with transparent setup. Cryptology ePrint Archive, Paper 2021/100 (2021)
19. Dodis, Y.: Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping. In: ECCC (2000)
20. Drand. `https://drand.love/`
21. Feist, D.: Rsa assumptions. `rsa.cash/rsa-assumptions/` (2022)
22. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. IACR Cryptol. ePrint Arch. **2020** (2020)
23. Goldwasser, S., Kalai, Y.T., Park, S.: Adaptively secure coin-flipping, revisited. In: ICALP (2015)

24. Guo, Z., Shi, L., Xu, M.: SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability. IEEE Access (2020)
25. Haitner, I., Karidi-Heller, Y.: A tight lower bound on adaptively secure full-information coin flip. In: FOCS (2020)
26. Kalai, Y.T., Komargodski, I., Raz, R.: A lower bound for adaptively-secure collective coin flipping protocols. Combinatorica **41**(1) (2021)
27. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: Asiacrypt (2010)
28. Kavousi, A., Wang, Z., Jovanovic, P.: SoK: Public Randomness. Cryptology ePrint Archive, Paper 2023/1121 (2023)
29. 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)
30. Lenstra, A.K., Wesolowski, B.: A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Paper 2015/366 (2015)
31. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: ACNS (2007)
32. Lipmaa, H.: Secure accumulators from euclidean rings without trusted setup. In: ACNS (2012)
33. Long, L.: Binary quadratic forms. `https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf` (2018)
34. Nguyen, L.: Accumulators from bilinear pairings and applications. In: CT-RSA (2005)
35. Pietrzak, K.: Simple Verifiable Delay Functions. In: ITCS (2018)
36. Qian, Y.: Randao: Verifiable random number generation. `randao.org/whitepaper/Randao_v0.85_en.pdf` (2017)
37. Rabin, M.O.: Transaction protection by beacons. Journal of Computer and System Sciences (1983)
38. Raikwar, M., Gligoroski, D.: SoK: Decentralized randomness beacon protocols. In: Australasian Conference on Information Security and Privacy (2022)
39. Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., Weippl, E.: RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In: NDSS (2023)
40. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Hydrand: Efficient continuous distributed randomness. In: IEEE Security & Privacy (2020)
41. Srinivasan, S., Karantaidou, I., Baldimtsi, F., Papamanthou, C.: Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In: ACM CCS (2022)
42. 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)
43. Wesolowski, B.: Efficient Verifiable Delay Functions. In: Eurocrypt (2019)
44. Yakira, D., Asayag, A., Grayevsky, I., Keidar, I.: Economically viable randomness. CoRR (2020)