Distributed Randomness in Adversarial Settings

by

Kevin Choi

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

August, 2025

Professor Joseph Bonneau

Dedication

To my family: my father, my mother, my older brother, and my younger sister.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to my advisor, Joe. His anchored support and insights as a mentor have always guided me through the many ups and downs of my PhD journey, for which I am forever grateful. Joe was also generous enough to lead me through many memorable experiences, such as outdoor climbing at the Gunks, touring San Francisco on nighttime trolleys, and the New Paltz retreat. His generosity, both inside and outside of school, is something I will always remember.

I would also like to extend my gratitude to the rest of my dissertation committee: Yevgeniy Dodis, Benedikt Bünz, Alin Tomescu, and Philipp Jovanovic. Their presence on the committee and openness to help mean a lot to me. Furthermore, I would like to thank all other outstanding professors at NYU from whom I had the privilege of learning, including Oded Regev, Subhash Khot, Davi Geiger, Anasse Bari, Ralph Grishman, Aurojit Panda, Anirudh Sivaraman, Rajesh Ranganath, Michael Walfish, and Lakshmi Subramanian.

Next, I would like to acknowledge my co-authors and other collaborators: Arasu Arun, Miranda Christ, Aathira Manoj, Nirvan Tyagi, Walter McKelvie, Tal Malkin, Brad Windsor, Zachary DeStefano, Chelsea Komlo, Benedikt Bünz, Rachit Garg, Benedikt Wagner, Faxing Wang, William Wang, and Noah Golub. In particular, Arasu, I feel lucky to have had you as my academic sibling who started around the same time as I did. While it was sometimes a bit of a challenge to get you out of Stuytown for us to do something together, it was always rewarding to convince you to hang out, take our photos together at Yankee Stadium, and, most importantly, go through the

ups and downs together. Miranda, it was such a joy whenever you invited me to climbing, and I am glad we were able to spend time together in different contexts. Zach, I am grateful for the many fond memories we shared during PhD. Faxing, somehow you ended up at NYU, and it was always great to see you in the office and talk about crypto and life.

In addition to academic conversations, Will and Noah have been my muscle-up collaborators. As each year goes by, I am reminded that health is wealth, and after my knee surgery in 2024, my body was not in the best shape. Will, thank you for starting your muscle-up journey with me, which helped mine after a hiatus, and Noah, thank you for sharing your cultural, historical, and musical knowledge with me along the way.

A huge thank you goes to my officemates, Jacob Salzberg and Alexandre Moine, without whom I would not be able to hear words like Coq/Roq, Lean, and separation logic. Jacob, I also appreciate your wisdom on kefir—thank you. Alexandre, merci beaucoup pour nos conversations sur la musique. On that note, I would also like to thank my first officemate, Cheng Tan, when I first arrived at NYU, as well as John Westhoff. Further thanks to other building mates at various points in time: Xiangyu Gao, Haseeb Ashfaq, Shiva Iyer, Taegyun Kim, Fabian Ruffy, Betty Li Hou, and Derek Yen. Thanks also to Sihyun Lee and the Sihyun Bar.

Outside of NYU, I would like to acknowledge Prashanth Ramakrishna and Alireza Kavousi for meaningful conversations, Magdalena Stern-Baczewska and Beth Pratt for their heartening support from Columbia, Gina Lee for artistic inspirations, Sangah Park for the yoga journey, and Minseok Baik and Hobin Kim for many memories in Korea.

Having done my undergraduate studies in NYC as well, I am privileged to be surrounded by friends I have known for years. A huge thank you to maestros Augusto Ghiotto and Javier Llaca; my life would not be the same without you both in many respects. I thoroughly appreciate our respective travel memories (e.g. to Korea, Japan, or Mexico) in addition to countless memories since college. Further thanks to Jiajia Zhao, Barbie Matthews, Binna Han, Jorge Rojas Zamalloa, Dajung Yoo, and Helen Yang.

Finally, thanks to the Mashpotato Pizza group chat with Javier, Jorge, Salvador, and Souren. I always have the best Mexican food when I am with the group.

Abstract

Distributed randomness in adversarial settings concerns the problem of jointly computing a random output in a network of mutually untrusting participants such that the output is not predictable or biasable by any participant or any coalition of participants. A distributed randomness beacon (DRB) is a service that periodically emits random outputs through such distributed randomness protocols and has found applications in cryptographically verifiable lotteries and gaming as well as leader election in distributed systems and consensus algorithms. In the past decade, the landscape of DRBs has evolved, with many DRB protocols relying on ad hoc heuristics rather than structured design principles. While this bottom-up approach has led to interesting integrations of cryptographic techniques, establishing a unifying framework of DRBs has remained open prior to this work. Similarly, the consideration of security properties of DRBs, such as unbiasability and unpredictability, has typically been restricted to specific settings.

This dissertation seeks to address these gaps by adopting a top-down approach to realizing a distributed randomness beacon. We conceptualize the broader design space of DRBs, introduce comprehensive security definitions applicable to all DRBs, and consider a variety of practical deployment scenarios. Simultaneously, we compare protocols based on their communication and computational efficiency and also highlight the functionality of various cryptographic building blocks in light of DRBs rather than solely focusing on their technical details.

Furthermore, we shed light on the security gap that exists between theoretical models and real-world scenarios, where most theoretical DRBs rely on the honest majority assumption (net-

work assumption that more than half of the nodes are honest) which has shown to break down in practice (e.g. the $625 million Axie Infinity's Ronin hack in 2022). Recognizing this issue, we propose two new optimized DRB protocols—Bicorn and Cornucopia—that offer robustness even in the presence of a dishonest majority. Bicorn is a "commit-reveal-recover" protocol designed to recover withheld values through special-purpose timed commitments. In its optimistic case, Bicorn is reminiscent of a classic commit-reveal protocol and thus remains efficient despite the dishonest majority setting. From a different angle, Cornucopia explores the possibility of achieving sublinear verification cost per beacon output in a dishonest majority setting and offers an affirmative solution. As the amount of data posted to the public bulletin board (in a public bulletin board model) is reduced from prior work's linear to constant, Cornucopia is in fact able to enjoy a lower communication complexity in general. We achieve this using cryptographic accumulators and define a novel security property of accumulators called insertion security, to handle the case of a malicious coordinator.

Through these contributions, we aim to advance the understanding and implementation of distributed randomness beacons in light of cryptography and beyond, offering current insights, recent progress, and directions for future research.

Contents

Dedication iii

Acknowledgments iv

Abstract vii

List of Figures xiv

List of Tables xv

1 Introduction 1

1.1 Historical Context . . . 2

1.2 Organization . . . 5

1.3 Statement of Work . . . 6

2 Preliminaries 9

2.1 Verifiable delay functions . . . 9

2.2 Accumulators . . . 11

2.3 Vector commitments . . . 13

2.4 Algebraic group model . . . 14

2.5 Groups of unknown order and RSW assumptions . . . 15

2.6 Randomizing exponent sizes . . . 17

2.7 Non-interactive zero-knowledge proofs . . . 17

2.8 Verifiable Secret Sharing (VSS) . . . 19
2.9 Distributed Key Generation (DKG) . . . 20
2.10 Publicly Verifiable Secret Sharing (PVSS) . . . 21
2.11 Verifiable Random Function (VRF) . . . 23
2.12 Distributed Verifiable Random Function (DVRF) . . . 24
2.13 DDH-DVRF . . . 25
2.14 GLOW-DVRF . . . 25
2.15 Dfinity-DVRF . . . 26
2.16 Other Cryptographic Primitives . . . 26
2.16.1 Lagrange Interpolation . . . 26
2.16.2 BLS Signature . . . 27
2.16.3 NIZK of Discrete Logarithm Equality (DLEQ) . . . 28

3 Design Space of Distributed Randomness Beacons 29

3.1 Context . . . 29
3.1.1 System Model . . . 29
3.1.2 Strawman Protocol: Rock-Paper-Scissors . . . 30
3.1.3 Commit-Reveal . . . 31
3.1.4 Ideal Distributed Randomness Beacons . . . 32
3.2 Delay-Based Protocols . . . 35
3.2.1 Modifying Commit-Reveal . . . 36
3.2.2 Adding Recovery to Commit-Reveal . . . 37
3.2.3 Chain of VDFs . . . 38
3.3 Commit-Reveal-Punish . . . 39
3.3.1 Enforcing Every Reveal . . . 40
3.3.2 Rational Threshold Commit-Reveal . . . 40

3.4 Commit-Reveal-Recover . . . 44
3.4.1 From Threshold Secret Sharing . . . 44
3.4.1.1 Commit-Reveal-Recover . . . 45
3.4.1.2 Share-Reconstruct-Aggregate . . . 46
3.4.1.3 Share-Aggregate-Reconstruct . . . 46
3.4.2 From Threshold Encryption . . . 47
3.5 Committee-Based Protocols . . . 48
3.5.1 Step 1. Committee Selection . . . 48
3.5.1.1 Public Committee Selection . . . 48
3.5.1.2 Private Committee Selection . . . 50
3.5.2 Step 2. Beacon Output Generation . . . 52
3.5.2.1 Fresh Per-Node Entropy . . . 52
3.5.2.2 Combining Previous Output and Precommitted Per-Node Entropy 53
3.6 Protocols With No Marginal Entropy . . . 55
3.7 Discussion . . . 58
3.7.1 Relation to Collective Coin Flipping Protocols . . . 58
3.7.2 Withholding Attacks . . . 60
3.7.3 Adaptive Security . . . 61
3.7.4 Comparison of DRBs . . . 63
3.7.5 Concurrent Work . . . 64
3.8 Notes for Practitioners . . . 69

4 Bicorn: Tolerating Dishonest Majority with Optimistic Efficiency 71
4.1 Context . . . 71
4.2 Overview . . . 74
4.2.1 Protocol Outline . . . 74

4.2.2 Bicorn-ZK: Using Zero-Knowledge Proofs . . . 76
4.2.3 Bicorn-PC: Using Precommitment . . . 78
4.2.4 Bicorn-RX: Using Pseudorandom Exponents . . . 78
4.2.5 Comparison . . . 79
4.3 Timed DRBs: Syntax and Security Definitions . . . 80
4.4 Security of Bicorn-RX . . . 82
4.5 Security of Bicorn-ZK . . . 87
4.6 Security of Bicorn-PC . . . 89
4.7 Implementation . . . 92
4.8 Discussion . . . 94
4.9 Individual protocol presentations . . . 97

5 Cornucopia: Tolerating Dishonest Majority in Large-Scale Networks 101
5.1 Context . . . 101
5.2 Timed DRBs: Definitions and Constructions . . . 104
5.2.1 Unicorn . . . 106
5.2.2 Cornucopia . . . 107
5.3 Cornucopia Security . . . 107
5.4 Insertion-secure accumulators . . . 112
5.4.1 Accumulators without insertion security . . . 112
5.4.2 Merkle trees . . . 113
5.4.3 RSA accumulators . . . 114
5.4.4 Bilinear accumulators . . . 116
5.4.5 From generic universal accumulators . . . 117
5.4.6 From vector commitments . . . 120
5.5 Efficiency comparison of accumulator constructions . . . 122

5.6 Concluding Discussion 124

6 Conclusion **128**

Bibliography **130**

List of Figures

2.1 VDF sequentiality game 10

2.2 Accumulator security game 12

2.3 Security games for the repeated squaring hardness assumptions: computational RSW (left), computational power-of-RSW (center), and decisional RSW (right). . 15

2.4 Soundness (left) and zero-knowledge (right) security games for non-interactive zero-knowledge proof systems. 19

3.1 Security game for DRB unbiasability. 32

4.1 Security games for the main security properties: consistency (left), t-unpredictability (center), and t-indistinguishability (right). 81

5.1 Security games for (p, σ)-indistinguishability and (p, σ)-unpredictability. 105

5.2 The Unicorn timed DRB protocol [Lenstra and Wesolowski, 2015] 107

5.3 The Cornucopia protocol 108

5.4 Insertion security game 108

List of Tables

3.1 DRB Comparison 62

3.2 Committee-Based DRBs 65

4.1 A brief comparison of the Bicorn variants. See Protocol 1 for notation ($\langle\mathbb{G}\rangle$ and $\langle\mathcal{B}\rangle$ are the sizes of elements from $\mathbb{G}$ and $\mathcal{B}$, respectively) and Chapter 2 for a background on the RSW assumptions, the algebraic group model (AGM), the random oracle model (ROM), and zero-knowledge proof of knowledge of exponent (ZK-PoKE). 79

4.2 Ethereum gas costs and main operations involved for each Bicorn variant as well as Unicorn [Lenstra and Wesolowski, 2015] and Commit-Reveal DRBs. For Bicorn-PC, the Commit cost is split to show Precommit and Commit costs. The operations are: $\text{store}_{\mathbb{G}/2\lambda}$, storing a group element or 2λ-bit value; mul, multiplication of two group elements; exp, raising a group element to a power of size 2λ bits; poe.v and zk-poke.v, verifying a proof of exponentiation and proof of knowledge of exponent, respectively. Concrete costs are given with $\mathbb{G} = \mathbb{QR}_N^+$ within an RSA-2048 group and $\lambda = 128$. 93

5.1 Comparison of accumulator options for Cornucopia, at a security level of $\lambda = 128$ bits. Witness generation time is the time required to compute all n witnesses. †RSA accumulators can be instantiated using class groups [Long, 2018], which do not require trusted setup. We report numbers here for the classic RSA group $\mathbb{Z}_N^*$. . 122

1 Introduction

Secure randomness generation has long been a fundamental problem. With the advent of distributed ledger technologies and proof-of-stake protocols, the particular problem setting of generating a common random output from distributed sources in a network of mutually untrusting participants—where some may actively attempt to bias or predict the output—has gained significant attention in the past decade. The reason is that a random leader or committee must be selected periodically ad infinitum in many of these distributed systems for the purpose of coordinating the network state.

While this is reminiscent of the traditional literature on leader election in distributed systems, the key distinction lies in the network and trust assumptions. In traditional distributed systems, being a leader is not inherently beneficial or desirable. Participating nodes do not necessarily compete for leadership, which is why protocols like PBFT [Castro and Liskov, 1999] simply rotate leadership in a round-robin fashion whereas Raft [Ongaro and Ousterhout, 2014] bases leader election on honest local clocks of replicas.

The recent setting in a distributed ledger like Ethereum [Wood et al., 2014] is quite different. Here, nodes actively compete to become leaders, as each leader is rewarded for their coordination efforts which may lead to securing billions of dollars in an incentivized network. This unique context has spurred significant research in distributed randomness that is both publicly verifiable and robust in adversarial settings, with the goal of powering various leader election mechanisms and more.

1.1 Historical Context

Before delving into distributed randomness and distributed randomness beacons (DRBs), we make a couple historical notes and motivate the study by identifying prior gaps in the literature.

Coin Flipping. Although the problem of distributed randomness in adversarial settings is more relevant today than ever, it is certainly not a new challenge. Take coin flipping in information theory, for example. Foundational works by Ben-Or and Linial [Ben-Or and Linial, 1985, Ben-Or and Linial, 1989] examine coin flipping in the *full information* model, where computationally unbounded participants use a single broadcast channel to agree on a random bit. This body of literature primarily focuses on asymptotic (rather than concrete) bounds for corruption thresholds, bias, and round complexity. While these results are insightful from a theoretical perspective, a gap is that practical distributed randomness ideally needs to be as unbiasable as possible and must allow cryptography for practical purposes in the first place.

Now, let us consider coin flipping in classical cryptography. Blum's protocol for coin flipping over the phone [Blum, 1983], for instance, introduces cryptographic techniques but does not address adversarial participants who may abort or withhold values. Withholding is an issue that is fundamental in the sense that honest failures (e.g. due to network connectivity) and maliciously motivated withholding are indistinguishable. It is therefore a practical issue which Blum's work leaves as a gap; ideally, a well-designed distributed randomness protocol should handle this scenario gracefully. Some later works [Moran et al., 2009, Haitner and Tsfadia, 2014] do incorporate cryptography and account for withholding. However, they are constrained by Cleve's lower bound [Cleve, 1986]: for any r-round coin flipping protocol, an efficient adversary controlling half or more of the participants can bias the output by $\Omega(1/r)$. As a result, the best that these works can achieve is to meet Cleve's lower bound under classical assumptions. This again leaves a gap, as we want distributed randomness that has no bias. Interestingly, we show how to cir-

cumvent this lower bound with modern delay functions later in the dissertation, opening new possibilities for distributed randomness even with a dishonest majority.

Randomness beacon. The main classical work most closely related to modern research on distributed randomness is the concept of a *randomness beacon*, first formalized by Rabin [Rabin, 1983] in 1983. A randomness beacon is an ideal service that regularly emits fresh random values that no party can manipulate or predict. This simple yet powerful concept, if realized, could serve many amazing applications such as verifiable lotteries and gaming [Bonneau et al., 2015, Gainsbury and Blaszczynski, 2017], electronic voting [Adida, 2008], selecting parameters for cryptographic protocols [Baigneres et al., 2015, Lenstra and Wesolowski, 2015], leader election in proof-of-stake protocols [Gilad et al., 2017,Kiayias et al., 2017,Boneh et al., 2020,Edgington, 2023,Johnson et al., 2024,Oshitani and Drake, 2025], distributed ledger sharding [Al-Bassam et al., 2017, Kokoris-Kogias et al., 2018, Wang et al., 2019, David et al., 2022], timestamps [Chatzigiannis and Chalkias, 2021, Arun et al., 2022], asynchronous Byzantine consensus protocols [Abraham et al., 2022, Zhang and Duan, 2022, Duan et al., 2023], private stream aggregation [Brorsson and Gunnarsson, 2023], and federated learning in machine learning [Ma et al., 2023,van Kempen et al., 2023, Chen et al., 2024, Karthikeyan and Polychroniadou, 2024, Ma et al., 2024].

However, because no such ideal beacon exists, various protocols have been developed to approximate this functionality for practical use.

Centralized Beacon. Relying on a trusted third party like NIST [Fischer et al., 2011, Kelsey et al., 2019] or random.org [Haahr, 2010] might be the simplest way to realize a beacon. For example, NIST's beacon service publishes a 512-bit randomness output every 60 seconds. However, a centralized beacon carries drawbacks typically associated with centralized services, such as the risk of compromise or misbehavior due to a single point of failure as well as the inability of the end user to verify the security of the beacon. In particular, it is straightforward to design a malicious beacon that outputs statistically random values which are predictable given a trapdoor. For

instance, given a semantically secure encryption scheme, the underhanded beacon can simply use a secret key to encrypt a counter in each interval. The security of the underlying encryption scheme guarantees that this is indistinguishable from random without access to the key, but completely predictable given the key.

Implicit Beacon. Alternatively, implicit beacons exhibit randomness via publicly available entropy sources such as stock market data [Clark and Hengartner, 2010] or proof-of-work (PoW) blockchains like Bitcoin [Nakamoto, 2008,Bonneau et al., 2015,Bentov et al., 2016,Han et al., 2020]. These approaches leverage the apparent randomness in financial markets or mining processes to generate beacon outputs. Bitcoin's proof-of-work mechanism, for instance with its inherent unpredictability in solving cryptographic puzzles, can represent a source of entropy. Nonetheless, implicit beacons are potentially vulnerable to malicious insiders. Financial markets are susceptible to manipulation by actors making unnatural trades to fix prices or exchanges reporting incorrect data. Similarly, Bitcoin miners can withhold blocks or choose between colliding blocks, potentially biasing the randomness. While these beacons are plausibly secure and low-cost in practice, they still lack formal models of security and may not provide the guarantees required for meaningful applications.

Satellite-Based Beacon. Powered by Cryptosat's satellites launched since 2022 [Michalevsky, 2022], the Cosmic True Random Number Generator (cTRNG) project by SpaceComputer sources entropy from cosmic flares and is live in 2025 in beta form [Langellotti, 2025]. A recent development, such a beacon certainly offers unique security advantages by placing the randomness generation infrastructure in physical locations not readily accessible by any earthly actor. Powerful is the fact that this idea is now tangible as opposed to merely theoretical, as it instantiates Rabin's vision and also makes a meaningful step towards realizing a satellite-based blockchain altogether which is shown to have the potential to be highly performant at a very low energy cost [Shasha et al., 2023].

Distributed Randomness Beacon. While a satellite-based beacon represents an exciting direction, a different approach, arguably the most flexible and pragmatic, for contemporary usage is to reduce trust in a centralized or implicit beacon by performing a distributed protocol for the purpose of randomness, i.e. via multi-party distributed randomness beacon (DRB). The benefit of the DRB approach is its participatory nature (which for instance a satellite-based beacon does not provide), allowing flexible configuration of a beacon depending on context, network, and trust assumptions. At the same time, rich ideas from distributed computing, systems, and consensus related to fault tolerance can be leveraged to build a highly robust beacon. Indeed, DRB protocols are designed to remain secure and live despite some fraction of malicious participants, and various ways exist to often juggle with security and scalability overall.

1.2 Organization

The goal of the following chapters is to systematize distributed randomness beacons from a top-down perspective and then to present two novel protocols that can withstand a dishonest majority. We propose a general framework encompassing all DRB protocols in the landscape. To aid comparison and discussion of properties, we provide an overview of these protocols along with the cryptographic building blocks used to construct them. We identify two key components of DRB design: selection of entropy providers and beacon output generation, which can be decoupled from each other. Enabling a more holistic analysis of a DRB as a result, we also provide new insights and discussion on potential attack vectors, countermeasures, and techniques that lead to better scalability.

We begin with mathematical preliminaries in Chapter 2 and proceed to analyzing the design space of distributed randomness beacons in Chapter 3. We include our system model, a strawman DRB under perfect synchrony (an ideal assumption), *commit-reveal* [Blum, 1983], and the definition of an ideal DRB in Section 3.1. Section 3.2 introduces protocols using *delay functions* (verifi-

able delay functions [Boneh et al., 2018a] and timed commitments [Boneh and Naor, 2000]), which offer the best fault tolerance (dishonest majority) and simplicity, assuming secure delay functions can be implemented in practice. In Section 3.3 to 3.6, we introduce non-delay-based DRB protocols categorized by the number of nodes contributing *marginal entropy* (i.e. per-epoch randomness that is independently generated at a node level) in each epoch. Sections 3.3 and 3.4 review protocols in which all nodes contribute marginal entropy. These protocols vary in mechanisms used to recover from faulty nodes, including financial punishment [RANDAO, 2016, Yakira et al., 2020], threshold secret sharing [Schoenmakers, 1999, Cascudo and David, 2017], and threshold encryption [Desmedt and Frankel, 1990]. Section 3.5 covers committee-based protocols in which each epoch includes an extra committee selection step, after which only a committee (subset) of nodes contributes marginal entropy. These protocols are more complex but can offer greater communication efficiency with large numbers of nodes. Section 3.6 covers pseudorandom protocols that do not require any marginal entropy; these protocols can be highly efficient but have no mechanism to recover from compromise.

We highlight the importance of a dishonest majority setting when elaborating on Bicorn (Chapter 4) and Cornucopia (Chapter 5), in which we via Bicorn achieve the best of commit-reveal and delay functions, and via Cornucopia achieve sublinear verification cost per beacon output.

1.3 Statement of Work

This dissertation is entirely my own work except where noted based on prior publications:

SoK: Distributed Randomness Beacons. The authors are Kevin Choi, Aathira Manoj, and Joseph Bonneau, and the work was published in *IEEE Symposium on Security and Privacy* in 2023. This work is expanded in Chapter 3.

- A holistic literature review, coming up with the framework to encompass all DRBs, and updating the DRB Comparison Table over time

Bicorn: An optimistically efficient distributed randomness beacon. The authors are Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau, and the work was published in *Financial Cryptography and Data Security* in 2023. This work corresponds to Chapter 4.

- Conceiving the RX (randomized exponent) variant of Bicorn, working on security proofs (unpredictability and indistinguishability) for the RX variant in the algebraic group model, and collaborating on security proofs for the other variants

Cornucopia: Distributed Randomness Beacons at Scale. The authors are Miranda Christ, Kevin Choi, and Joseph Bonneau, and the work was published in *Advances in Financial Technologies* in 2024. This work corresponds to Chapter 5.

- DRB definition and security properties, and narrating the protocol in light of related work (such as Unicorn and Bicorn)

I also published the following works which are not included in this dissertation:

Thistle: A vector database in Rust. The authors are Brad Windsor and Kevin Choi, and the work was published as *arXiv preprint* (arXiv:2303.16780) in 2023.

- Implementing the HNSW algorithm (graph-based approximate nearest neighbor search algorithm called hierarchical navigable small world) in Rust, pipelining deep learning models, and making vector queries in light of Transformer-based BERT model and embeddings

Accountable Secret Leader Election. The authors are Miranda Christ, Kevin Choi, Walter McKelvie, Joseph Bonneau, and Tal Malkin, and the work was published in *Advances in Financial Technologies* in 2024.

- Conceiving the idea, refining definitions of accountability, and working on the committee-based approach which uses threshold encryption

2 Preliminaries

In this chapter, we define common or recurring building blocks, assumptions, and notations. We first define verifiable delay functions (VDFs) [Boneh et al., 2018a], accumulators [Benaloh and De Mare, 1993], and vector commitments [Catalano and Fiore, 2013]. They 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 $\mathrm{poly}(\lambda)$ and $\mathrm{negl}(\lambda)$ 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. We let $[k]$ denote the set $\{1, \ldots, k\}$.

2.1 Verifiable delay functions

Definition 2.1 (Verifiable delay function [Boneh et al., 2018a])**.** A *verifiable delay function* (VDF) 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

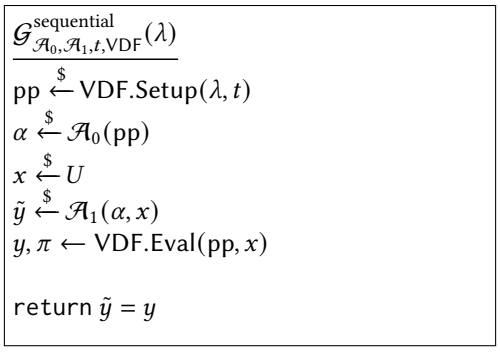

Figure 2.1: VDF sequentiality game

function should run in t sequential steps.

$\text{VDF.Verify}(\text{pp}, x, y, \pi) \to \{0, 1\}$ takes an input x, output y, and optional proof π, and verifies if (y, π) is a genuine output of Eval.

VDFs must satisfy the following three properties:

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

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

Sequentiality. VDFs must impose a computational delay. Roughly speaking, computing a VDF successfully with non-negligible probability over a uniformly distributed challenge x should be impossible without executing t sequential steps. Formally (adapted from [Boneh et al., 2018a]):

Definition 2.2 (VDF sequentiality [Boneh et al., 2018a])**.** A VDF is (p, σ)-*sequential* if for all randomized algorithms $\mathcal{A}_0$ which run in total time $O(\text{poly}(t, \lambda))$, and $\mathcal{A}_1$ which run in parallel

time $\sigma(t)$ on at most $p(t)$ processors:

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

where $\mathcal{G}^{\text{sequential}}_{\mathcal{A}_0,\mathcal{A}_1,t,\mathsf{VDF}}(\lambda)$ is defined in Figure 2.1.

2.2 Accumulators

Definition 2.3 (Accumulator [Benaloh and De Mare, 1993, Camenisch and Lysyanskaya, 2002])**.** Given a data universe U, an *accumulator* [Benaloh and De Mare, 1993] is a tuple of algorithms (Setup, Accumulate, GetMemWit, 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.

$\mathsf{Acc.MemVer}(A, x, w) \to \{0, 1\}$ takes as input an accumulator value A, an element x, and a membership proof (membership witness) w. It verifies if x is included in the accumulated set represented by A.

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 [Camenisch and Lysyanskaya, 2002]. 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.

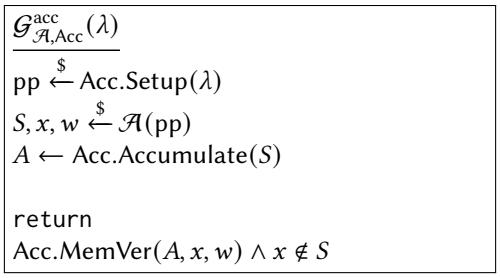

Figure 2.2: Accumulator security game

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 2.4 (Accumulator security [Camenisch and Lysyanskaya, 2002])**.** An accumulator Acc is *secure* if no PPT adversary $\mathcal{A}$ can succeed with non-negligible probability in $\mathcal{G}^{\text{acc}}_{\mathcal{A},\text{Acc}}(\lambda)$ as defined in Figure 2.2.

A *universal accumulator* [Li et al., 2007] also supports non-membership proofs; that is, it supports two additional functions:

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

$\text{Acc.NonMemVer}(A, x', w') \to \{0, 1\}$ takes as input an accumulator value A, an element x', and a non-membership proof (non-membership witness) w'. It verifies if x' is *not* included in the accumulated set represented by A.

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 5.4.5 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 2.5 (Universal accumulator security [Li et al., 2007])**.** A universal accumulator Acc is *secure* if for all PPT adversaries $\mathcal{A}$:

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

2.3 Vector commitments

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

Definition 2.6 (Vector commitment [Catalano and Fiore, 2013])**.** Given a message space $\mathcal{M}$, a *vector commitment* is a tuple of algorithms including:

$\mathsf{Setup}(\lambda, s) \to \mathsf{pp}$ takes in the security parameter λ and the size s of the committed vector, and outputs public parameters pp.

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

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

$\mathsf{Ver}(C, m, i, \pi_i) \to \{0, 1\}$ takes as input a commitment, a message m, an index i, and a proof that the i^{th} component of the committed vector is m. It verifies the proof.

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 2.7 (Position binding [Catalano and Fiore, 2013])**.** A vector commitment satisfies *position binding* if for all $i \in [s]$ and for all PPT adversaries $\mathcal{A}$:

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

2.4 Algebraic group model

In some of our security proofs, we consider security against *algebraic* adversaries which we model using the algebraic group model (AGM), following the treatment of [Fuchsbauer et al., 2018]. We call an algorithm $\mathcal{A}$ *algebraic* if for all group elements Z that are output (either as final output or as input to oracles), $\mathcal{A}$ additionally provides the representation of Z relative to all previously received group elements. The previously received group elements include both original inputs to the algorithm and outputs received from calls to oracles. More specifically, if $[X]_i$ is the list of group elements $[X_0, \ldots, X_n] \in \mathbb{G}$ that $\mathcal{A}$ has received so far, then, when producing group element Z, $\mathcal{A}$ must also provide a list $[z]_i = [z_0, \ldots, z_n]$ such that $Z = \prod_i X_i^{z_i}$.

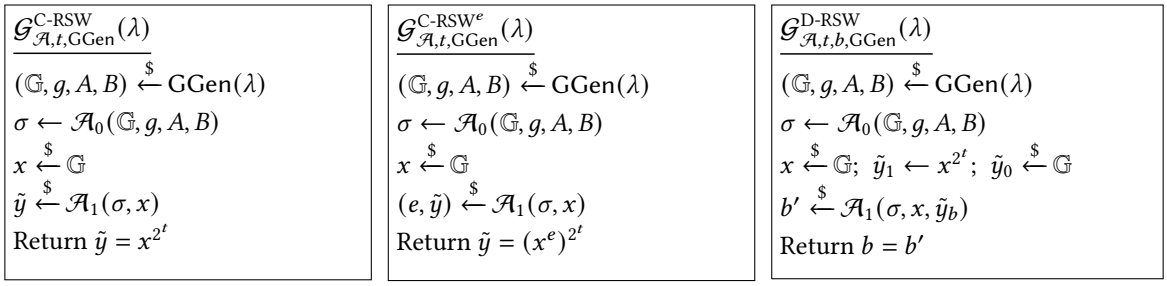

Figure 2.3: Security games for the repeated squaring hardness assumptions: computational RSW (left), computational power-of-RSW (center), and decisional RSW (right).

2.5 Groups of unknown order and RSW assumptions

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

There are a few currently known options with which to instantiate a group of unknown order. One option that requires only a transparent setup is through class groups of imaginary quadratic order [Buchmann and Hamdy, 2011]. However, class groups typically incur high concrete overheads. Instead, one may opt for more efficient RSA groups, which require a trusted setup or multiparty computation "ceremony" [Chen et al., 2021] to compute the modulus $N = pq$ without revealing safe primes p, q. Looking forward, we will require our group to additionally be cyclic and satisfy the low order assumption [Boneh et al., 2018b]. So instead we will use the group $\mathbb{QR}^+_N$, the group of signed quadratic residues modulo N (we refer to Pietrzak for more details [Pietrzak, 2018]).

The security of our constructions is based on the assumption, proposed by RSW [Rivest et al., 1996], that, given a random element $x \in \mathbb{G}$, the fastest algorithm to compute $y = x^{(2^t)}$ takes t sequential steps. We follow the formalism of Katz et al. [Katz et al., 2020]. We use three RSW

assumptions; we provide security games in Figure 2.3.

Computational RSW. In the computational RSW game, the adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1)$ is tasked with computing the 2^t-th power of a challenge element. The adversary acts in two stages. In the preprocessing stage, $\mathcal{A}_0$ is given a description of the group and outputs an intermediate state passed to $\mathcal{A}_1$. In the challenge stage, $\mathcal{A}_1$ is given the challenge input x and intermediate state σ and attempts to output $y = x^{2^t}$. We define the advantage of an adversary as:

$$\mathsf{Adv}^{\text{C-RSW}}_{\mathcal{A},t,\text{GGen}}(\lambda) = \Pr\left[\mathcal{G}^{\text{C-RSW}}_{\mathcal{A},t,\text{GGen}}(\lambda) = 1\right]$$

In this game and in the following, the advantage is only meaningful when the challenge stage adversary's running time is limited to $< t$. In the Strong AGM (SAGM) [Katz et al., 2020] and when $\mathbb{G}$ is $\mathbb{QR}_N$, it is shown that when $\mathcal{A}_0, \mathcal{A}_1$ run in fewer than $poly(\lambda)$, t steps, respectively, $\mathcal{G}^{\text{C-RSW}}$ can be won with only negligible probability assuming the hardness of factoring [Katz et al., 2020, Theorem 2].

Computational power-of-RSW. We introduce a stronger variant of computational RSW that we term computational "power-of-RSW." In this game, the adversary need not output $y = x^{2^t}$ directly, rather the adversary may output (e, y_e) such that $y_e = (x^e)^{2^t}$. The hardness of computational power-of-RSW for time-bounded adversaries can be shown with a slight modification of the proof used to show the hardness of computational RSW in [Katz et al., 2020]. The SAGM adversary outputs d (alongside $e \neq 0$) with $|d| < 2^t$ such that $x^d = x^{e \cdot 2^t}$. Computing $4(e \cdot 2^t - d)$ then gives a multiple of $\phi(N)$, allowing us to factor N. We define the advantage of an adversary as:

$$\mathsf{Adv}^{\text{C-RSW}^e}_{\mathcal{A},t,\text{GGen}}(\lambda) = \Pr\left[\mathcal{G}^{\text{C-RSW}^e}_{\mathcal{A},t,\text{GGen}}(\lambda) = 1\right]$$

Decisional RSW. Finally, a stronger decisional assumption is that an attacker cannot even distinguish x^{2^t} from a random group element. There is no proof for this assumption, even in generic models. We define the advantage of an adversary as:

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

2.6 Randomizing exponent sizes

We recall a useful lemma for randomizing group elements [Micciancio, 2005].

Lemma 2.8. *For any cyclic group $\mathbb{G}$ and generator g, if $r \xleftarrow{\$} \mathcal{B}$ is chosen uniformly at random, then the statistical distance between g^r and the uniform distribution over $\mathbb{G}$ is at most $\frac{|\mathbb{G}|}{2|\mathcal{B}|}$.*

Looking forward, we will use this lemma in our security proofs to replace a generator taken to the power of a large exponent of size $|\mathcal{B}| \approx 2^{2\lambda} \cdot |\mathbb{G}|$ with a random element. Alternatively, one may opt for the stronger *short exponent indistinguishability (SEI) assumption* [Couteau et al., 2021] which asserts that an adversary cannot computationally distinguish between a uniformly random element of $\mathbb{G}$ and g^r for $r \xleftarrow{\$} [0, 2^{2\lambda}]$. The latter assumption enables significant efficiency gains in practice, with participants publishing 32-byte α values instead of 288 bytes.

2.7 Non-interactive zero-knowledge proofs

A *non-interactive proof system* for a relation $\mathcal{R}$ over *statement-witness* pairs (x, w) enables producing a proof, $\pi \leftarrow \mathsf{Prove}(\mathsf{pk}, x, w)$, that convinces a verifier $\exists w : (x, w) \in \mathcal{R}$, $0/1 \leftarrow \mathsf{Verify}(vk, \pi, x)$; pk and vk are proving and verification keys, $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{Keygen}(\mathcal{R})$. A *non-interactive argument of knowledge* further convinces the verifier not only that the witness w exists but also that the prover *knows* w, and if proved in *zero-knowledge*, the verifier does not learn any additional information about w. The formal security properties of a non-interactive proof system are as follows.

Completeness. A proof system is *complete* if given a true statement, a prover with a witness can convince the verifier. We will make use of proof systems with perfect completeness. A proof system has *perfect completeness* if for all $(x, w) \in \mathcal{R}$,

$$\Pr\left[\mathsf{Verify}(x, \mathsf{Prove}(x, w)) = 1\right] = 1 \ .$$

Knowledge soundness. A proof system is computationally *knowledge sound* if whenever a prover is able to produce a valid proof for a statement x, it is a true statement, i.e., there exists some witness w such that $(x, w) \in \mathcal{R}$. We require a stronger property to allow for simulating proofs for false statements. This scenario is common in security proofs and so it is desirable to have soundness even in the presence of simulated proofs. This stronger notion of knowledge soundness is known as *simulation-extractability* and is defined by the security game $\mathcal{G}^{\text{sound}}_{\mathcal{A},\mathcal{R},\mathsf{PoK},\mathsf{S},\mathsf{Ext}}(\lambda)$ (Figure 2.4) in which an adversary is tasked with finding a verifying statement and proof where the statement is not in $\mathcal{R}$. The advantage of an adversary is defined as

$$\mathsf{Adv}^{\text{sound}}_{\mathcal{A},\mathcal{R},\mathsf{PoK},\mathsf{S},\mathsf{Ext}}(\lambda) = \Pr[\mathcal{G}^{\text{sound}}_{\mathcal{A},\mathcal{R},\mathsf{PoK},\mathsf{S},\mathsf{Ext}}(\lambda) = 1] \ .$$

Zero-knowledge. A proof system is computationally *zero-knowledge* if a proof does not leak any information besides the truth of a statement. Zero-knowledge is defined by the security game $\mathcal{G}^{\text{zk}}_{\mathcal{A},b,\mathcal{R},\mathsf{PoK},\mathsf{S}}(\lambda)$ (Figure 2.4) in which an adversary is tasked with distinguishing between proofs generated from a valid witness and simulated proofs generated without a witness by simulator S. The advantage of an adversary is defined as

$$\mathsf{Adv}^{\text{zk}}_{\mathcal{A},\mathcal{R},\mathsf{PoK},\mathsf{S}}(\lambda) = \left|\Pr[\mathcal{G}^{\text{zk}}_{\mathcal{A},1,\mathcal{R},\mathsf{PoK},\mathsf{S}}(\lambda) = 1] - \Pr[\mathcal{G}^{\text{zk}}_{\mathcal{A},0,\mathcal{R},\mathsf{PoK},\mathsf{S}}(\lambda) = 1]\right| \ .$$

In this work, we will make use of proof systems for two relations. First, we use PoE for the following relation for proofs of exponentiation in groups of unknown order [Wesolowski, 2019, Pietrzak, 2018, Boneh et al., 2019]: $\{((x, y \in \mathbb{G}, \alpha \in \mathbb{Z}), \perp) : y = x^{\alpha}\}$. Second, we use ZKPoKE (realized by ZKPoKRep from [Boneh et al., 2019]) for zero-knowledge proofs of knowledge of exponent in groups of unknown order: $\{((x, y \in \mathbb{G}), \alpha \in \mathbb{Z}) : y = x^{\alpha}\}$.

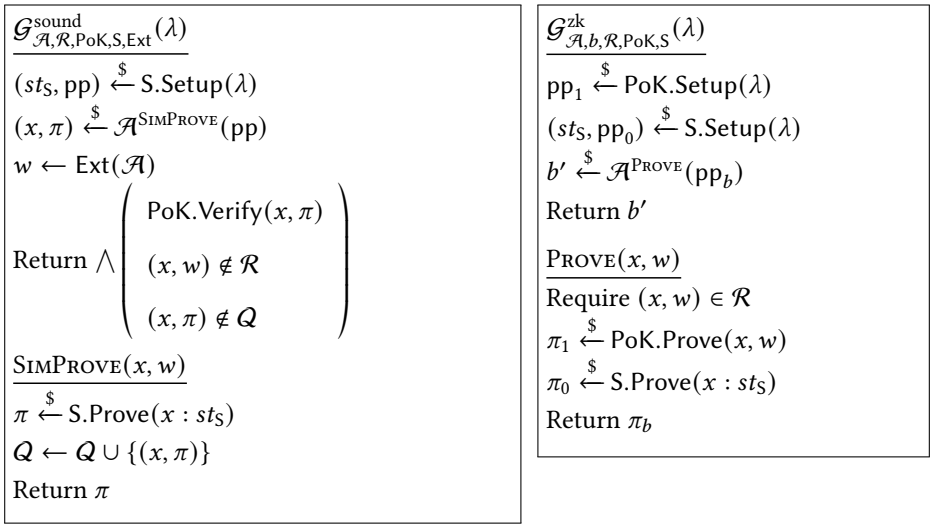

Figure 2.4: Soundness (left) and zero-knowledge (right) security games for non-interactive zero-knowledge proof systems.

2.8 Verifiable Secret Sharing (VSS)

VSS schemes have two security requirements.

- Secrecy. If the dealer is honest, then the probability of an adversary learning any information about the dealer's secret in the sharing phase is $\text{negl}(\lambda)$.
- Correctness. If the dealer is honest, then the honest nodes output the secret s at the end of the reconstruction phase with a high probability of $1 - \text{negl}(\lambda)$.

Feldman-VSS [Feldman, 1987] and Pedersen-VSS [Pedersen, 1991a] are the most commonly used VSS schemes.

Feldman-VSS. The following summarizes a simple VSS scheme proposed by Paul Feldman for sharing a secret s among n participants where any subset of $t + 1$[1] among them can reconstruct the secret.

[1] t related to secret sharing denotes the maximum number of nodes an adversary can corrupt.

- ShareGen$(s) \rightarrow (\{s_i\}, C)$ with $s \in \mathbb{Z}_q$ involves the dealer sampling t random coefficients $a_1, \ldots, a_t \in \mathbb{Z}_q$ and constructing $p(x) = s + a_1 x + a_2 x^2 + \cdots + a_t x^t$. The shares are computed as $s_i = p(i)$ in mod q for $1 \leq i \leq n$ and shared privately with each participant. The commitments to the secret $C_0 = g^s$ as well as coefficients $C_j = g^{a_j}$ for $j = 1, \ldots, t$ are also broadcast by the dealer.

- ShareVerify$(s_i, C) \rightarrow \{0, 1\}$ involves each participant P_i checking if:

$$g^{s_i} = \prod_{j=0}^{t} C_j^{i^j} = C_0 C_1^i C_2^{i^2} \cdots C_t^{i^t}$$

  If it does not hold for some i, then P_i broadcasts an accusation against the dealer, who has to respond by broadcasting the correct s_i. Correct reconstruction is achieved by filtering out shares not passing ShareVerify.

- Recon$(A, \{s_i\}_{i \in A}) \rightarrow s$ outputs the secret s by performing Lagrange interpolation (see Section 2.16.1) with $t + 1$ valid shares from the reconstruction set A of nodes:

$$s = p(0) = \sum_{j \in A} p(j) \lambda_{0,j,A}$$

The verifiability in Feldman-VSS comes from inclusion of commitments to the coefficients. These commitments enable participants to verify the validity of the shares that they receive from the dealer.

2.9 Distributed Key Generation (DKG)

One of the best known DKG schemes is Joint-Feldman, proposed by Pedersen [Pedersen, 1991b].

Joint-Feldman. In this DKG scheme, each participant uses Feldman-VSS to share a randomly chosen secret. The protocol is implemented as follows:

- $\text{DKG}(1^\lambda, t, n) \to (sk_i, pk_i, pk)$ proceeds in two phases—Sharing and Reconstruction.

    1. In Sharing phase, each participant P_i runs Feldman-VSS by choosing a random polynomial over $\mathbb{Z}_q$ of degree t, $p_i(z) = \sum_{j=0}^{t} a_{ij} z^j$, and sending a subshare $s_{ij} = p_i(j)$ in mod q to each participant P_j privately. To satisfy the verifiability portion of VSS, P_i also broadcasts $C_{ik} = g^{a_{ik}}$ for $k = 0, \ldots, t$. Let the commitment corresponding to the secret be denoted by $y_i = C_{i0}$. Each participant P_j also verifies the subshares it receives from other participants by performing verification steps of Feldman-VSS on each subshare. If verification for index i fails, P_j broadcasts a complaint against P_i. If P_i receives more than t complaints, then P_i is disqualified. Otherwise, P_i reveals the subshare s_{ij} for every P_j that has broadcast a complaint. We call C the set of non-disqualified participants.
    2. Reconstruction phase calculates the keys based on C. The group public key is calculated as $pk = \prod_{i \in C} y_i$ where the individual public keys are $pk_i = y_i$. Each participant P_j's individual secret key is computed as $sk_j = \sum_{i \in C} s_{ij}$. Though not computed explicitly, the group secret key sk is equal to both $\sum_{i \in C} a_{i0}$ and the Lagrange interpolation involving the individual secret keys $\{sk_j\}_{j \in C}$.

2.10 Publicly Verifiable Secret Sharing (PVSS)

PVSS can be described by the following algorithms:

- $\text{Setup}(\lambda) \to pp$ generates the public parameters pp, an implicit input to all other algorithms.

- $\mathsf{KeyGen}(\lambda) \rightarrow (sk_i, pk_i)$ generates the PVSS key pair used for encryption and decryption for node i.
- $\mathsf{Enc}(pk_i, m) \rightarrow c$ and $\mathsf{Dec}(sk_i, c) \rightarrow m'$ are subalgorithms used to encrypt and decrypt the share to node i, respectively. Both Enc and Dec may optionally output a proof (e.g. π_{DLEQ}).
- $\mathsf{ShareGen}(s) \rightarrow (\{\mathsf{Enc}(pk_i, s_i)\}, \{s_i'\}, \pi)$ with $s_i' = \mathsf{Dec}(sk_i, \mathsf{Enc}(pk_i, s_i))$ is a two-part process. First, the dealer with secret s generates secret shares $\{s_i\}$ and sends each encrypted share $\mathsf{Enc}(pk_i, s_i)$ to node i with an optional encryption proof π_{Enc_i}. Second, node i decrypts the received encrypted share to generate s_i' and broadcasts it with an optional decryption proof π_{Dec_i}. Note that it is possible that $s_i' \neq s_i$. In fact, $s_i' = h^{s_i}$ is standard (referred to as sharing a group element). π incorporates $\{\pi_{Enc_i}\}$ and $\{\pi_{Dec_i}\}$ as well as any auxiliary proof necessary.
- $\mathsf{ShareVerify}(\{\mathsf{Enc}(pk_i, s_i)\}, \{s_i'\}, \pi) \rightarrow \{0, 1\}$ verifies if ShareGen is correct overall using π.
- $\mathsf{Recon}(A, \{s_i'\}_{i \in A}) \rightarrow s'$ reconstructs the shared secret s' via Lagrange interpolation (in the exponent) from a set A of $t + 1$ nodes whose contributions are passed by the ShareVerify algorithm. Typically, $s' = h^s$ in the landscape (referred to as sharing a group element).

PVSS is a secure VSS scheme providing the following additional guarantee:

- Public Verifiability. If the ShareVerify algorithm returns 1, then the scheme is valid in a publicly verifiable manner with high probability $1 - \mathsf{negl}(\lambda)$.

Schoenmakers PVSS. One of the simplest PVSS schemes used in practice is one by Schoenmakers [Schoenmakers, 1999]. As typical, the setup involves $g, h \in \mathbb{G}_q$. Additionally, each participant P_i generates a secret key $x_i \in \mathbb{Z}_q^*$ and registers $y_i = h^{x_i}$ as its public key.

- $\mathsf{ShareGen}(s) \rightarrow (\{\mathsf{Enc}(y_i, s_i)\}, \{s'_i\}, \pi)$ with s'_i equal to $\mathsf{Dec}(x_i, \mathsf{Enc}(y_i, s_i))$ first involves production of $\{\mathsf{Enc}(y_i, s_i)\}$ by the dealer with secret s. Namely, the dealer picks a random polynomial p of degree t with coefficients in $\mathbb{Z}_q$

$$p(x) = \sum_{i=0}^{t} a_i x^i$$

where $s = p(0) = a_0$ and computes $Y_i = \mathsf{Enc}(y_i, s_i) = y_i^{p(i)}$, which is sent to each node i along with information needed to prove its correctness: $C_j = g^{a_j}$ for $0 \leq j \leq t$ such that $X_i = \prod_{j=0}^{t} C_j^{i^j} = g^{p(i)}$ and $\mathsf{DLEQ}(g, X_i, y_i, Y_i)$ (see Section 2.16.3). Upon receiving Y_i, node i computes $s'_i = \mathsf{Dec}(x_i, Y_i) = Y_i^{1/x_i} = h^{p(i)}$ and generates information needed to prove its correctness: $\mathsf{DLEQ}(h, y_i, s'_i, Y_i)$.

- $\mathsf{ShareVerify}(\{Y_i\}, \{s'_i\}, \pi) \rightarrow \{0, 1\}$ verifies the encryption proof $\mathsf{DLEQ}(g, X_i, y_i, Y_i)$ where X_i's are computed from C_j's as well as the decryption proof $\mathsf{DLEQ}(h, y_i, s'_i, Y_i)$.

- $\mathsf{Recon}(A, \{s'_i\}_{i \in A}) \rightarrow h^s$ performs the following Lagrange interpolation in the exponent

$$\prod_{i \in A} (s'_i)^{\lambda_{0,i,A}} = h^{\sum_{i \in A} p(i) \lambda_{0,i,A}} = h^{p(0)} = h^s$$

where $\lambda_{0,i,A}$ denotes the Lagrange coefficients. Note that, unlike VSS, the scheme does not require the knowledge of the values $p(i)$ by the participants. The secret keys x_i are not exposed as well and thus can be reused.

2.11 Verifiable Random Function (VRF)

A VRF [Micali et al., 1999, Dodis and Yampolskiy, 2005] is a function that, given an input x and a secret key sk, generates a unique, pseudorandom output y as well as a proof π verifying that the computation has been done correctly. Due to π, it is possible to repeatedly generate new

pseudorandom outputs with one sk and varying inputs in a verifiable manner whereas otherwise (e.g. using a classical pseudorandom function) one needs to divulge the secret key and sacrifice its reusability for public verification purposes. It can be represented by the following tuple of algorithms.

- $\mathsf{Prove}(sk, x) \rightarrow (F_{sk}(x), \pi_{sk}(x))$ generates the pseudorandom output $F_{sk}(x)$ and its proof of correctness $\pi_{sk}(x)$ given input x and secret key sk.
- $\mathsf{Verify}(pk, x, y, \pi) \rightarrow \{0, 1\}$ outputs 1 if it is verified that $y = F_{sk}(x)$ using the proof π and 0 otherwise.

2.12 Distributed Verifiable Random Function (DVRF)

A *distributed verifiable random function* (DVRF) is a VRF where any $t + 1$ out of n nodes can jointly compute a pseudorandom output while any t nodes cannot. It can be described by the following algorithms:

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ runs a typical DKG.
- $\mathsf{PartialEval}(sk_i, x) \rightarrow (y_i, \pi_i)$ outputs the partial evaluation y_i as well as its proof of correctness π_i given an input x and a node's secret key sk_i.
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i) \rightarrow \{0, 1\}$ verifies the correctness of the partial evaluation y_i given its proof π_i, an input x, and a node's public key pk_i.
- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i \in A}) \rightarrow (y, \pi)$ outputs the DVRF evaluation y as well as its proof of correctness π given a set A of $t + 1$ nodes and their outputs of $\mathsf{PartialEval}(sk_i, x)$, all of which pass PartialVerify.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi) \rightarrow \{0, 1\}$ verifies the DVRF evaluation y given π, input x, and public keys.

2.13 DDH-DVRF

DDH-DVRF (from the decisional Diffie-Hellman assumption) is described by the following DVRF algorithms.

- $\mathsf{DKG}(1^\lambda, t, n)$ runs a typical DKG.
- $\mathsf{PartialEval}(sk_i, x)$ outputs (y_i, π_i) where $y_i = H(x)^{sk_i}$ and $\pi_i = \mathsf{DLEQ}(g, g^{sk_i}, H(x), H(x)^{sk_i})$ denoting the non-interactive Chaum-Pedersen protocol (see Section 2.16.3).
- $\mathsf{PartialVerify}(pk_i, x, y_i, \pi_i)$ is equivalent to $\mathsf{DLEQ\text{-}Verify}(g, pk_i, H(x), y_i, \pi_i)$ (Section 2.16.3) and verifies the correctness of the PartialEval algorithm using π_i.
- $\mathsf{Combine}(A, \{(y_i, \pi_i)\}_{i\in A})$ outputs (y, π) where $y = \prod_{i\in A} y_i^{\lambda_{0,i,A}}$ and $\pi = \{(y_i, \pi_i)\}_{i\in A}$. Details related to Lagrange coefficients $\lambda_{0,i,A}$ are included in Section 2.16.1.
- $\mathsf{Verify}(pk, \{pk_i\}, x, y, \pi)$ verifies all partial proofs via PartialVerify for all $i \in A$ from π and checks $y = \prod_{i\in A} y_i^{\lambda_{0,i,A}}$.

2.14 GLOW-DVRF

Providing a compact proof π, GLOW-DVRF uses a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ similar to BLS (Section 2.16.2) such that the setup includes hash functions $H_1 : \{0,1\}^* \to \mathbb{G}_1$ and $H_2 : \mathbb{G}_1 \to \{0,1\}^{y(\lambda)}$. While resembling DDH-DVRF, the following algebraic modifications are made due to pairings.

- $\mathsf{DKG}(1^\lambda, t, n)$ is adapted so that pk_i resides in $\mathbb{G}_1$ while pk resides in $\mathbb{G}_2$. This is achieved by letting $(pk_i, pk) = (g_1^{sk_i}, g_2^{sk})$ for $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$. The purpose of this is to facilitate a compact proof in the final Verify step.
- $\mathsf{PartialEval}(sk_i, x)$ outputs (y_i, π_i); $y_i = H_1(x)^{sk_i}$ and $\pi_i = \mathsf{DLEQ}(g_1, g_1^{sk_i}, H_1(x), H_1(x)^{sk_i})$.

- PartialVerify(pk_i, x, y_i, π_i) is equivalent to DLEQ-Verify$(g_1, pk_i, H_1(x), y_i, \pi_i)$ and verifies the correctness of the PartialEval algorithm using π_i.
- Combine$(A, \{(y_i, \pi_i)\}_{i \in A})$ outputs (y, π) where $\pi = \prod_{i \in A} y_i^{\lambda_{0,i,A}}$ and $y = H_2(\pi)$. Note that π is a group element.
- Verify$(pk, \{pk_i\}, x, y, \pi)$ verifies $y = H_2(\pi)$ and a pairing equation $e(\pi, g_2) = e(H_1(x), pk)$.

2.15 Dfinity-DVRF

Dfinity-DVRF is given by the following DVRF algorithms.

- DKG$(1^\lambda, t, n)$ is adapted so that both pk_i and pk reside in $\mathbb{G}_2$. This is achieved by letting $(pk_i, pk) = (g_2^{sk_i}, g_2^{sk})$ for $g_2 \in \mathbb{G}_2$. The purpose of this is to facilitate the check of some pairing equation in both PartialVerify and Verify.
- PartialEval(sk_i, x) outputs (y_i, π_i) where $y_i = H_1(x)^{sk_i}$ and $\pi_i = \perp$. The reason for a null proof is that a pairing equation check is used in PartialVerify (i.e. the differentiator from GLOW-DVRF) with no need for any auxiliary information.
- PartialVerify(pk_i, x, y_i, π_i) checks a pairing equation $e(y_i, g_2) = e(H_1(x), pk_i)$.
- Combine$(A, \{(y_i, \pi_i)\}_{i \in A})$ equals that in GLOW-DVRF.
- Verify$(pk, \{pk_i\}, x, y, \pi)$ equals that in GLOW-DVRF.

2.16 Other Cryptographic Primitives

2.16.1 Lagrange Interpolation

Given a non-empty reconstruction set $A \subset \mathbb{Z}_q$, the *Lagrange basis polynomials* are given by $\lambda_{j,A}(x) = \prod_{k \in A \setminus \{j\}} \frac{x-k}{j-k} \in \mathbb{Z}_q[X]$ such that the *Lagrange coefficients* $\lambda_{i,j,A} = \lambda_{j,A}(i) \in \mathbb{Z}_q$ enable

the equality $p(i) = \sum_{j \in A} p(j)\lambda_{i,j,A}$ for any polynomial $p \in \mathbb{Z}_q[X]$ of degree at most $|A| - 1$. The process of computing this equality is called *Lagrange interpolation*.

2.16.2 BLS Signature

Introduced by Boneh, Lynn, and Shacham in 2003, the BLS signature scheme [Boneh et al., 2001] consists of the following tuple of algorithms given a key pair (sk, pk).

- $\mathsf{Sign}_{sk}(m) \rightarrow H_1(m)^{sk}$ outputs a digital signature $\sigma = H_1(m)^{sk}$ given secret key sk and message m where H_1 is a hash function such that $H_1 : \{0,1\}^* \rightarrow \mathbb{G}_1$.
- $\mathsf{Verify}_{pk}(m, \sigma) \rightarrow \{0,1\}$ verifies σ given signature σ, message m, and public key pk via $e(\sigma, g_2) = e(H_1(m), pk)$.

Note that BLS uses a bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ with $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, $\mathbb{G}_T$ denoting a cyclic group of prime order q, and the following requirements.

- Bilinearity. $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$ for all $x, y \in \mathbb{Z}_q^*$.
- Non-degeneracy. $e(g_1, g_2) \neq 1$.
- Computability. $e(g_1, g_2)$ can be efficiently computed.

The threshold variant [Boldyreva, 2003] of BLS (i.e. threshold BLS) requires $\mathsf{Sign}_{sk}(m)$ to be computed by $t + 1$ out of n nodes. This is achieved via DKG such that sk denotes the implicit group secret key whereas each node broadcasts its partial signature $H_1(m)^{sk_i}$, $t+1$ of which from the set A of honest nodes are combined to generate

$$H_1(m)^{sk} = \prod_{i \in A} \left(H_1(m)^{sk_i} \right)^{\lambda_{0,i,A}}$$

via Lagrange interpolation in the exponent.

2.16.3 NIZK of Discrete Logarithm Equality (DLEQ)

Also known as the Chaum-Pedersen protocol [Chaum and Pedersen, 1992], the Σ protocol [Damgård, 2002] for proving that the two discrete logarithms are equal without revealing the discrete logarithm value itself can be turned into a NIZK by applying the Fiat-Shamir heuristic [Fiat and Shamir, 1986]. Namely, the prover can non-interactively prove the knowledge of α such that $(h_1, h_2) = (g_1^\alpha, g_2^\alpha)$ via $\pi_{DLEQ} = \mathsf{DLEQ}(g_1, h_1, g_2, h_2)$ with group elements in $\mathbb{G}_q$.

$\underline{\mathsf{DLEQ}(g_1, h_1, g_2, h_2)}$

Input: $g_1, h_1, g_2, h_2 \in \mathbb{G}_q$, $\alpha \in \mathbb{Z}_q$

Output: $\pi = (e, s)$

1. $A_1 = g_1^w, A_2 = g_2^w$ for $w \xleftarrow{\$} \mathbb{Z}_q$
2. $e = H(h_1, h_2, A_1, A_2)$
3. $s = w - \alpha \cdot e \pmod{q}$
4. $\pi = (e, s)$

$\underline{\mathsf{DLEQ\text{-}Verify}(g_1, h_1, g_2, h_2, \pi)}$

Input: $g_1, h_1, g_2, h_2 \in \mathbb{G}_q$, $\pi = (e, s)$

Output: $b \in \{0, 1\}$

1. $A'_1 = g_1^s h_1^e, A'_2 = g_2^s h_2^e$
2. $e' = H(h_1, h_2, A'_1, A'_2)$
3. $b = \begin{cases} 1 & \text{if } e' = e \\ 0 & \text{otherwise} \end{cases}$

3 Design Space of Distributed Randomness Beacons

3.1 Context

3.1.1 System Model

We consider a system with a fixed set of n participants $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ (also called nodes). We may also write $\mathcal{P} = \{1, 2, ..., n\}$ for the purpose of algebraic formulations. Of the n, up to t nodes may be *faulty* (also called *malicious* or *Byzantine*) and engage in incorrect (arbitrary) behavior during a protocol run. An adversary $\mathcal{A}$ that controls up to t such nodes is called *t-limited*. Otherwise, nodes that are *honest* abide by the specified protocol.

We assume a standard public key infrastructure (PKI) such that all nodes know each others' public keys, and that all nodes are connected via point-to-point secure (providing authenticity) communication channels. All messages exchanged by honest nodes are digitally signed by the sender, and recipients always validate each message before proceeding. By default, we assume a *synchronous* network, in which there exists some known finite message delay bound Δ. This means that an adversary can delay a message by at most Δ.

Moreover, we assume a computationally bounded adversary $\mathcal{A}$ which runs in probabilistic polynomial time (PPT). In particular, this means $\mathcal{A}$ cannot break standard cryptographic primi-

tives such as hash functions, digital signatures, etc. For delay-based protocols, we also assume the adversary cannot compute delay functions in fewer than T time steps. The three ways in which $\mathcal{A}$ can deviate from a protocol are omitting a message (i.e. *withholding attack*), sending invalid messages, and colluding to coordinate an attack based on private information shared among malicious nodes. Additionally, $\mathcal{A}$ has the power to perform a *grinding attack*, in which $\mathcal{A}$ privately precomputes and iterates through polynomially many combinations of inputs to an algorithm in order to derive a desirable output. By default, we assume a (t-limited) *static* adversary that chooses nodes to be corrupted before a protocol run whereas an *adaptive* adversary can choose nodes to be corrupted at any time during a protocol run (we assume a model where nodes remain corrupted once corrupted).

We denote our computational model's security parameter by λ. We call a function $\text{negl}(\lambda)$ *negligible* if for all $c > 0$ there exists a λ_0 such that $\text{negl}(\lambda) < \frac{1}{\lambda^c}$ for all $\lambda > \lambda_0$. The group elements $g, h \in \mathbb{G}$ are generators of $\mathbb{G}$ while p, q denote primes where $q \mid p - 1$ (unless stated explicitly) such that $\mathbb{G}_q$ is a group of prime order q. The notation $tuple[0]$ denotes the first element of $tuple$. Furthermore, we model any hash function $H(\cdot)$ as a random oracle [Bellare and Rogaway, 1993]. In the context of a distributed randomness beacon, we use τ to denote epoch number and Ω_τ to denote the *beacon output* (i.e. the distributed randomness output) in epoch τ. The *entropy-providing committee* denoted by C_τ refers to a subset of nodes (hereafter called *entropy providers*) that proactively generate and provide marginal entropy in epoch τ.

3.1.2 Strawman Protocol: Rock-Paper-Scissors

Distributed randomness assuming perfect synchrony ($\Delta = 0$) is straightforward. Consider the following one-round protocol where each participant i broadcasts its *entropy contribution* (i.e. independently generated randomness) $e_i \xleftarrow{\$} \mathbb{Z}_p$ to every other participant at the same time.

The protocol's random output Ω is calculated (via modular addition in $\mathbb{Z}_p$) as:

$$\Omega = \sum_{i=1}^{n} e_i \tag{3.1}$$

Repeating this protocol periodically would yield a DRB. This protocol is simple—in fact, it is essentially what humans approximate when playing rock-paper-scissors (with $e_i \xleftarrow{\$} \mathbb{Z}_3$). Under perfect synchrony, it is secure as long as any single participant chooses its e_i randomly. However, security falls apart completely once messages can be delayed. Consider a simple scenario with three participants $\{P_1, P_2, P_3\}$ producing $\Omega = e_1 + e_2 + e_3$. If P_3 can read e_1 and e_2 before sending e_3 (due to non-zero message latency) to P_1 and P_2, then P_3 can fix the output Ω to any value $\tilde{\Omega}$ by choosing $e_3 = \tilde{\Omega} - e_1 - e_2$. Effectively, the protocol cannot tolerate any malicious participants without perfect synchrony. Indeed, humans may attempt to cheat in rock-paper-scissors by quickly adjusting their play in reaction to what their opponent is playing.

3.1.3 Commit-Reveal

A classic fix for the above synchrony problem is to introduce a cryptographic commitment step before each party reveals its entropy contribution.

1. Commit. Each participant P_i broadcasts a cryptographic commitment $c_i = \mathrm{Com}(e_i, r_i)$ (with fresh randomness r_i) to its entropy contribution e_i rather than e_i itself. Note that $\mathrm{Com}(x, r_0)$ denotes a cryptographic commitment to x with hiding and binding properties [Blum, 1983, Damgård, 1998]. If participants sample e_i from a suitably large space, it is also secure to simply publish $c_i = H(e_i)$.

2. Reveal. Once all participants have shared their corresponding commitments, each participant P_i then opens its commitment by revealing the pair (e_i, r_i). In turn, P_i verifies each received pair (e_j, r_j) for $j \neq i$ by recomputing $c_j = \mathrm{Com}(e_j, r_j)$. Given that these checks

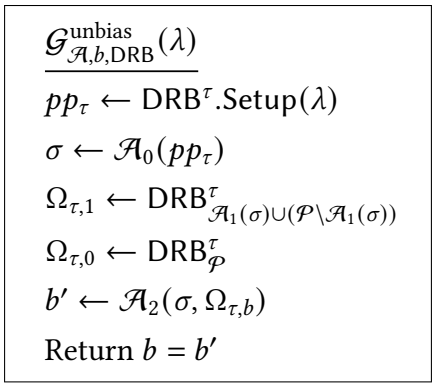

Figure 3.1: Security game for DRB unbiasability.

pass, the final output Ω can be computed as in Equation 3.1. If any of the checks do not pass, however, the protocol aborts and outputs $\perp$.

With the additional commit step, it becomes impossible for any participant to manipulate the output Ω, as the contribution values are bound by commitments published before any participants reveal. Nonetheless, the protocol can still be biased, as the last-revealing participant P_k can in fact compute Ω earlier than others and hence can decide to withhold (not reveal) (e_k, r_k) if Ω is not to its liking. This is called the *last-revealer attack*. Note that this attack is indistinguishable from an honest node going offline, and indeed the protocol in this basic form also has no robustness against non-Byzantine faults.

3.1.4 Ideal Distributed Randomness Beacons

Clearly, a DRB should prevent any one participant from tampering with (e.g. predicting, biasing, or aborting) the output. We formalize the security properties of an ideal DRB as follows.

Definition 3.1 (Ideal distributed randomness beacon)**.** An ideal distributed randomness beacon satisfies the following security properties:[1]

[1]We present a game-based security definition here, as it is the most commonly used in the literature. Other formulations, such as ideal functionalities as used in UC-security [Canetti, 2001], are possible.

1. Unbiasability. A DRB is *unbiasable* if, for any PPT adversary $\mathcal{A} = (\mathcal{A}_0, \mathcal{A}_1, \mathcal{A}_2)$, the adversary's advantage in the game depicted in Figure 3.1, given by

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

   is negligible, i.e. $\mathrm{Adv}^{\mathrm{unbias}}_{\mathcal{A},\mathrm{DRB}}(\lambda) \leq \mathrm{negl}(\lambda)$.

2. Liveness. We define liveness [Cherniaeva et al., 2019] by requiring that the advantage of $\mathcal{A}$ denoted by $\Pr[\Omega_\tau = \perp]$ (i.e. the probability that the beacon output at the end of epoch τ is null) is negligible, given a DRB that runs among honest participants and $\mathcal{A}$.

3. Unpredictability. We define two types of unpredictability. Suppose a DRB's epoch τ starts at time $T_{\tau,0}$ and finalizes (Ω_τ becomes publicly available) at $T_{\tau,1}$ in the optimistic case (if every node is honest and online) and at $T_{\tau,2}$ in the worst case.

   - A DRB is *α-intra-unpredictable* ($\alpha > 0$) if $\mathcal{A}$ participating in DRB^τ (Figure 3.1) cannot predict any property of Ω_τ at time $T_{\tau,2} - \alpha$ with non-negligible advantage:

$$\Pr[\Omega_\tau \in Y] \leq \frac{|Y|}{2^{\ell_{\Omega_\tau}(\lambda)}} + \mathrm{negl}(\lambda)$$

     where Y denotes the set of possible values predicted by $\mathcal{A}$ and $\ell_{\Omega_\tau}(\lambda)$ denotes the bit-length of Ω_τ. It is *α-intra-predictable* otherwise.

   - A DRB is *β-inter-unpredictable* ($\beta \geq 1$) if $\mathcal{A}$ cannot predict any property of $\Omega_{\tau+\beta'}$ (as defined above) for any $\beta' \geq \beta$ before $T_{\tau,2}$ with non-negligible advantage.

Unbiasability. In the unbiasability game $\mathcal{G}^{\mathrm{unbias}}_{\mathcal{A},b,\mathrm{DRB}}$ depicted in Figure 3.1, the adversary algorithm $\mathcal{A}_0$ first precomputes an advice string (σ) given the DRB's public parameters pp_τ in epoch τ (which may include the previous beacon output, a list of participants in epoch τ, the identity of the epoch leader, etc. depending on protocol specifications). Our definition is quite general in that the advice

string may encode any biasing "strategy." This string is taken by $\mathcal{A}_1$, which then (statically and within epoch τ) corrupts up to t nodes in the honest node set $\mathcal{P} = \{1, \ldots, n\}$, interacts with $\mathcal{P} \setminus \mathcal{A}_1$, and outputs the beacon output $\Omega_{\tau,1}$. In contrast, $\Omega_{\tau,0}$ denotes the honest beacon output generated by $\mathcal{P}$ (and not involving $\mathcal{A}_1$). For both, the notation $\mathrm{DRB}^{\tau}_{\tilde{\mathcal{P}}}$ denotes a DRB in epoch τ with a participant set $\tilde{\mathcal{P}}$. Then $\mathcal{A}_2$ distinguishes the two cases given the same advice string. A DRB is unbiasable if the adversary can do so with negligible probability. Our definition includes the possibility of *private* biasing, e.g. the adversary biases the result in a way that requires a secret key to detect. This means that the biased beacon outputs can even appear pseudorandom (indistinguishable from uniform distribution) to an outsider, a notion not considered in previous works [Bonneau et al., 2015, Bhat et al., 2021, Das et al., 2022].

Liveness. While liveness implies a notion called *guaranteed output delivery* [Schindler et al., 2020, Das et al., 2022] (all honest nodes receive Ω_τ at the end of epoch τ), it is in turn implied by unbiasability (due to the fact that $\Omega_{\tau,0}$ from Figure 3.1 is never null). For instance, commit-reveal, due to the last-revealer attack, does not satisfy liveness and thus is biasable.

Unpredictability. We note that β-inter-unpredictability (though in different forms) has been considered in previous works [Bhat et al., 2021, Bhat et al., 2023] for $\beta \geq 1$, but we extend the notion to "$\beta = 0$" and explicitly consider α-intra-unpredictability in conjunction with β-inter-unpredictability, in order to exhibit variations across all possible DRBs. While neither implies the other, both are defined using the same probability formulation (involving $\Pr[\Omega \in Y]$), which has not been considered in previous works [Bhat et al., 2021, Schindler et al., 2020, Das et al., 2022] and captures cases where (say) $\mathcal{A}$ predicts the first bit of Ω is 1 (in which case Y is a set of possible beacon output values whose first bit is 1), predicts the middle 10 bits make a prime number, etc. We also note that $\tilde{\beta}$-inter-unpredictability implies β-inter-unpredictability for all $\beta > \tilde{\beta}$, and that α-intra-unpredictability for all $\alpha > 0$ implies unbiasability. The reason is that biasability allows $\mathcal{A}$ in $\mathcal{G}^{\mathrm{unbias}}_{\mathcal{A},b,\mathrm{DRB}}$ to make a prediction towards its biasing strategy encoded in σ, implying that there exists $\alpha > 0$ such that the protocol is α-intra-predictable.

3.2 Delay-Based Protocols

One way to prevent the last-revealer attack is to compute Ω_τ using a *delay function* after combining each node's entropy contribution. If the delay is suitably long, no participant can predict what effect a potential contribution will have on the output before its contribution must be published. Typically, a *verifiable delay function* (VDF) [Boneh et al., 2018a, Boneh et al., 2018b] is used to accomplish this while maintaining efficient verifiability of the result.

Definition 3.2 (Verifiable delay function)**.** A *verifiable delay function* (VDF) is a function that takes a specified number of sequential steps to compute (even with a large amount of parallelism available) but takes significantly less time to verify. It is described by the following algorithms:

- $\mathsf{Setup}(\lambda, T) \rightarrow pp$ is a randomized algorithm that outputs public parameters pp given security parameter λ and delay parameter T[2].
- $\mathsf{Eval}(pp, x) \rightarrow (y, \pi)$ computes y in T sequential steps and (optionally) a proof π, given pp and an input x.
- $\mathsf{Verify}(pp, x, y, \pi) \rightarrow \{0, 1\}$ outputs 1 if y is the unique correct evaluation of the VDF on input x and 0 otherwise.

Two well-known VDF proposals, due to Pietrzak [Pietrzak, 2018] and Wesolowski [Wesolowski, 2019], make use of the (believed) inherently sequential nature of repeated squaring in a group of unknown order. VDFs can also be constructed from incrementally verifiable computation [Boneh et al., 2018a, Khovratovich et al., 2022, Kothapalli et al., 2022, Kothapalli and Setty, 2024, Arun and Setty, 2024], isogenies [De Feo et al., 2019, Chavez-Saab et al., 2021], and lattices [Lai and Malavolta, 2023, Cini et al., 2023, Osadnik et al., 2025]. VDFs can be used to derive unbiasable randomness either from existing, biasable protocols (e.g. commit-reveal or public implicit beacons) or as the building block for an entirely new protocol (like RandRunner [Schindler et al., 2021]).

[2] t denotes corruption threshold for a t-limited adversary in this chapter.

3.2.1 Modifying Commit-Reveal

The Unicorn protocol [Lenstra and Wesolowski, 2015] uses the Sloth function (a VDF precursor based on computing square roots modulo a prime) in a manner similar to commit-reveal. In fact, commitments are no longer needed; participants simply publish their entropy contributions directly. Unicorn can be improved using a modern VDF in place of Sloth to achieve faster (constant time) verification time for a given delay parameter. We refer to VDF-enhanced Unicorn as *Unicorn++*. It runs as follows:

1. **Collect.** Every participant P_i broadcasts its entropy contribution e_i between time T_0 and T_1 (assuming synchronized clocks). At T_1, they are combined into $x_\tau = H(e_1, \ldots, e_n)$.

2. **Evaluate.** Some party evaluates the VDF with x_τ and a chosen delay parameter T (part of pp) via

$$y_\tau, \pi_\tau = \mathsf{VDF.Eval}(pp, x_\tau)$$

   such that $\Omega_\tau = H(y_\tau)$, which is posted and can be efficiently verified by any observer using π_τ via VDF.Verify.

As long as T is longer than the duration of $T_1 - T_0$, Unicorn++ successfully defends against any attack possible by the last entropy provider. Also desirably, it is unbiasable by an adversary that controls $n - 1$ of the participants, as even one honest entropy contribution requires computation of VDF.Eval from scratch. The downside of the protocol is that *somebody* must evaluate the VDF, which is slow by design. It is possible to outsource this computation, even in a decentralized manner [Thyagarajan et al., 2021b], as it does not matter for security who evaluates since VDFs are deterministic and verifiable.

We note a variation of above [Bonneau et al., 2015, Bünz et al., 2017], replacing or bolstering the participant entropy contributions with stock prices [Clark and Hengartner, 2010] or PoW blockchain headers [Nakamoto, 2008, Bentov et al., 2016, Han et al., 2020] (which are otherwise

susceptible to manipulation) to supply x_τ in Collect. Such schemes are collectively denoted by *Ext. Beacon+VDF* in Table 3.1. Unfortunately, they do not easily compare to other DRBs, as the security model depends on the cost of manipulating the external beacon, which has not yet been formally analyzed.

Cornucopia (Chapter 5) presents an optimization where a cryptographic accumulator is harnessed to capture all user contributions succinctly. HeadStart [Lee et al., 2022] is similar, uses Merkle trees, and emphasizes a multi-round pipelined protocol to scale up Unicorn++ while reducing latency. Cornucopia adopts the same conceptual approach as HeadStart, but the approach differs in that Cornucopia offers a generic construction from any accumulator and develops precise security notions that are needed.

3.2.2 Adding Recovery to Commit-Reveal

Another way to modify commit-reveal is to leverage a different class of delay functions called *timed commitments* [Boneh and Naor, 2000] in place of regular commitments used in commit-reveal. The idea is simple: timed commitments are commitments with an additional slow recovery process (the committed value can be recovered in T sequential steps but not before) in case the committer withholds.

Definition 3.3 (Timed commitment)**.** A *timed commitment* is a commitment with an additional algorithm whereby the committed value can be recovered (or *forced open*) in T sequential steps but not before:

- $\mathsf{ForceOpen}(c) \longrightarrow (x, r_0)$ outputs the committed value (x, r_0) in T sequential steps given a commitment $c = \mathsf{Com}(x, r_0)$.

This recovery process avoids the last-revealer attack, as a withholding participant's contribution can be recovered. Thus, the resulting distributed randomness protocol can be seen as a "commit-reveal-recover" protocol (Section 3.4). This approach was suggested by Boneh and

Naor [Boneh and Naor, 2000] though not specified in detail. Thyagarajan et al. [Thyagarajan et al., 2021a] proposed leveraging *homomorphic* timed commitments to combine contributions and only require one delay function even if *all* participants refuse to open (rather than one computation per withholder). Bicorn (Chapter 4) realizes the above logic in a simple, efficient DRB with comparable overhead to basic commit-reveal.

The advantage of Bicorn over Unicorn++ is that in the optimistic case (where every participant is honest) the protocol has no delay, analogous to a simple commit-reveal. Context may be important to consider if the optimistic case is unlikely to occur (e.g. due to poor network conditions or too many participants), in which case Unicorn++ is simpler and also requires only one delay function computation.

All of the protocols in this family share a fundamental predictability downside: if only one participant (or a colluding coalition) withholds while all others reveal, then the attacker(s) can simulate the optimistic case and learn Ω_τ early. As this implies T-intra-unpredictability (with delay parameter T), a protocol should consider Ω_τ to be potentially available to adversaries as of $T_{\tau,1}$ (optimistic case), even if it is not publicly known until $T_{\tau,2}$ (worst case).

3.2.3 Chain of VDFs

A disadvantage of above approaches is that each epoch may require consensus [Castro and Liskov, 1999] on inputs to the delay function, incurring communication cost. Also, the rate at which beacon outputs are generated is limited by T (by default in Unicorn++ and in Bicorn's ForceOpen case). RandRunner [Schindler et al., 2021] tackles these issues by leveraging a VDF design that builds a deterministic chain of outputs (more precisely, a chain of n interleaved VDFs each set up by a node) to bypass per-epoch consensus while allowing each epoch's duration to be independent of T in the optimistic case.

Namely, RandRunner uses Pietrzak's VDF [Pietrzak, 2018], where knowledge of a *trapdoor* allows an efficient evaluation of a VDF without T sequential steps unlike VDF.Eval (but with

T steps otherwise). In its setup, each P_i broadcasts pp_i (each corresponding to a different VDF per participant). Then the idea is that, in each epoch, $H(\Omega_{\tau-1})$ is input to the VDF of the epoch leader—selected via either *round-robin* (i.e. taking turns in some permuted order) or *random selection* (i.e. using $\Omega_{\tau-1}$ as seed), discussed in Section 3.5.1.1. In the optimistic case, the honest leader (the only one that knows its trapdoor) efficiently evaluates and publishes the VDF output as the beacon output while in the faulty case (if the leader withholds), others can evaluate the same value albeit more slowly.

Thus, RandRunner optimistically generates each beacon output rapidly with only $O(n)$ communication complexity. Adversarial leaders can increase the epoch duration to T and the communication complexity to $O(n^2)$.[3] The protocol exhibits two other beneficial properties. First, liveness is retained even with a dishonest majority and when network connectivity breaks down completely, as one can simply compute the beacon outputs over time via VDF.Eval. Second, it is impossible to bias the beacon once bootstrapped such that even the strongest adversary can only predict but not bias. The trade-off is that RandRunner can never achieve the ideal 1-inter-unpredictability property due to the existence of leaders that can withhold and adversaries with higher compute power. In other words, β-inter-unpredictability can be achieved only with $\beta > 1$, though β can be bounded [Schindler et al., 2021] with assumptions.

3.3 Commit-Reveal-Punish

Another approach to preventing last-revealer attacks is *commit-reveal-punish*, which assumes that all participants are rational entities and use financial penalties to discourage withholding. This requires some form of *escrow* (e.g. smart contracts on Ethereum [Wood et al., 2014]) to collect initial deposits from the participants which can be *slashed* (destroyed or redistributed) if misbehavior is detected. Commit-reveal-punish schemes defend against the last-revealer attack

[3] We assume consensus at a protocol level incurs $O(n^2)$ communication cost (bitwise) by default.

either by forcing every participant to reveal [Andrychowicz et al., 2014, Bentov and Kumaresan, 2014, RANDAO, 2016] or by tolerating some number of withholding participants via threshold commit-reveal [Yakira et al., 2020]. These two approaches are summarized below.

3.3.1 Enforcing Every Reveal

Extending basic commit-reveal, RANDAO ('16) [RANDAO, 2016] implements commit-reveal-punish in a straightforward way. Each participant is required to deposit coins at the time of commitment, which are slashed if that participant withholds its value during the reveal phase. The drawback of this approach is twofold. First, honest failures are indistinguishable from withholding and must also be punished. Attackers might exploit this by trying to block victim nodes from publishing (e.g. by bidding up the price of gas in a smart contract platform). Second, a high deposit of $O(n^2)$ coins is required to ensure fairness [Andrychowicz et al., 2014, Bentov and Kumaresan, 2014]. Thus, RANDAO ('16) is suitable only if participants are expected to be highly available and possess an ample supply of coins. Practical deployment also requires understanding of the value to participants of manipulating the beacon (to ensure the opportunity cost of lost deposits is higher). This assumption is reasonable for applications such as a lottery but may not apply for a public beacon whose use is not known in advance.

3.3.2 Rational Threshold Commit-Reveal

Economically Viable Randomness (EVR) [Yakira et al., 2020] provides an alternative requiring *constant* deposits while tolerating (honest) faults to an extent. This is achieved by devising a threshold variant of commit-reveal (i.e. in which $t+1$, as opposed to all n, nodes reveal to compute Ω_τ) and having an incentive mechanism around it. The threshold nature also invites collusion, which is counteracted by EVR's *informing* mechanism: if the escrow is notified of collusion (via informing), it rewards the informer and slashes the deposits of all others (*collective punishment*).

Realizing this, nodes are discouraged to collude, fearing another node within the coalition would inform.

EVR requires multiple cryptographic building blocks (introduced here, as they are used in other DRBs throughout the paper). EVR uses Escrow-DKG [Yakira et al., 2019], an extension of DKG (distributed key generation) [Pedersen, 1991b, Gennaro et al., 1999, Groth, 2021, Gurkan et al., 2021], to realize a threshold commit-reveal. DKG allows a set of n nodes to collectively generate a pair (sk, pk) of group secret and public keys such that sk is shared and "implied" (i.e. never computed explicitly) by n nodes via the following building blocks.

Definition 3.4 ((t, n)-secret sharing)**.** The dealer in a (t, n)-*secret sharing* shares a secret to n participants such that any subset of $t + 1$ or more participants can reconstruct the secret, but smaller subsets cannot.

Definition 3.5 (Shamir's secret sharing)**.** A concrete realization of (t, n)-secret sharing, *Shamir's secret sharing* [Shamir, 1979] allows a dealer to share a secret $s = p(0)$ for some *secret sharing polynomial* $p \in \mathbb{Z}_q[X]$ of degree t among n participants each holding a *share* $s_i = p(i)$ for $i = 1, ..., n$. Any subset of $t + 1$ or more participants can reconstruct the secret s via *Lagrange interpolation* (Section 2.16.1), but smaller subsets cannot. In this paper, we use (t, n)-secret sharing and Shamir's secret sharing interchangeably.

Definition 3.6 (Verifiable secret sharing)**.** *Verifiable secret sharing* (VSS) [Feldman, 1987, Pedersen, 1991a] protects a (t, n)-secret sharing scheme against a malicious dealer sending incorrect shares by enabling verification of each share. VSS can be described by the following algorithms (see Section 2.8 for details):

- $\text{Setup}(\lambda) \rightarrow pp$ generates the public parameters pp, an implicit input to all other algorithms.
- $\text{ShareGen}(s) \rightarrow (\{s_i\}, C)$ is executed by the dealer with secret s to generate secret shares

$\{s_i\}$ (each of which is sent to node i correspondingly) as well as commitment C to the secret sharing polynomial of degree t.

- $\mathsf{ShareVerify}(s_i, C) \rightarrow \{0, 1\}$ verifies share s_i using C.
- $\mathsf{Recon}(A, \{s_i\}_{i \in A}) \rightarrow s$ reconstructs s via Lagrange interpolation from a set A of $t + 1$ nodes that pass ShareVerify.

Definition 3.7 (Distributed key generation)**.** A *distributed key generation* (DKG) [Pedersen, 1991b, Gennaro et al., 1999] allows n participants to collectively generate a *group public key* (with an implicit *group secret key*), *individual secret keys*, and *individual public keys* without a trusted third party. It does so by running n instances of VSS (with each participant acting as a dealer for its independent secret):

- $\mathsf{DKG}(1^\lambda, t, n) \rightarrow (sk_i, pk_i, pk)$ outputs the i-th node's secret key, its public key (e.g. $pk_i = g^{sk_i}$), and a group public key pk (e.g. $pk = g^{sk}$) for an implicit group secret key sk given security parameter 1^λ, t, and n.

The intuition behind DKG is that it allows $t + 1$ (but not less) out of n nodes to jointly *use* sk via Lagrange interpolation without necessarily *knowing* sk. See Section 2.9 for details. Unlike secret sharing schemes, one DKG setup can lead to a number (at least polynomial in the security parameter) of usages, as the group secret key is never computed explicitly during normal use.

The crux of EVR is adopting Escrow-DKG. It is first different from a classic DKG in that an escrow platform Escrow (e.g. smart contract) disincentivizes misbehavior. Second and more importantly, Escrow-DKG's implicit group secret key sk is in fact the beacon output that becomes computed and publicized (unlike traditional DKGs in which the group secret key is never revealed). EVR proceeds in four phases:

1. **Setup.** Every participant registers by depositing 1 coin per secret (i.e. entropy contribution), and Escrow accordingly sets the threshold parameter $t = 2n/3$ required for Escrow-

DKG. It also sets the *illicit profit bound* (i.e. extra profit an adversary can gain as a result of using EVR's output as opposed to an ideal beacon) to $n - t = n/3$ and the *informing reward* to n.

2. **Commit.** Escrow-DKG is run, and each participant ends up with an individual key pair (sk_i, pk_i) as well as pk.

3. **Inform.** Any colluding participant that preemptively knows Ω_τ is incentivized to inform Escrow to earn an informing reward obtained via collective punishment.

4. **Reveal.** $\Omega_\tau = sk$ is reconstructed once $t + 1$ (or more) participants reveal their sk_i's. Initial deposits are returned after verification by Escrow. If Ω_τ is not reconstructed by the end, Escrow also initiates collective punishment.

While a malicious node in EVR might withhold to abort the protocol during Reveal or collude to learn Ω_τ before Reveal, security comes from the fact that both are disincentivized. First, setting the illicit profit bound to $n - t$ makes withholding unprofitable, as the $n - t$ or more participants needed to successfully abort EVR would earn an amount bounded by the illicit profit bound at the cost of losing their deposits. This prevents biasability. Second, setting the informing reward to n makes informing more profitable than any illicit profit. Thus, any coalition of nodes colluding to preemptively learn Ω_τ is economically unstable, as all nodes are incentivized to defect and act as an informer. This prevents predictability.

Despite the benefits of the threshold nature and constant deposits enabling a flexible incentive mechanism, EVR requires further economic assumptions beyond those needed for commit-reveal-punish. Specifically, EVR assumes a limit on illicit profit and a bound on the total number of coins $n/3$ (a participant with more coins than this is not allowed to join EVR as per decentralization assumption [Yakira et al., 2020]).

3.4 Commit-Reveal-Recover

Without using escrow to enforce desired behavior, *commit-reveal-recover* variants defend against the last-revealer attack by providing a mechanism to *recover* or *reconstruct* a participant's entropy contribution if withheld. This can be achieved by either *threshold secret sharing* or *threshold encryption*. Protocols based on commit-reveal-recover assume a t-limited adversary and require the cooperation of at least $t + 1$ nodes to reconstruct such that two desirable properties are achieved simultaneously: there is no need for all n nodes to reveal while any subset of t Byzantine nodes cannot collude to preemptively reconstruct. Note that this creates an inherent trade-off: while a smaller value of t helps tolerate more honest faults, it also means that a smaller subset can collude to predict the beacon output in advance.

3.4.1 From Threshold Secret Sharing

Commit-reveal-recover variants often use *publicly verifiable secret sharing* (PVSS) [Schoenmakers, 1999, Cascudo and David, 2017] as a subprotocol in order to allow any external party (not just the participants) to verify the correctness of sharing and reconstruction.

Definition 3.8 (Publicly verifiable secret sharing)**.** *Publicly verifiable secret sharing* (PVSS) is a VSS with the following additional algorithms to enable public verification: PVSS.KeyGen (which generates a key pair per participant), PVSS.Enc (for public-key encryption), and PVSS.Dec (decryption). The idea is that PVSS.ShareGen uses above to encrypt and decrypt PVSS shares and also to generate public proofs, e.g. non-interactive zero-knowledge (NIZK) proofs. Then PVSS.ShareVerify can be run by anyone (not just the participants). See Section 2.10 for details.

The idea in these commit-reveal-recover variants is that each participant generates a secret (i.e. entropy contribution), distributes PVSS shares to each other participant, and receives n respective shares of n other participants' secrets. These shares are then used to compute Ω_τ via

Lagrange interpolation[4] (PVSS.Recon) in case some nodes withhold. Based on when and how such Lagrange interpolation takes place, we subdivide the protocols into the following categories: commit-reveal-recover, share-reconstruct-aggregate, and share-aggregate-reconstruct.

3.4.1.1 Commit-Reveal-Recover

Extending commit-reveal, *commit-reveal-recover* adds a step to the commit phase where every participant is additionally required to distribute PVSS shares of its corresponding secret so that others can reconstruct it via Lagrange interpolation (*recover*) if withheld. The trade-off is additional communication cost, which amplifies if $O(n)$ Lagrange interpolations need to take place. Scrape [Cascudo and David, 2017] adopts this technique.

Scrape. With its own PVSS scheme [Cascudo and David, 2017] designed for efficiency, Scrape runs as follows after the initial generation (PVSS.KeyGen) of (sk_i, pk_i) for each of the n nodes:

1. **Commit.** Every node P_j runs PVSS.ShareGen($s^{(j)}$) as a dealer and publishes the encrypted shares $\mathsf{Enc}(pk_i, s_i^{(j)})$ for $i \in [n]$ and encryption proofs. P_j also publishes a commitment to the secret exponent $\mathsf{Com}(s^{(j)}, r_j)$ (with fresh randomness r_j). Upon receiving encrypted shares and proofs, all nodes run PVSS.ShareVerify to verify correct encryption. Let C_τ be the set of nodes with published commitments and valid shares.

2. **Reveal.** Once $t+1$ nodes have distributed their commitments and valid shares, every node P_j, $j \in C_\tau$, opens its commitment by revealing $(s^{(j)}, r_j)$.

3. **Recover.** For every node $P_a \in C_\tau$ that withholds $(s^{(a)}, r_a)$ in Reveal, other nodes P_j for $j \neq a$ reconstruct $h^{s^{(a)}}$ via PVSS.Recon, which requires each node to publish its decrypted share $h^{s_j^{(a)}}$ and the proof of correct decryption passing PVSS.ShareVerify.

4. **Aggregate.** The final randomness is $\Omega_\tau = \prod_{j \in C_\tau} h^{s^{(j)}}$.

[4] In this paper, we assume one Lagrange interpolation at a protocol level incurs $O(n^2)$ and $O(n^3)$ communication cost (bitwise) in the optimistic and worst cases, respectively.

Note that Scrape, in the optimistic case (without Recover), is just a commit-reveal with $O(n^2)$ PVSS shares distributed in the network during commit, $O(n)$ per node. In the worst case (with Recover), it requires an entirely new round of communication and potentially $O(n)$ Lagrange interpolations.

Albatross. Extending Scrape, Albatross [Cascudo and David, 2020] provides an improved amortized communication complexity of $O(n)$ per beacon output by generating a batch of $O(n^2)$ beacon outputs per epoch (as opposed to one). This is achieved by two techniques: packed Shamir secret sharing and linear t-resilient functions. As packed Shamir secret sharing allows sharing of $O(n)$ secrets (as opposed to one) per instance while linear t-resilient functions allow outputting of $O(n)$ values (as opposed to one) in the final randomness aggregation step, each of these techniques multiplicatively contributes $O(n)$ to the number of beacon outputs produced per epoch.

3.4.1.2 Share-Reconstruct-Aggregate

Another approach is to skip the commit-reveal phase and by default reconstruct each secret shared via PVSS. In other words, all nodes can distribute their PVSS shares (*share*), perform Lagrange interpolation per secret for a total of $O(n)$ times (*reconstruct*), and aggregate the interpolated secrets to output Ω_τ (*aggregate*). While the resulting *share-reconstruct-aggregate* saves a round of communication (Reveal) from Scrape's worst case, its average case does incur substantial communication cost due to $O(n)$ Lagrange interpolations, each of which requires cooperation of $t + 1$ nodes. Hence, this approach is preferable when it can be assumed that most epochs will require recovery due to faulty participants. RandShare [Syta et al., 2017] uses this technique.

3.4.1.3 Share-Aggregate-Reconstruct

Another alternative is to harness the homomorphic property of PVSS, due to which only one, as opposed to $O(n)$, Lagrange interpolation reconstructs Ω_τ if nodes perform *aggregate* before *reconstruct*, hence *share-aggregate-reconstruct*. SecRand [Guo et al., 2020] uses this technique

to optimize communication complexity simply, and Rondo [Meng et al., 2025] takes it further by applying more sophisticated optimizations via their version of VSS with Bulletproofs [Bünz et al., 2018] and batching (which they call batched asynchronous VSS with partial output) to achieve $O(n^2 \log n)$ communication complexity in the worst case.

3.4.2 From Threshold Encryption

While protocols based on threshold secret sharing can incur high communication cost of $O(n^4)$ due to $O(n)$ Lagrange interpolations, protocols relying on a different cryptographic primitive, namely threshold encryption [Desmedt and Frankel, 1990] (which does not rely on PVSS), offer a variant where only one Lagrange interpolation suffices even in the worst case. Though reminiscent of share-aggregate-reconstruct, these protocols differ in that they require a DKG, which may be run multiple times to refresh keys. In this section, we summarize how a protocol like HERB [Cherniaeva et al., 2019] uses threshold encryption to construct a DRB.

The main idea is simple: n participating nodes run a DKG, encrypt their respective entropy contributions under the group public key pk, homomorphically combine all ciphertexts into one group ciphertext, and jointly (requiring at least $t + 1$ nodes) decrypt the group ciphertext via one Lagrange interpolation. Effectively, the DKG is what makes this possible, as it allows the usage of sk (to decrypt a ciphertext under pk) without knowing it (recall from Definition 3.7).

HERB achieves a communication complexity of $O(n^2)$ and $O(n^3)$ in the optimistic and worst cases, respectively. Its requirement of DKG in the setup presents a caveat however, as a new DKG must take place for any attempt to refresh keys of participants, e.g. in case of a suspected hack or a simple *reconfiguration* (in which the set of participants changes). This can incur additional cost per DKG.

3.5 Committee-Based Protocols

All aforementioned commit-reveal variants include every node in the entropy-providing committee C_τ for every epoch. Incorporating marginal entropy from all nodes scales poorly with large numbers of participants, and hence a natural optimization is to select a smaller subset of nodes to contribute marginal entropy in each epoch (i.e. reduce $|C_\tau|$).

In this section, we consider DRBs that are committee-based, with C_τ such that $1 \leq |C_\tau| < n$. Committee-based protocols proceed in two steps: *committee selection* and *beacon output generation*. As the names suggest, C_τ is agreed upon during committee selection while the beacon output Ω_τ is generated and agreed upon during beacon output generation. We observe that committee selection and beacon output generation are, at least theoretically, modular such that subprotocols can be independently chosen for the two components. We visualize these two dimensions of committee-based DRBs in Table 3.2. We also observe that the protocols introduced so far (e.g. commit-reveal-recover) can be used as a module in a larger committee-based protocol, with the chosen committee executing the chosen protocol in each epoch.

3.5.1 Step 1. Committee Selection

The first step of a committee-based DRB involves selecting C_τ in a way agreed by all nodes. We classify committee selection mechanisms into two: public and private.

3.5.1.1 Public Committee Selection

In a *public committee selection*, only public information is needed to derive C_τ.

Round-Robin (RR). A simple example is *round-robin* (RR), in which nodes simply take predetermined turns being selected. While RR can work with committees of any size, typically RR is used to select a committee of size one (i.e. a leader) corresponding to node $i \equiv \tau \pmod{n}$. Protocols like BRandPiper [Bhat et al., 2021] (in which the epoch leader is the only active entropy provider)

adopt RR as their leader selection mechanism due to its innate fairness property [Azouvi et al., 2018] (also known as chain quality [Garay et al., 2015] in the blockchain context) where all nodes, by RR's definition, take equal leadership.

Random Selection (RS). A second example is *random selection* (RS), which uses some public randomness (most commonly the last beacon output $\Omega_{\tau-1}$) to derive C_τ. In HydRand [Schindler et al., 2020] and GRandPiper [Bhat et al., 2021], C_τ consists of a node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ where $\tilde{n}$ is the number of eligible nodes. Ouroboros [Kiayias et al., 2017] uses a similar process called follow-the-satoshi [Bentov et al., 2014,Kiayias et al., 2017] which selects nodes weighted by stake. Ethereum's RANDAO ('25) [Edgington, 2023,Johnson et al., 2024,Oshitani and Drake, 2025] also uses the previous beacon output to randomly select nodes weighted by stake. HashRand [Bandarupalli et al., 2024] has a similar mechanism.

Randomized selection means that some nodes may, in theory, never be selected as entropy providers. A more serious concern is that an adversary can attempt to bias Ω_τ via grinding in order to bias $C_{\tau+1}$ (which can bias $\Omega_{\tau+1}$). In the worst case, this can lead to a vicious cycle in which an adversary controlling enough nodes on the current committee to manipulate the beacon output can ensure it will also control enough nodes on the next committee, and so on ad infinitum.

This is not an issue in RR, as its committee selection is deterministic and independent of the preceding beacon output. Nonetheless, a trade-off of RR is that denial-of-service (DoS) becomes indefinitely possible (for all epochs τ for $\tau > \tilde{\tau}$ given $\Omega_{\tilde{\tau}}$) since each committee is publicly known in advance. All in all, RR gains unbiasability (due to determinism) at the cost of indefinite DoS, while RS reduces the risk of DoS, i.e. that only for epoch $\tau + 1$ (due to randomization given Ω_τ), at the cost of potential grinding attacks.

Leader-Based Selection (LS). A third example, *leader-based selection* (LS) is a hybrid method that exhibits both determinism and randomization. It runs in two steps: the first step involves electing an epoch leader (either by RR or RS) while the second involves selection of C_τ by the elected leader. It is in this way that the mechanism is deterministic from the leader's perspective

while randomized from that of others.

One approach to limit the power delegated to the leader is that $|C_\tau|$ needs to be greater than t so that a malicious leader wouldn't be able to choose C_τ maliciously. RandHound [Syta et al., 2017], SPURT [Das et al., 2022], and OptRand [Bhat et al., 2023] demonstrate such LS.

- RandHound. As instantiated in RandHerd [Syta et al., 2017], RandHound's leader election (i.e. via RS as the first step of LS) involves a public lottery where each node generates a lottery ticket $H(C \,\|\, pk_i)$ given a public configuration parameter C (assuming its randomness) such that node $\arg\min_i H(C \,\|\, pk_i)$ becomes the leader (originally called client). In the second step of LS, RandHound adopts a form of sharding (involving PVSS groups). The leader selects more than a threshold number of nodes in each shard (PVSS group), guaranteeing a threshold number of entropy providers across all shards.

- SPURT and OptRand. Unlike RandHound, SPURT and OptRand adopt RR as the first step of LS, with nodes simply taking turns as an epoch leader. Then the leader chooses C_τ based on received encrypted messages.

Given an underlying DRB that utilizes a leader to orchestrate communication, LS is a natural choice to committee selection, as a leader helps mitigate the protocol's communication cost overall.

3.5.1.2 Private Committee Selection

In a *private committee selection*, also known as a *private lottery*, each node needs to input some private information (e.g. secret key) in order to check whether or not it has been selected into C_τ (i.e. has won a lottery to serve on the committee). The general formulation of a private lottery is given by

$$f_{priv}(\cdot) < target$$

where $f_{priv}(\cdot)$ is a lottery function (i.e. pseudorandom function) that takes some private input *priv* and *target* denotes the lottery's "difficulty level" (a la proof-of-work), which can be adjusted to make the lottery arbitrarily easy or hard to win.

Each node calculates $f_{priv}(\cdot)$ and checks if the above inequality is satisfied, in which case it "wins" the lottery and becomes an entropy provider. As an adversary can perform a grinding attack by trying many values of *priv* until a desirable function output is achieved, one crucial requirement is that *priv* should be provably committed in the past and thus be ungrindable at the time of computation of $f_{priv}(\cdot)$.

A prime example of a private lottery is one based on VRFs (verifiable random functions [Micali et al., 1999,Dodis and Yampolskiy, 2005]), which output a pseudorandom value (as well as a proof for verification) given secret key *sk* and input *x* (see Section 2.11). Most notably, Algorand [Gilad et al., 2017] uses VRFs to realize a lottery every epoch. Quite naturally, one's private input to $\mathsf{VRF}_{sk}(\cdot)$ is its secret key. The lottery[5] is given by

$$\mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel role) < target$$

where *role* is some parameter specific to Algorand. As both $\Omega_{\tau-1}$ and *role* are already public and ungrindable at the time of computation, Algorand makes sure *sk* is likewise ungrindable by requiring that *sk* is committed in advance. Similar private lotteries are used by Ouroboros Praos [David et al., 2018], Caucus [Azouvi et al., 2018] (where a hash chain replaces VRFs), and NV (from Nguyen-Van et al. [Nguyen-Van et al., 2019]). See Table 3.2 for details.

Private lotteries provide two notable benefits: resilience to DoS attack (due to its property of delayed unpredictability [Azouvi et al., 2018] where one cannot predict the eligibility of honest nodes until they reveal) and *independent participation* (i.e. nodes do not have to know other participants in advance to participate) allowing less communication cost as well as a more per-

[5]While there are multiple versions of Algorand, we consider its first version, as they do not differ fundamentally.

missionless setting. Nonetheless, it can introduce the possibility of biasing via withholding (as discussed in Section 3.7.2).

3.5.2 Step 2. Beacon Output Generation

Given a concrete committee C_τ, the next step is to output Ω_τ. While any aforementioned protocol may be run among nodes in C_τ to realize a DRB, other approaches provide different trade-offs. We classify variants which require *fresh* (independently generated on the spot) per-node entropy (contribution) and those which combine previous beacon output with *precommitted* (independently generated but precommitted, hence ungrindable) per-node entropy.

3.5.2.1 Fresh Per-Node Entropy

Beacon output generation approaches involving fresh (also referred to as true randomness [Cascudo et al., 2021, Das et al., 2022] as opposed to pseudorandomness) per-node entropy are typically commit-reveal-recover variants from Section 3.4. Some protocols in this family include the following:

Share-Reconstruct-Aggregate. In Ouroboros, nodes in C_τ (i.e. slot leaders of epoch τ) perform a RandShare-style share-reconstruct-aggregate using PVSS to output Ω_τ. RandHound uses a similar approach (facilitated by an epoch leader), and HashRand applies more sophisticated optimizations to this process via their version of VSS (which they call batched asynchronous weak VSS) to achieve $O(n^2 \log n)$ communication complexity in the worst case.

Share-Aggregate-Reconstruct. In SPURT, OptRand, and BRandPiper, nodes in C_τ perform a SecRand-style share-aggregate-reconstruct to output Ω_τ. BRandPiper has a twist: it utilizes the idea of buffering PVSS shares in advance. While there is one entropy provider per epoch, n secrets (one from each node) are combined such that it provides the ideal 1-inter-unpredictability property as opposed to t-inter-unpredictability (as in HydRand or GRandPiper). The trick is that each epoch leader generates n fresh secrets (entropy contributions) that become combined with

others' secrets in the next n epochs, respectively. In an epoch, one node distributes $O(n^2)$ PVSS shares (buffered by other nodes) whereas, in a typical share-aggregate-reconstruct like SPURT and OptRand, each of $O(n)$ nodes distributes $O(n)$ PVSS shares (with no buffering).

From Threshold Encryption. Similar to HERB, entropy providers in NV [Nguyen-Van et al., 2019] contribute their fresh entropy using ElGamal although they use its classical, non-threshold version due to NV's centralized model in which a third party called the Requester is the direct recipient of a beacon output. As a result, each entropy provider generates and encrypts its entropy and sends it to the Requester, which then decrypts all the messages received from entropy providers and outputs their sum as Ω_τ. Naturally, this Requester version of NV can be modified into what we call *NV++*, which differs from NV in two ways. First, nodes in C_τ (once finalized) can be made to perform HERB among themselves. This eliminates the existence of the centralized Requester. Second, entropy provision (i.e. broadcasting one's entropy) can be coupled with proof of membership to C_τ (i.e. broadcasting the fact that a node has won the VRF private lottery). In NV, these two are separate steps potentially incurring adaptive insecurity (a concept delineated in Section 3.7.3).

3.5.2.2 Combining Previous Output and Precommitted Per-Node Entropy

To optimize communication cost, one can require less input from entropy providers each epoch. The canonical optimization involves utilizing $\Omega_{\tau-1}$ as a source of entropy to produce Ω_τ. The caveat in doing so is that grinding may become possible once $\Omega_{\tau-1}$ becomes public, which is why it is necessary to require entropy contributions for epoch τ to be precommitted before combining with $\Omega_{\tau-1}$ to output Ω_τ. This prevents grindability while taking advantage of the convenience of $\Omega_{\tau-1}$. Such a requirement is observed in many committee-based protocols, though their details may seem unrelated on the surface.

- HydRand and GRandPiper. Each epoch, an entropy provider (i.e. epoch leader) in HydRand commits its entropy that becomes opened (revealed) in the next epoch it is selected as the

leader again. In other words, the epoch leader's precommitted entropy $e_{\tilde{\tau}}$ from its last epoch $\tilde{\tau}$ of leadership is the one that becomes combined with $\Omega_{\tau-1}$ in the form of $h^{e_{\tilde{\tau}}}$ to generate

$$\Omega_\tau = H(\Omega_{\tau-1} \parallel h^{e_{\tilde{\tau}}})$$

while PVSS recovery is used in case the leader fails to open $e_{\tilde{\tau}}$ in epoch τ. Notable in HydRand is the fact (achieving ungrindability of $h^{e_{\tilde{\tau}}}$) that one honest node must be present in any $t + 1$ consecutive epochs due to the requirement that a leader cannot gain another leadership in the next t epochs. Similar overall is GRandPiper's beacon output generation (see Table 3.2).

- Algorand and Ouroboros Praos. These schemes use a VRF for beacon output generation (rather than only for committee selection as in NV++). The secret key sk of the epoch leader often corresponds to precommitted per-node entropy as long as the assumption that nodes cannot switch their sk at the time of VRF's computation holds. Algorand's beacon output is given by

$$\Omega_\tau = \mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel \tau)$$

combining the previous output $\Omega_{\tau-1}$ with the precommitted entropy sk. Note that the input to the VRF in beacon output generation is different from that in committee selection, as the VRF output in committee selection is always going to be less than $target$ by design. Ouroboros Praos' beacon output is generated similarly (see Table 3.2).

- Caucus. Each new reveal (h_τ in epoch τ) from an entropy provider's private hash chain in Caucus corresponds to that node's precommitted entropy. The beacon output

$$\Omega_\tau = h_\tau \oplus \Omega_{\tau-1}$$

naturally follows its committee selection mechanism $H(h_\tau \oplus \Omega_{\tau-1}) < target$. See Table 3.2 for details.

Note on RANDAO. Historically, the term RANDAO has been an overloaded one. In this dissertation, we make the distinction between RANDAO ('16) [RANDAO, 2016] and RANDAO ('25) [Edgington, 2023, Johnson et al., 2024, Oshitani and Drake, 2025] to clarify. While the former reflects a commit-reveal-punish protocol (Section 3.3) implemented on Ethereum in 2016, the latter denotes the underlying mechanism of Ethereum's consensus layer (Beacon Chain) as outlined in EIP-7917 [Oshitani and Drake, 2025] (C_τ is selected more deterministically in advance) subsuming the Electra update [Johnson et al., 2024] (which makes updates to system parameters related to node balances but does not yet include EIP-7917's determinism). Even though the mechanism of RANDAO ('25) is quite different from that of the classic commit-reveal and commit-reveal-punish, it is still referred to as a type of commit-reveal where, as opposed to fresh entropy as in RANDAO ('16), a one-time precommitted entropy (i.e. secret key) is used to pseudorandomly "reveal" multiple (hashes of) BLS signatures over respective epochs.

3.6 Protocols With No Marginal Entropy

It is possible to devise a protocol where no node contributes any marginal entropy ($|C_\tau| = 0$) as the beacon runs, producing the beacon output solely via cryptographic pseudorandomness. This can improve efficiency as no node needs to generate and communicate fresh entropy. However, the beacon becomes predictable forever (β-inter-unpredictability fails for all β) if compromised (perhaps undetectably).

Such a DRB can be based on a *distributed verifiable random function* [Cachin et al., 2005, Galindo et al., 2021, Camenisch et al., 2022] (DVRF, also known as threshold VRF or TVRF [Cascudo et al., 2021]). The idea is that the VRF's sk is distributed among n nodes via DKG such that

$t + 1$ nodes can cooperate to compute a per-epoch VRF output (as well as its proof), as if the computation involves one master node with sk; see Section 2.12.

DVRF-based DRB. Each beacon output of a DVRF-based DRB is then given by

$$\Omega_\tau = \mathsf{DVRF.Combine}(A, \{\mathsf{DVRF.PartialEval}(sk_i, f(\Omega_{\tau-1}))\}_{i\in A})[0]$$

where sk_i denotes each node's secret key after a DKG and f denotes some deterministic function of $\Omega_{\tau-1}$.

The output is equivalent to one trustworthy master node with complete knowledge of sk computing the output as:

$$\Omega_\tau = \mathsf{VRF}_{sk}(f(\Omega_{\tau-1}))$$

There is no marginal entropy contributed by the participants, as f typically takes a form resembling $f(\Omega_{\tau-1}) = H(\tau \parallel \Omega_{\tau-1})$. The ideal 1-inter-unpredictability of the above DVRF formulation relies on the fact that no one node (or up to t nodes) can gain knowledge of sk to be able to compute and predict future beacon outputs.

DVRF-based DRB from a chain of unique signatures. Since taking the hash of a verifiable unpredictable function (VUF) [Micali et al., 1999] is equivalent to a VRF, a unique digital signature (which is a VUF [Dodis and Yampolskiy, 2005]) can be made into a DVRF by computing its threshold variant [Boldyreva, 2003] and hashing the output (assuming a hash function as a random oracle [Bellare and Rogaway, 1993]). Dfinity [Camenisch et al., 2022] and drand [drand, 2020] (while differing slightly in minor details) both use the BLS signature scheme [Boneh et al., 2001] to realize a DRB as

$$\Omega_\tau = H(\mathsf{Sign}_{sk}(\tau \parallel \Omega_{\tau-1}))$$

where $\mathsf{Sign}_{sk}(\cdot)$ is a threshold BLS signature computed by at least $t + 1$ nodes with sk as the implicit group secret key generated via DKG. The actual computation involves combining of

partial signatures computed using sk_i (see Section 2.16.2).

Variations on a chain of unique signatures. Besides a chain of BLS signatures, there exist several other variations.

- RandHerd [Syta et al., 2017]. Two modifications are made in RandHerd. First, a form of "sharding" into groups (each of size c) allows reduction of overall communication complexity. Second, the underlying signature scheme used is Schnorr instead of BLS. Each Ω_τ is a threshold Schnorr signature on message $m = t_\tau$ where t_τ denotes the timestamp at the epoch's beginning. As m can technically be chosen (and thus biased) by the leader, one simple improvement can be setting $m = \tau \parallel \Omega_{\tau-1}$ a la Dfinity or drand.

- DDH-DRB and GLOW-DRB [Galindo et al., 2021]. These two DRBs modify Dfinity-DVRF (i.e. each epoch of Dfinity) and explore space-time trade-off by using DLEQ NIZKs (Section 2.16.3) in place of pairing equations (Section 2.15). See Section 2.13 and 2.14 for details.

- Strobe [Beaver et al., 2023]. In Strobe, threshold RSA decryption conceptually replaces the threshold BLS process. Note that RSA decryption and BLS signature are similar in that one needs a secret value (decryption key d and signer's sk, respectively) to perform the respective operations. The analogy is that threshold BLS distributes sk in a threshold manner (via DKG) while threshold RSA distributes d (not via DKG). The difference is that the latter requires a trusted setup (knowledge of factors of N, the RSA modulus), and this is Strobe's main downside. Its benefit of using threshold RSA decryption is equally clear: the simple relationship $\Omega_\tau^d = \Omega_{\tau-1} \pmod N$ allows efficient generation of all past beacon outputs (a novel property of a DRB called *history generation*) and thus extremely efficient verification of the beacon history.

- GRandLine [Bacho et al., 2024a] and Aptos [Das et al., 2025]. In GRandLine, the novelty is that a "group-element DKG" (DKG in which the resulting group secret key is a group

element as opposed to a field element) powers the DVRF leveraging $e(sk, H(m))$ as opposed to $H(m)^{sk}$. The benefit of this approach is that (say) Scrape's PVSS can be used more intuitively to realize a publicly verifiable DKG and thus this DRB. Another technique in GRandLine involves a recursive communication protocol to reduce the overall communication complexity. Taking a similar group-element DKG approach, Aptos optimizes its PVSS (combining those by Scrape and Groth [Groth, 2021]) and more importantly unlocks the weighted variants of PVSS, DKG, and DVRF in a practical manner.

3.7 Discussion

3.7.1 Relation to Collective Coin Flipping Protocols

Conceptually, distributed randomness is not a new line of research. Dating to Blum's classic work on coin flipping over the phone [Blum, 1983], distributed randomness has been much researched, albeit in a different context as elaborated below. Namely, Ben-Or and Linial in their seminal work [Ben-Or and Linial, 1985, Ben-Or and Linial, 1989] introduced the *full information* model for the *collective coin flipping* problem, in which n participants with unbounded computational power communicate only via a single broadcast channel to generate a common random bit (such that an honest majority is required [Saks, 1989, Boppana and Narayanan, 2000] and thus assumed). Numerous works exist in this setting, largely classifiable into different types of adversaries dealt with: static [Ben-Or and Linial, 1989, Kahn et al., 1989, Saks, 1989, Ajtai and Linial, 1993, Alon and Naor, 1993, Feige, 1999, Russell et al., 1999, Boppana and Narayanan, 2000], adaptive [Ben-Or and Linial, 1989, Lichtenstein et al., 1989, Dodis, 2000, Goldwasser et al., 2015, Haitner and Karidi-Heller, 2020, Kalai et al., 2021], and variants of adaptive [Cleve and Impagliazzo, 1993, Aspnes, 1998, Goldwasser et al., 2015, Mahloujifar and Mahmoody, 2019, Etesami et al., 2020]. See [Haitner and Karidi-Heller, 2020, Kalai et al., 2021] for this line of research.

Overall, these works concern upper and lower bounds on corruption threshold, bias (deviation from coin flipping probability 1/2), and round complexity, all of which provide interesting theoretical insights. Nonetheless, these bounds are often asymptotic (hence not practical) and are grounded in a more lax definition of security where it is sufficient that bias is bounded (but can still be nontrivial). This is in contrast to the modern literature on DRBs considered in this paper, which aims to design protocols which are as unbiasable as possible, output multiple bits per epoch, have explicit round complexity and fault tolerance, and assume computationally bounded adversaries in a cryptographic setting as well as point-to-point communication channels in the first place.

Outside the full information model (such that cryptography is allowed), the well-known lower bound by Cleve [Cleve, 1986] states that for any r-round coin flipping protocol there exists an efficient adversary controlling half or more of the participants that can bias the output by $\Omega(1/r)$. In other words, it is impossible to have an unbiasable coin flipping protocol with a dishonest majority.

While this may seem to contradict the fault tolerance of delay-based DRBs from Section 3.2, we note that delay functions help circumvent Cleve's impossibility result in the following two ways. First, timed commitments allow recovery of a value that is withheld (either due to honest or Byzantine fault) and lost from an honest node's perspective. In Cleve's proof, the notion of a "default bit" is used in such withholding situation whereas timed commitments effectively deprecate this default bit mechanism, sidestepping the proof logic.

Second, an implicit assumption in Cleve's model is that a Byzantine node is capable of grinding through possibilities to its liking and can arbitrarily choose which messages to output based on inputs from other nodes that are honest. However, VDFs limit this capability such that it is not possible for even a dedicated attacker to grind through possibilities to fix an output of some computation. As a result of above, delay functions can be used to build DRBs that enjoy both the highest fault tolerance and unbiasability without violating any classical lower bounds, and it is

rather surprisingly shown in Bonneau et al. [Bonneau et al., 2025] that the converse holds too, i.e. any fair coin flipping protocol secure against a dishonest majority implies the existence of a delay function. In a similar vein, Bailey et al. [Bailey et al., 2022] shows that VDFs can circumvent some classical impossibility results for general multiparty computation (MPC), of which DRBs are a special case.

3.7.2 Withholding Attacks

In a withholding attack, an adversary can influence the outcome by not publishing some information. Any leader-based protocol is vulnerable due to the inherent reliance on a leader's availability, affecting the protocol's liveness (as well as unbiasability and potentially unpredictability). Any protocol with a private lottery is also fundamentally vulnerable.

1. **Protocols with a leader.** RandHound, RandHerd, and SPURT suffer from the leader unavailability issue in case the leader withholds such that their liveness is affected and a beacon output can be aborted.[6] A fallback is needed if a leader withholds (e.g. HydRand's PVSS recovery).

2. **Protocols with a private lottery.** The issue of withholding is more fundamental with private lottery schemes like Algorand, as there is no accountability. There are two possible remedies. First, we can require all participants to post their lottery outputs every single epoch even if they lose the lottery, in which case any lack of message would be indicative of withholding. However, this incurs communication cost, negating the advantages of a private lottery. Second, SSLE (single secret leader election) [Boneh et al., 2020] can be used to guarantee one winner per epoch, enabling detection of withholding. The guarantee of one winner as opposed to the expectation of one winner is what differentiates SSLE. While this makes withholding obvious, it does not prevent withholding by itself, nor does it detect

[6] In contrast to a leader in SPURT, that in RandHound or RandHerd can abort after seeing the beacon output in plaintext.

who withholds in the case of withholding. Designing efficient SSLE protocols remains an open research area.

3.7.3 Adaptive Security

A DRB is *adaptively secure* if its security properties remain unaffected against an adaptive adversary instead of a static one. Here, we discuss a way to remedy adaptive attacks.

1. **Requiring private lottery winners to broadcast marginal entropy and proof of selection into C_τ in the same message.** While some private lottery schemes may involve less than or equal to t entropy providers per epoch, the fact that one message (per entropy provider) comprises both announcement of winning the lottery and provision of marginal entropy allows adaptive security (e.g. as in Algorand). By the time an adversary knows which nodes to corrupt adaptively in an epoch (after the nodes reveal their identity as entropy providers), there is no extra step left to be corrupted. A similar line of research exists in the MPC literature in a model called YOSO (You-Only-Speak-Once) [Gentry et al., 2021, Liu-Zhang et al., 2025], inspired by Algorand.

On the one hand, it is important for the sake of adaptive security that there is no central point of dependency in any step of a protocol. Otherwise, participating nodes depend on the leader (functioning as either an orchestrator or an entropy provider) such that an adversary can adaptively corrupt such leaders. On the other hand, we note that a proof of adaptive security does not follow immediately from a protocol's lack of leaders and can in fact be tricky to show. That threshold BLS achieves adaptive security is a result (with variations in assumptions) established only in recent years [Bacho and Loss, 2022, Das and Ren, 2024, Libert, 2025], and the same is true for OptRand [Bacho and Loss, 2023].

Table 3.1: DRB Comparison

	Section (dissertation)	Cryptographic Primitive	Fault Tolerance (less than)	Independent Participation	Per-Epoch Entropy Provider	α-Intra-Unpredictability	β-Inter-Unpredictability	Immunity to Withholding	Verifier Complexity	Communication Complexity		Max Damage	Recovery Cost	Post-Quantum
										Optimistic	Worst			
Commit-Reveal	3.1	Commitment	1	✓	All	$O(\Delta)$	1	✗	$O(n)$	$O(n^2)$	$O(n^3)$	Bias	$O(1)$	✓
Unicorn++	3.2	VDF	n	✓	All	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^3)$	None	$O(1)$	✗
Cornucopia (§ 5)		VDF	n	✓	All	$O(\Delta)$	1	✗	$O(1)$	$O(n)$	$O(n^2)$	None	$O(1)$	✗
Ext. Beacon+VDF		VDF	n	✓	External	$O(\Delta)$	1	✓	$O(1)$	$O(n)$	$O(n^2)$	None	$O(1)$	✗
HeadStart		VDF	n	✓	All	$O(\Delta)$	1	✗	$O(L+\log n)^{\#}$	$O(n \log n)$	$O(n^2 \log n)$	None	$O(1)$	✗
RandRunner		Trapdoor VDF	n	✗	None	$T^{\ddagger}$	$t^{\S}$	✓	$O(\log T)^{\ddagger}$	$O(n)$	$O(n^2)$	Predict	$O(n^3)$	✗
Bicorn (§ 4)		Timed commitment	n	✓	All	$T^{\ddagger}$	1	✓	$O(n)$	$O(n^2)$	$O(n^3)$	None	$O(1)$	✗
RANDAO ('16)	3.3	Commitment	n	✓	All	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^2)^{\dagger}$	None	$O(n)^{\dagger}$	✓
EVR		Escrow-DKG	$n/3$	✗	All	$O(\Delta)$	1	✓	$O(n^3)$	$O(n^3)$	$O(n^4)$	None	$O(n)$	✗
Scrape	3.4	PVSS	$n/2$	✗	All	$O(\Delta)$	1	✓	$O(n^2)$	$O(n^3)$	$O(n^4)$	Bias^{r}	$O(n^3)$	✗
Albatross		PVSS	$n/2$	✗	All	$O(\Delta)$	1	✓	$O(1)$	$O(n)$	$O(n^2)$	Bias^{r}	$O(n^3)$	✗
RandShare		(P)VSS	$n/3$	✗	All	$O(\Delta)$	1	✓	$O(n^3)$	$O(n^3)$	$O(n^4)$	Bias^{r}	$O(1)$	✗
SecRand		PVSS	$n/2$	✗	All	$O(\Delta)$	1	✓	$O(n^2)$	$O(n^3)$	$O(n^4)$	Bias^{r}	$O(n^3)$	✗
Rondo		(P)VSS	$n/3$	✗	All	$O(\Delta)$	1	✓	$O(n^2)$	$O(n^2)$	$O(n^2 \log n)$	Bias^{r}	$O(n^4)$	✗
HERB		Thr. ElGamal	$n/3$	✗	All	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^3)$	Bias^{r}	$O(n^4)$	✗
HydRand	3.5	PVSS	$n/3$	✗	Comm*	$O(\Delta)$	t	✓	$O(n)$	$O(n^2)$	$O(n^3)$	Bias	$O(n^3)$	✗
GRandPiper		PVSS	$n/2$	✗	Comm*	$O(\Delta)$	t	✓	$O(n^2)$	$O(n^2)$	$O(n^2)$	Bias	$O(n^3)$	✗
BRandPiper		(P)VSS	$n/2$	✗	Comm*	$O(\Delta)$	1	✓	$O(n^2)$	$O(n^2)$	$O(n^3)$	Bias	$O(n^4)$	✗
Ouroboros		PVSS	$n/2$	✗	Comm	$O(\Delta)$	1	✓	$O(n^2)$	$O(n^3)$	$O(n^3)^{\dagger}$	Bias	$O(n^2)^{\dagger}$	✗
RandHound		PVSS	$n/3$	✗	Comm	$O(\Delta)$	1	✗	$O(cn)$	$O(c^2 n)$	$O(c^2 n^2)$	Bias	$O(n^3)$	✗
HashRand		(P)VSS	$n/3$	✗	Comm	$O(\Delta)$	1	✓	$O(n^2 \log n)$	$O(n^2 \log n)$	$O(n^2 \log n)$	Bias	$O(n^4)$	✓
SPURT		PVSS	$n/3$	✗	Comm	$O(\Delta)$	1	✗	$O(n)$	$O(n^2)$	$O(n^2)$	Bias	$O(n^3)$	✗
OptRand		PVSS	$n/2$	✗	Comm	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^2)$	Bias	$O(n^3)$	✗
Caucus		Hash chain	$n/3$	✓	Comm*	$O(\Delta)$	1	✗	$O(1)$	$O(n)$	$O(n^2)$	Bias	$O(n^3)$	✓
NV++		VRF, thr. ElGamal	$n/3$	✗	Comm	$O(\Delta)$	1	✗	$O(n)$	$O(n)$	$O(n)^{\dagger}$	Bias	$O(n^2)^{\dagger}$	✗
Algorand		VRF	$n/3$	✓	Comm*	$O(\Delta)$	1	✗	$O(1)$	$O(n)$	$O(n)^{\dagger}$	Bias	$O(n^2)^{\dagger}$	✗
Ouroboros Praos		VRF	$n/2$	✓	Comm	$O(\Delta)$	1	✗	$O(n)$	$O(n^2)$	$O(n^2)^{\dagger}$	Bias	$O(n^2)^{\dagger}$	✗
RANDAO ('25)		BLS	$n/3$	✗	Comm	$O(\Delta)$	1	✗	$O(1)$	$O(n)$	$O(n)^{\dagger}$	Bias	$O(n^2)^{\dagger}$	✗
drand	3.6	Thr. BLS	$n/2$	✗	None	$O(\Delta)$	1	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$	✗
RandHerd		Thr. Schnorr	$n/3$	✗	None	$O(\Delta)$	1	✗	$O(1)$	$O(c^2 n)$	$O(n^4)$	Bias	$O(n^4)$	✗
DDH-DRB		DDH-based DVRF	$n/2$	✗	None	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$	✗
GLOW-DRB		Pairing-based DVRF	$n/2$	✗	None	$O(\Delta)$	1	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^4)$	✗
Strobe		RSA, VSS	$n/2$	✗	None	$O(\Delta)$	1	✓	$O(1)$	$O(n^2)$	$O(n^3)$	Predict	$O(n)$	✗
GRandLine		PVSS, pairing	$n/2$	✗	None	$O(\Delta)$	1	✓	$O(n)$	$O(n^2)$	$O(n^3)$	Predict	$O(n^3)$	✗

Comm is short for Committee. c is the size of a shard in RandHerd and RandHound. We assume a leader can be Byzantine for both. Albatross' verifier and communication complexities are per beacon output. In Ouroboros and Ouroboros Praos, we assume the number of slot leaders in an epoch is denoted by n. In RANDAO ('25), it is 32. [r] In a non-rushing adversary model, max damage would be predict rather than bias. [*] Each committee consists of a leader by default or by expectation. [†] PBB (public bulletin board) is assumed.
[‡] T denotes VDF's delay parameter, and verification of Pietrzak's VDF is logarithmic in T. [§] $\beta = t$ for RandRunner's β-inter-unpredictability assuming a dishonest minority without any computational advantage. See [Schindler et al., 2021] for more scenarios.
[#] In HeadStart, L is the number of contribution rounds within an epoch.

3.7.4 Comparison of DRBs

Table 3.1 provides an overall comparison of DRBs. *Fault Tolerance* indicates the minimum number of faulty nodes that can abort a protocol (after the initial setup). Protocols with *Independent Participation* allow a node to contribute to beacon output without the knowledge of other nodes in advance.

Verifier Complexity refers to the computational cost for a passive observer (not participating in the protocol) to verify a beacon output. We exclude the cost associated with the initial setup for both verifier and communication complexities. We assume a verifier complexity of $O(n)$ per Lagrange interpolation or Scrape's PVSS [Cascudo and David, 2017]. *Communication Complexity* concerns bitwise point-to-point communication among nodes by default. Alternatively, we consider a *public bulletin board* (PBB) (e.g. blockchain) as a reliable information exchange medium in protocols where it is intrinsic. In a PBB model, we consider both the bitwise writing cost (amount of data posted to PBB) and reading cost (by all nodes where each node only reads data relevant to it). In the absence of PBB, Byzantine consensus [Castro and Liskov, 1999] incurs $O(n^2)$ cost per decision by default.

Max Damage refers to the maximum damage possible when $n - 1$ rushing [Gennaro et al., 1999] adversarial nodes cooperate. The reason for this column is to observe the consequence of when the honest majority assumption fails, which has happened in practice (e.g. a $625 million Axie Infinity's Ronin hack in 2022 [Scharfman, 2023]). A rushing adversary may delay sending messages until *after* reading messages sent by all honest nodes in a given round of communication. There can exist a separation between what rushing versus non-rushing adversaries can do especially in protocols from Section 3.4: if one can generate its entropy contributions after seeing (or otherwise simultaneously with) all the honest nodes' entropy contributions, then biasing (or otherwise predicting) is possible. The same fundamental reasoning applies to (say) Ouroboros, where slot leaders (i.e. entropy providers) communicate in sequential slots (and hence the ad-

versary can reconstruct an honest node's entropy contribution before generating its own). In escrow-based protocols, we assume the adversaries are rational.

Recovery Cost refers to the communication cost associated with recovering from an adversarial corruption. Regenerating keys (e.g. PVSS.KeyGen or for private lottery schemes) and VDF.Setup incur $O(n^3)$ recovery cost without PBB (and $O(n^2)$ with PBB) while we conservatively assume each DKG incurs $O(n^4)$ recovery cost (although it can be optimized [Gurkan et al., 2021, Abraham et al., 2023, Bacho et al., 2024a, Bacho et al., 2024b, Feng et al., 2024]). *Post-Quantum* indicates whether or not the protocol is post-quantum secure. Finally, we note that it is possible to employ multiple DRBs as subprotocols to a multi-tiered DRB (an approach taken by Mt. Random [Cascudo et al., 2021]) in order to combine and take advantage of various DRB properties at the same time.

3.7.5 Concurrent Work

In addition to our work, two concurrent SoKs (one by Raikwar and Gligoroski [Raikwar and Gligoroski, 2022] and one by Kavousi et al. [Kavousi et al., 2024]) exist and offer different perspectives on distributed randomness. We provide a detailed comparison here and shed light on the motivation of our SoK as well as on the progress of the DRB landscape in general since the publication of the SoKs.

Security definitions. In the SoK by Raikwar and Gligoroski [Raikwar and Gligoroski, 2022], the authors first define the four security properties of a DRB and additionally provide what it means for a DRB to be *secure* (by delineating a different security game altogether). While the two properties liveness and public verifiability are roughly the same as our notions, unbiasability (or bias-resistance in their work) and unpredictability are defined differently. Specifically, bias-resistance is defined by two inequalities—$\Pr[\mathrm{bit}_i(\Omega_\tau) = \mathcal{A}_i(\Omega_1, \ldots, \Omega_{\tau-1})] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$ and $\Pr[\mathrm{bit}_i(\Omega_\tau) = 0] \leq \frac{1}{2} + \mathrm{negl}(\lambda)$—where $\mathrm{bit}_i(\Omega_\tau)$ denotes the i-th bit of Ω_τ. Essentially, this definition of bias-resistance is reflective of bitwise pseudorandomness and thus accounts for attacks

Table 3.2: Committee-Based DRBs

			Step 2: Beacon Output Generation	
			Fresh per-node entropy	$\Omega_{\tau-1}$ & precommitted per-node entropy
Step 1: Committee Selection	Public	RR	**BRandPiper** *Step 1*: Node $i \equiv \tau \pmod{n}$ *Step 2*: Share-aggregate-reconstruct	
		RS	**Ouroboros** *Step 1*: Follow-the-satoshi[1] *Step 2*: Share-reconstruct-aggregate	**HydRand** *Step 1*: Node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ *Step 2*: $\Omega_\tau = H(\Omega_{\tau-1} \parallel h^{e_{\tilde{\tau}}})$ **GRandPiper** *Step 1*: Node $i \equiv \Omega_{\tau-1} \pmod{\tilde{n}}$ *Step 2*: $\Omega_\tau = H(h^{e_{\tilde{\tau}}}, \Omega_{\tau-1}, ..., \Omega_{\tau-t})$
		LS	**RandHound** *Step 1*: Node $\arg\min_i H(C \parallel pk_i)$ *Step 2*: Share-reconstruct-aggregate **SPURT, OptRand** *Step 1*: Node $i \equiv \tau \pmod{n}$ *Step 2*: Share-aggregate-reconstruct	
	Private	VRF	**NV++** *Step 1*: $\mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel nonce) < target$ *Step 2*: Threshold ElGamal	**Algorand** *Step 1*: $\mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel role) < target$ *Step 2*: $\Omega_\tau = \mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel \tau)$ **Ouroboros Praos**[2] *Step 1*: $\mathsf{VRF}_{sk}(\Omega_{\tau-1} \parallel slot \parallel \text{TEST}) < target$ *Step 2*: $\Omega_\tau = H(\Omega_{\tau-1} \parallel epoch \parallel \rho_1 \parallel ... \parallel \rho_K)$
		Hash chain		**Caucus**[3] *Step 1*: $H(h_\tau \oplus \Omega_{\tau-1}) < target$ *Step 2*: $\Omega_\tau = h_\tau \oplus \Omega_{\tau-1}$

Public committee selection mechanisms (Section 3.5.1.1) include RR (round-robin), RS (random selection), and LS (leader-based selection) while details regarding private committee selection can be found in Section 3.5.1.2. For details on beacon output generation, see Sections 3.5.2.1 and 3.5.2.2. Note that HashRand and RANDAO ('25) reside in the same categories as Ouroboros and HydRand, respectively.

[1] This sampling mechanism selects nodes weighted by stake [Bentov et al., 2014, Kiayias et al., 2017].

[2] The protocol is a variant of Algorand. While $|C_\tau|$ is expected to be one in Algorand (with 1 final winner per lottery and 1 lottery per epoch), that in Ouroboros Praos is expected to be K where each epoch consists of K slots and thus K per-slot lotteries. Parameters *slot* and *epoch* denote the slot and epoch numbers, respectively, and $\rho_i = \mathsf{VRF}_{sk_i}(\Omega_{\tau-1} \parallel slot_i \parallel \text{NONCE})$ is returned by the slot leader of $slot_i$. TEST and NONCE are strings.

[3] In Caucus, a VRF is replaced by a hash function combined with a hash chain, i.e. a list $(h_1, ..., h_m)$ with $h_\tau = H(h_{\tau+1})$ for all $\tau = 1, ..., m-1$ where $h_m = s$ for some random seed. A hash chain provides the functionality of provably committing to private inputs as one publicizes one h_τ at a time (h_τ in epoch τ). Each node independently generates a private hash chain. One downside is that the hash chain needs to be periodically regenerated, as m is finite.

like bit-fixing attacks where the attacker is able to fix a bit of the beacon output. However, it does not account for "inter-bit" attacks where (say) the attacker fixes all bits of a beacon output

except the first one to be the value taken by the first bit which is still pseudorandom; in this case, the beacon output is still pseudorandom at a bit level but is biased to be either $0 \cdots 0$ or $1 \cdots 1$ at an output level. Interestingly, their notion of a *secure* DRB given in their Appendix does work with pseudorandomness at an output level, but it does not account for private biasing as in our definition of unbiasability. On a different note, their definition of unpredictability concerns predicting beacon outputs of all future epochs but not the next immediate one (similar to our β-inter-unpredictability with $\beta \geq 1$). We extend the definition to $\beta = 0$ and consider α-intra-unpredictability, which better corresponds to an intuitive prediction attack where the attacker tries to predict the next immediate beacon output and its properties.

Our definitions are motivated by intuitive applications, such as lotteries, a potentially biased beacon that appears pseudorandom to outsiders, and drand (which has already been deployed for years). For the purpose of lotteries (and any other intuitive use of randomness), our definitions consider not only bit-fixing scenarios, but also scenarios (including the above inter-bit attack) where any property of Ω_τ may be manipulated or predicted. This is helpful, as an adversary would likely want to know if (say) the winning lottery number is even or odd even if it cannot compute the value exactly. In light of a potentially biased beacon that appears pseudorandom, our definitions are also helpful, as we make the biasing strategy σ explicit and allow the possibility of private biasing where a secret key may be needed to detect the bias.

In the context of a drand-like beacon, our definitions more tightly analyze the possibility of such a beacon to be predictable (forever) but not biasable (which is the realistic threat of these beacons). The reason is that for a compromised drand-like beacon, $\mathcal{A}$ participating in DRB^τ (as in Figure 3.1) is able to basically distinguish $\Omega_{\tau,1}$ from random at time $T_{\tau,2} - \alpha$ for $\alpha = w(\Delta)$ with the help of $\sigma = sk$ (DKG group secret key); however, unbiasability is still satisfied given that the values of $\Omega_{\tau,1}$ and $\Omega_{\tau,0}$ are exactly equal (assuming $\mathcal{A}$ does produce value for $\Omega_{\tau,1}$) such that the advantage in distinguishing the two cases ($b = 0$ or $b = 1$) is zero when challenged with the exact same value. In general, any compromised beacon that has no marginal entropy is conceptually

going to be predictable but not biasable, and our definitions capture this scenario tightly due to the fact that our unbiasability game is essentially a distinguishing one between when the adversary participates in a protocol run or not, which is subtly different than distinguishing from random.

Top-down and bottom-up. One of the main gaps filled by our SoK is that we take a top-down approach to DRB design as opposed to bottom-up. We consequently illustrate based on concepts like marginal entropy and entropy providers. In the other SoKs, emphasis is placed on the underlying cryptographic primitives and their properties (security, scalability, etc.), complementing our approach. Especially in Kavousi et al. [Kavousi et al., 2024], the theme of public randomness is also thematically complementary, making their discussions of PVSS, VRF, VDF, and blockchain particularly insightful and interesting from the perspective of public randomness.

Research gaps and progress. Here, we highlight several research gaps identified in the SoKs and describe how our proposed solutions address them. The first one is called Research Problem 3 by Raikwar and Gligoroski [Raikwar and Gligoroski, 2022] and concerns the design of a DRB with sub-quadratic communication complexity and optimal fault-tolerance. While Unicorn++ technically achieves sub-quadratic communication complexity in a PBB model (where n participants post n entropy contributions), its communication complexity is at least quadratic in the absence of PBB if a transcript of size $O(n)$ must be agreed upon every epoch. To achieve sub-quadratic communication complexity without PBB, a protocol would thus ideally require an agreeable transcript of size at most $O(1)$; with PBB, this means the amount of data posted to PBB should be of size at most $O(1)$. This is exactly what Cornucopia does; rather than n entropy contributions, a cryptographic accumulator accumulating those n entries is posted to PBB after coordinating with a *coordinator* from which each participant receives a proof of inclusion (membership proof) separately. While concretely instantiating a sub-quadratic DRB with an efficient consensus algorithm realizing (and obviating) a PBB is still an open problem, it is in this way that Cornucopia makes progress towards Research Problem 3.

The second gap we highlight is called Research Problem 4 by Raikwar and Gligoroski and concerns the design of a puzzle-based DRB incurring low computation complexity. Bicorn solves exactly this problem, and it does so by leveraging timed commitments in which the delay computation is in fact avoided in the optimistic case; only in the pessimistic case is the delay computation required. More details can be found in Chapter 4.

Among many technical research gaps portrayed by Kavousi et al. [Kavousi et al., 2024], one of them (Gap 3) concerns the minimization of bias in a protocol driven by leader-based VRF, which has the advantage of avoiding a DKG setup at the cost of potential bias. Progress here is given by Christ et al. [Christ et al., 2024] in the form of accountable secret leader election. In the context of SSLE, the accountability property is particularly useful, as SSLE provides a stronger guarantee of leader uniqueness per epoch compared to VRF-style leader elections. As the work defines accountability and also devises mechanisms to add accountability to existing constructions (both SSLE and VRF-style), it becomes possible to identify withholding leaders in a post hoc fashion. In a carefully implemented rational model, these mechanisms can effectively minimize the bias pointed out by this research gap.

Finally, both SoKs identify optimizing for reconfiguration (or *dynamic* participation) as a key research gap, as it is well-known that performing a new DKG setup whenever a node joins or leaves is expensive and inefficient. The notion of *silent setup* [Garg et al., 2024a,Garg et al., 2024b] has recently been introduced to address this inefficiency; conceptually, the suggested solution is to take the approach of computing a *multi-signature* (e.g. product of individual signatures by participants) which is innately amenable to dynamic participation as opposed to a threshold signature. For instance, a novel type of encryption known as *silent threshold encryption* [Garg et al., 2024b] can be achieved by taking such multi-signature as a witness to a witness encryption. While further optimizations such as making the construction homomorphically aggregatable and reducing the size of the common reference string are still needed for scalability, it is now theoretically possible to construct a DRB with silent setup via this technique. Hence, new possibilities await

beyond previous PVSS-based constructions, for designing DRBs that are capable of gracefully handling dynamic participation.

3.8 Notes for Practitioners

Our systematization highlights important insights both for practitioners and researchers. Based on our comparative framework, we advise practitioners planning to deploy a DRB to consider the following high-level guidelines:

- Delay-based protocols stand above the competition in terms of scalability, flexibility, and robustness, enabling an efficient DRB with unlimited, open participation and security given any honest participant. In theory, VDFs appear to be a silver bullet for DRBs, though they have yet to be widely used in practice and assumptions about VDF security and hardware speeds remain relatively new. They also invoke a unique practical cost in that *somebody* must be able to compute a VDF (preferably by running specialized hardware), which can add latency to the DRB.

- If not using VDFs, practitioners need to think critically about two design dimensions: how large is the set of participants, and how frequently will it change? Given a small, static set of participants, DKG-based protocols, e.g. HERB (from threshold encryption) and drand (from DVRF), scale better than PVSS-based protocols. HERB and drand are both competitive in this setting, differing in randomness quality and max damage.

- For a small but dynamic set of participants, PVSS-based protocols offer better flexibility (by avoiding a costly DKG setup per reconfiguration) and randomness quality. Committees may be needed to scale to more participants.

- Given a large, dynamic set of participants, protocols with private lotteries like Algorand offer better scalability and flexibility simultaneously although the randomness quality is

potentially affected by withholding.

- Finally, escrow-based protocols are suitable against purely financially-motivated adversaries in applications such as lotteries or finance, at the cost of locking up some amount of capital during the protocol.

4 | Bicorn: Tolerating Dishonest Majority with Optimistic Efficiency

4.1 Context

With our observation that the honest majority assumption (network assumption that more than half of the nodes are honest) may be critically violated in practice, for instance in the form of a $625 million Axie Infinity's Ronin hack in 2022 [Scharfman, 2023], we initiate a further study of the dishonest majority setting in the following chapters. In particular, we return to the classic *commit-reveal* [Blum, 1983] which consists of two steps. First, all participants publish a commitment $c_i = \mathsf{Commit}(r_i)$ to a random value r_i (we take this as our entropy contribution e_i from Section 3.1.3 as we hereafter abstract away r_i from Section 3.1.3). Next, participants reveal their r_i values and the result is $\Omega = \mathsf{Combine}(r_1, \ldots, r_n)$ for some suitable combination function (such as exclusive-or or a cryptographic hash). These protocols are simple, efficient, and secure as long as one participant chooses a random r_i value—assuming all participants open their commitments. However, the output can be biased by the last participant to open their commitment via the aforementioned *last-revealer attack*.

Related work. Several approaches exist to avoid last-revealer attacks. Commit-reveal-punish protocols impose a financial penalty on any participant who fails to open their commitment. This penalty can be automatically enforced using modern cryptocurrencies [Andrychowicz et al., 2014,

RANDAO, 2016], but this requires locking up capital and security relies on economic assumptions about the value of manipulation to the attacker.

Other protocols relax the security model of commit-reveal and assume an honest majority of participants. Many constructions enable a majority of participants to recover the input of a malicious minority of participants [Schoenmakers, 1999, Cascudo and David, 2017, Cascudo and David, 2020, Syta et al., 2017, Guo et al., 2020, Schindler et al., 2020, Bhat et al., 2021, Kiayias et al., 2017, Das et al., 2022, Bhat et al., 2023], using cryptographic tools such as publicly verifiable secret sharing (PVSS). Typically, these constructions can tolerate some threshold t of malicious participants failing to complete the protocol, with the trade-off that any coalition of $t + 1$ participants can (secretly) learn the impending output early and potentially bias the protocol, leading to a requirement that $t < \frac{n}{2}$ (honest majority). These protocols are also often quite complex, with communication and computation costs superlinear in n. Another approach is to rely on threshold cryptography for participants to jointly compute a cryptographic function which produces Ω, such as threshold signatures in Dfinity [Camenisch et al., 2022], threshold encryption [Cherniaeva et al., 2019], or threshold inversion in RSA groups [Beaver and So, 1993, Beaver et al., 2023]. The drand DRB [drand, 2020], which uses a chain of threshold BLS signatures, is now deployed publicly with a group of 16 participating nodes producing a new random output every 30 seconds.

A very different approach to constructing DRBs uses time-based cryptography, specifically using *delay functions* to prevent manipulation. The simplest example is Unicorn [Lenstra and Wesolowski, 2015], a one-round protocol in which participants directly publish (within a fixed time window) a random input r_i. The result is computed as $\Omega = \mathsf{Delay}(\mathsf{Combine}(r_1, \ldots, r_n))$. By assumption, a party cannot compute the Delay function before the deadline to publish their contribution r_i and therefore cannot predict Ω or choose r_i in such a way as to influence it. This protocol retains the strong $n - 1$ (just one honest node) security model of commit-reveal, but with no last-revealer attacks. It is also simple and, using modern verifiable delay functions[1]

[1]We consider a VDF-enhanced version of Unicorn (Unicorn++) hereafter.

(VDFs) [Boneh et al., 2018a], the result can be efficiently verified. The downside is that a delay function must be computed for every run of the protocol.

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

- Each of n participants chooses a random value r_i and publishes $c_i = \mathsf{TCom}(r_i)$ using a timed commitment scheme [Boneh and Naor, 2000] TCom before some deadline T_1.
- In the optimistic case, every participant opens their commitment by publishing r_i. The DRB output is $\Omega = \mathsf{Combine}(r_1, \ldots, r_n)$. In this case, the protocol is equivalent to a classic commit-reveal protocol.
- If any participant does not publish their r_i value, it can be recovered by computing $r_i = \mathsf{ForceOpen}(c_i)$, a slow function requiring t steps of sequential work which cannot be evaluated quickly enough for a malicious coalition of participants to learn honest participants' committed values early. The result Ω is the same as in the optimistic case, even if all participants don't reveal their committed values.

This protocol structure was used in a recent proposal by Thyagarajan et al. [Thyagarajan et al., 2021a]. They observe that by using a homomorphic commitment scheme, the commitments can be combined and only a single forced opening is required, instead of opening every withholding participant's commitment separately. Asymptotically, their protocols require linear ($O(n)$) communication and computation costs when run with n participants.

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

simpler and offer much better concrete performance.

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

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

In Section 4.4, we prove security by reducing to the RSW assumption [Rivest et al., 1996] in the algebraic group model (AGM) [Fuchsbauer et al., 2018], except for Bicorn-ZK where we assume a zero-knowledge proof of knowledge of exponent (ZK-PoKE) exists. The Bicorn-RX variant assumes a random oracle. In Section 4.7, we report on concrete implementations of these protocols in Ethereum, showing that our constructions are practical and incur 3–8× increase in per-user cost compared to commit-reveal (but with no manipulation due to aborts) and 5–7× compared to Unicorn (but with no delay function required in the optimistic case).

4.2 Overview

4.2.1 Protocol Outline

We specify all three of our protocol variants in Protocol 1. Our protocols are initialized via a security parameter λ and a delay parameter t, and work over a *group of unknown order*, which we denote $\mathbb{G}$ (see preliminaries in Section 2.5). In addition to the group $\mathbb{G}$, the public

parameters include a pair (g, h), where g is a generator of the group and $h = g^{2^t}$. If desired, a Wesolowski [Wesolowski, 2019] or Pietrzak [Pietrzak, 2018] proof of exponentiation can enable efficient verification that h was computed correctly. Note that this setup only needs to be run once ever (for a specific delay parameter t) and can be used repeatedly (and concurrently) by separate protocol instances; the number of participants does not need to be known and may dynamically change over time.

The common structure of Bicorn protocols is:

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

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

$$c_j = \omega_* \cdot \left(\prod_{i \in [n], i \neq j} c_i \right)^{-1} \tag{4.1}$$

This will cancel out every other participant's contribution and force the output value Ω_*. There are three ways to prevent this attack, each leading to a protocol variant with slightly different properties, which we will present in the following subsections. We present the protocols combined for comparison in Protocol 1, with separate presentations in Section 4.9.

4.2.2 Bicorn-ZK: Using Zero-Knowledge Proofs

The conceptually simplest fix is for each user to publish, along with their commitment c_i, a zero-knowledge proof-of-knowledge $\pi_i = \mathsf{ZK\text{-}PoKE}(g, c_i, \alpha_i)$ of the discrete logarithm of c_i to the base g_i (i.e. α_i). This version (Bicorn-ZK) is specified in Protocol 1 (left). This removes the attack above, as an adversary who computes c_j via Equation 4.1 will not know the discrete log of c_j to the base g. Such proofs can be done in groups of unknown order particularly efficiently in this case. The use of a fixed base g enables the simpler ZKPoKRep protocol of Boneh et al. [Boneh et al., 2019] (possibly in combination with their proof aggregation PoKCR protocol).

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

It might be tempting to optimize the protocol by not verifying each proof π_i in the optimistic case, instead checking directly that $c_i = g^{\tilde{\alpha}_i}$ using the revealed value $\tilde{\alpha}_i$. However, this would introduce a subtle attack: a malicious participant could publish a correctly generated $(c_i, \tilde{\alpha}_i)$ pair but with an invalid proof $\tilde{\pi}_i$. Next, after all other participants have revealed their α values, the

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

Protocol 1: All Bicorn protocol variants: Bicorn-ZK (left column), Bicorn-PC (center column), and Bicorn-RX (right column). Each protocol is presented individually in Section 4.9.

attacker can compute the impending result Ω with their own contribution included, as well as the alternative Ω' if it is removed. They could then choose which output is produced, introducing one bit of bias into the protocol: by publishing $\tilde{\alpha}_i$, they will remain in the protocol (as $\tilde{\pi}_i$ is not checked) and Ω will result, whereas by withholding $\tilde{\alpha}_i$ they will force the pessimistic case, in which they will be removed on account of the faulty $\tilde{\pi}_i$ and Ω' will result. Thus, it is important to verify every participant's proof π_i in both cases to prevent this attack.

4.2.3 Bicorn-PC: Using Precommitment

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

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

4.2.4 Bicorn-RX: Using Pseudorandom Exponents

Finally, we can prevent manipulation by raising each participant's contribution c_i to a unique (small) exponent which depends on all other participants' contributions. Specifically, we define b_* to be the hash of all c_i values: $b_* = H(c_1||c_2||\ldots||c_n)$. We then raise each value c_i to the pseudorandom exponent $b_i = H(c_i \parallel b_*)$. The intuition is that modifying any contribution c_i

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

Table 4.1: A brief comparison of the Bicorn variants. See Protocol 1 for notation ($\langle\mathbb{G}\rangle$ and $\langle\mathcal{B}\rangle$ are the sizes of elements from $\mathbb{G}$ and $\mathcal{B}$, respectively) and Chapter 2 for a background on the RSW assumptions, the algebraic group model (AGM), the random oracle model (ROM), and zero-knowledge proof of knowledge of exponent (ZK-PoKE).

will induce new exponents on each participant's contribution which prevents an adversary from forcing the value $\omega = \prod_{i\in[n]} c_i^{H(c_i\|b_*)}$ to a fixed value. A similar technique was used by Boneh et al. [Boneh et al., 2018c] to prevent rogue-key attacks in BLS multi-signatures. This version (Bicorn-RX) is specified in Protocol 1 (right).

4.2.5 Comparison

Each of these leads to a secure protocol, albeit reducing to slightly different computational assumptions, as we will prove in Section 4.4. All of our protocols reduce to the RSW assumptions with Bicorn-PC and Bicorn-RX requiring the algebraic group model (AGM) for the security reductions and Bicorn-RX also assuming a random oracle. Bicorn-ZK doesn't require the AGM explicitly but instead assumes a secure zero-knowledge proof of knowledge of exponent (ZK-PoKE) for which efficient existing protocols are proven secure only in the AGM [Boneh et al., 2019].

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

4.3 Timed DRBs: Syntax and Security Definitions

We first define a timed DRB using a generalized syntax which captures all of our protocol variants. A timed DRB protocol DRB with time parameter t is the following tuple of algorithms. We describe them below for a run of the protocol with n participants:

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

We require Finalize to be a deterministic algorithm running in time $\mathsf{polylog}(t)$ (the fast optimistic case), and Recover to be a deterministic algorithm running in time $(1+\epsilon)t$ for some small ϵ. We also require the following security properties of a timed DRB (given in pseudocode in Figure 4.1).

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

t-Unpredictability. The t-unpredictability game tasks an adversary with predicting the final

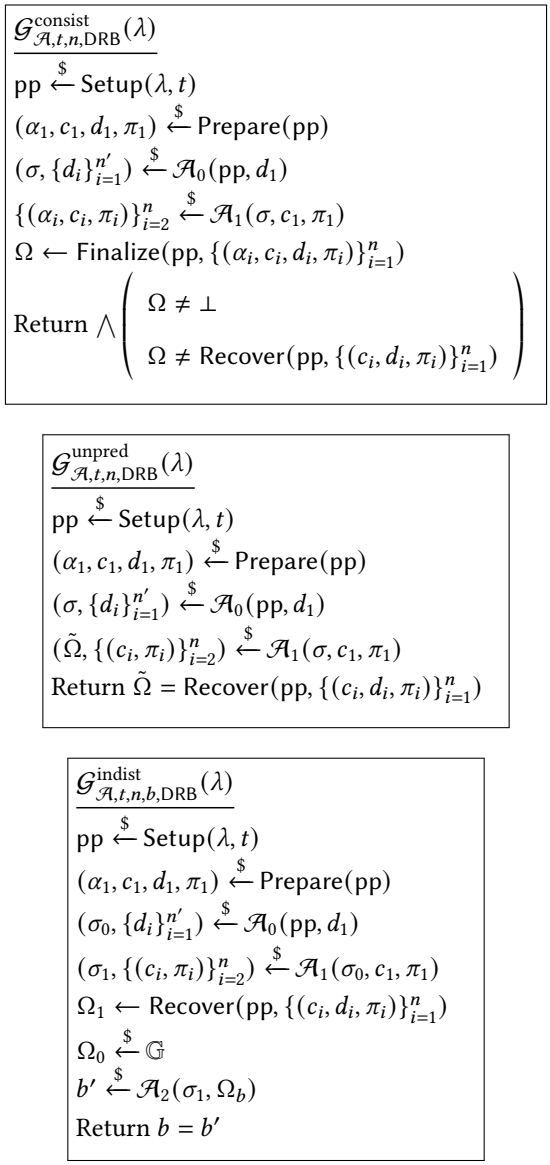

Figure 4.1: Security games for the main security properties: consistency (left), t-unpredictability (center), and t-indistinguishability (right).

output Ωexactly, allowing it control of all but a single honest protocol participant (which publishes first). We define the advantage of an adversary as $\text{Adv}^{\text{unpred}}_{\mathcal{A},t,n,\text{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,n,\text{DRB}}(\lambda) = 1\right]$.

t-Indistinguishability. The t-unpredictability property does not guarantee the output is indistinguishable from random. For that, we provide a stronger t-indistinguishability property in which the adversary must distinguish an honest output from a random output, again allowing

the adversary control of all but one participant. We define the advantage of an adversary as: $\mathsf{Adv}^{\text{indist}}_{\mathcal{A},t,n,\text{DRB}}(\lambda) = \left|\Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,n,1,\text{DRB}}(\lambda) = 1\right] - \Pr\left[\mathcal{G}^{\text{indist}}_{\mathcal{A},t,n,0,\text{DRB}}(\lambda) = 1\right]\right|$. A timed DRB that satisfies t-unpredictability can be transformed generically into one with t-indistinguishability by applying a suitable randomness extractor [Trevisan and Vadhan, 2000, Trevisan, 2001] or hash function (modeled as a random oracle) to the output. A nice feature of our DRBs is that they satisfy t-indistinguishability with respect to the group output space (without applying a randomness extractor) under the suitable decisional RSW assumption.

Discussion. In t-unpredictability and t-indistinguishability, the adversaries $\mathcal{A}_1$ and $\mathcal{A}_2$ are restricted to run in fewer than t sequential steps. This is a slight simplification of the (p, σ)-sequentiality assumption in VDFs [Boneh et al., 2018a], which is suitable for working in the AGM in which parallelism is not helpful in computing group operations.

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

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

4.4 Security of Bicorn-RX

Theorem 4.1 (t-Unpredictability of Bicorn-RX)**.** *Let $\mathcal{A}_{\text{brx}} = (\mathcal{A}_{\text{brx},0}, \mathcal{A}_{\text{brx},1})$ be an algebraic adversary against the t-unpredictability of* BRX *with random exponent space $\mathcal{B} = [2^{2\lambda} \cdot B]$ where hash function H is modeled as a random oracle. Then we construct an adversary $\mathcal{A}_{\text{rsw}} = (\mathcal{A}_{\text{rsw},0}, \mathcal{A}_{\text{rsw},1})$*

such that

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

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

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

More specifically, we bound the advantage of $\mathcal{A}_{\text{brx}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G} = \mathcal{G}^{\text{unpred}}_{\mathcal{A}_{\text{brx}},t,n,\mathsf{BRX}}(\lambda)$ and hybrids $\mathcal{G}_1$,$\mathcal{G}_2$,$\mathcal{G}_3$ for which we justify the following claims leading to the inequality above:

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

$\underline{\mathcal{G} \to \mathcal{G}_1}$. Hybrid $\mathcal{G}_1$ is defined the same as $\mathcal{G}$ except $\mathcal{G}_1$ samples c_1 in Prepare at random from $\mathbb{G}$ instead of through an exponent sampled from $\mathcal{B}$. By Lemma 2.8, the statistical distance between $\mathcal{G}$ and $\mathcal{G}_1$ is at most $1/2^{2\lambda+1}$.

We can view $\mathcal{G}_1$ as computing the beacon output Ω using the representations of $\{c_i\}_{i=2}^n$ provided by the algebraic adversary. Since $\mathcal{A}_{\text{brx}}$ is algebraic, it will provide a representation for

each c_i in terms of elements (c_1, g, h). That is, the adversary outputs $[(e_{i,0}, e_{i,1}, e_{i,2})]_{i=2}^{n}$ such that $c_i = c_1^{e_{i,0}} g^{e_{i,1}} h^{e_{i,2}}$.

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

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

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

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

and we build $\mathcal{A}_{\text{rsw}}$ to win the computational power-of-RSW game by setting c_1 equal to challenge element x and returning this value along with $\langle \mathbf{b}, \mathbf{e} \rangle$. All that is left to show is that $\langle \mathbf{b}, \mathbf{e} \rangle \neq 0$ which we can do through an application of the Schwartz-Zippel lemma modulo a composite [Schwartz, 1980, Zippel, 1979, Bünz and Fisch, 2022]. Define a non-zero polynomial $f(x_1, \ldots, x_n) = x_1 + \sum_{i=2}^{n} x_i e_{i,0}$. Note that $f(\mathbf{b}) = \langle \mathbf{b}, \mathbf{e} \rangle$.

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

the distinguishing advantage between $\mathcal{G}_1$ and $\mathcal{G}_2$.

$\underline{\mathcal{G}_2 \rightarrow \mathcal{G}_3}$**.** After we have that the evaluation point $\mathbf{b}$ does not coincide with precomputed values, we transition to $\mathcal{G}_3$ which is identical to $\mathcal{G}_2$ except it aborts if $f(\mathbf{b}) = 0$. We bound the distinguishing advantage to probability $\frac{n}{2^{2\lambda}} + \prod_{i=1}^{\ell} I_{\frac{1}{p_i}}(r_i, n)$ by applying Schwartz-Zippel modulo a composite [Bünz and Fisch, 2022]. Adversary $\mathcal{A}_{\text{rsw}}$ can simulate $\mathcal{G}_3$ perfectly, simulating the setup and computing $\hat{h}$ with $2t$ work, and wins the RSW game with the same advantage as $\mathcal{G}_3$. ■

Theorem 4.2 (t-Indistinguishability of Bicorn-RX)**.** *Let* $\mathcal{A}_{\text{brx}} = (\mathcal{A}_{\text{brx},0}, \mathcal{A}_{\text{brx},1}, \mathcal{A}_{\text{brx},2})$ *be an adversary against the* t*-indistinguishability of* BRX *with random exponent space* $\mathcal{B} = [2^{2\lambda} \cdot B]$ *where hash function* H *is modeled as a random oracle. Then we construct an adversary* $\mathcal{A}_{\text{rsw}} = (\mathcal{A}_{\text{rsw},0}, \mathcal{A}_{\text{rsw},1})$ *such that*

$$\mathsf{Adv}^{\text{indist}}_{\mathcal{A}_{\text{brx}},t,n,\text{BRX}}(\lambda) \leq \mathsf{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\text{rsw}},t,\text{GGen}}(\lambda) + \frac{2\mathsf{q}_{\text{ro}}^2 + 1}{2^{2\lambda}} + 2(\tilde{p} + \tilde{q} - 1)\left(\frac{n}{|im(H)|} + \frac{n^2}{\tilde{p}\tilde{q}}\right),$$

and where $\text{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ *generates a group of unknown order* $\mathbb{QR}_N^+$ *(whose order is* $\tilde{p}\tilde{q}$ *where* $N = pq$ *for* $p = 2\tilde{p} + 1$*,* $q = 2\tilde{q} + 1$*) used by* BRX*,* q_{ro} *is the number of queries made to the random oracle,* n *is the number of participants, and* $im(H)$ *is the image of* H*. The running time of* $T(\mathcal{A}_{\text{rsw},0}) \approx T(\mathcal{A}_{\text{brx},0}) + 2t$ *and* $T(\mathcal{A}_{\text{rsw},1}) \approx T(\mathcal{A}_{\text{brx},1}) + T(\mathcal{A}_{\text{brx},2})$.

Proof. We bound the advantage of $\mathcal{A}_{\text{brx}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G}^b = \mathcal{G}^{\text{indist}}_{\mathcal{A}_{\text{brx}},t,n,b,\text{BRX}}(\lambda)$ and hybrids $\mathcal{G}_1^b$,$\mathcal{G}_2^b$,$\mathcal{G}_3^b$ for which we justify the following claims leading to the inequality above:

- $\left|\Pr\left[\mathcal{G}^b(\lambda) = 1\right] - \Pr\left[\mathcal{G}_1^b(\lambda) = 1\right]\right| \leq \frac{1}{2^{2\lambda+1}}$
- $\left|\Pr\left[\mathcal{G}_1^b(\lambda) = 1\right] - \Pr\left[\mathcal{G}_2^b(\lambda) = 1\right]\right| \leq \frac{\mathsf{q}_{\text{ro}}^2}{2^{2\lambda}}$
- $\left|\Pr\left[\mathcal{G}_2^b(\lambda) = 1\right] - \Pr\left[\mathcal{G}_3^b(\lambda) = 1\right]\right| \leq (\tilde{p} + \tilde{q} - 1) \cdot \left(\frac{n}{|im(H)|} + \frac{n^2}{\tilde{p}\tilde{q}}\right)$
- $\left|\Pr\left[\mathcal{G}_3^1(\lambda) = 1\right] - \Pr\left[\mathcal{G}_3^0(\lambda) = 1\right]\right| = \mathsf{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\text{rsw}},t,\text{GGen}}(\lambda)$

Hybrid $\mathcal{G}_1^b$ is defined the same as $\mathcal{G}^b$ except $\mathcal{G}_1^b$ samples c_1 in Prepare at random from $\mathbb{G}$

instead of through an exponent sampled from $\mathcal{B}$. By Lemma 2.8, the statistical distance between $\mathcal{G}^b$ and $\mathcal{G}_1^b$ is at most $1/2^{2\lambda+1}$.

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

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

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

We consider a transition in which $c_1^{2^t}$ is replaced with a random group element following the decisional RSW game. We will want to show that the distinguishing advantage for this transition is equal to the advantage of $\mathcal{A}_{\text{rsw}}$. To do this, we will first need to set up the transition with a few more hybrids.

First, as in the unpredictability game, we transition through $\mathcal{G}_2^b$ to disallow collisions in the random oracle in the way specified in the proof of Theorem 4.1. Next, we further define $\mathcal{G}_3^b$ that is the same as $\mathcal{G}_2^b$ but aborts if $g^{\langle \mathbf{b}, \mathbf{e} \rangle}$ is not a generator of $\mathbb{QR}_N^+$. This happens if $\gcd(\langle \mathbf{b}, \mathbf{e} \rangle, |\mathbb{QR}_N^+|) \neq 1$, a condition that is equivalent (in modulo $|\mathbb{QR}_N^+| = \tilde{p}\tilde{q}$) to $f(\mathbf{b}) = \langle \mathbf{b}, \mathbf{e} \rangle = k$ for $k \in \{0, \tilde{p}, \tilde{q}, \ldots, \tilde{p}\tilde{q} - \tilde{p}, \tilde{p}\tilde{q} - \tilde{q}\}$. As we can apply Schwartz-Zippel modulo a composite [Bünz and Fisch, 2022, Remark 3] to each k and there are $\tilde{p} + \tilde{q} - 1$ such k's, we apply the union bound to bound the probability

that $g^{\langle \mathbf{b},\mathbf{e} \rangle}$ is not a generator of $\mathbb{QR}_N^+$ to $(\tilde{p}+\tilde{q}-1)\cdot(\frac{n}{|im(H)|}+\frac{n^2}{\tilde{p}\tilde{q}})$, where $|im(H)|$ denotes the size of the image of the random oracle. While a large $|im(H)|$ may be required for this theoretical bound, a remark here is that one may opt for stronger assumptions, such as the short exponent indistinguishability (SEI) assumption [Couteau et al., 2021] due to the fact that H outputs occur only in the exponents, for efficiency gains in practice.

Now, we are set up to construct $\mathcal{A}_{\text{rsw}}$ by bounding the distinguishing advantage between $\mathcal{G}_3^1$ and $\mathcal{G}_3^0$ where $\mathcal{G}_3^1$ computes the challenge Ω using the process above, while $\mathcal{G}_3^0$ computes the challenge Ω by replacing $c_1^{2^t}$ with a random group element. Note that in the case of $\mathcal{G}_3^1$, Ω is computed to match the output of Recover. On the other hand, in the case of $\mathcal{G}_3^0$, Ω is in fact a random group element. This can be seen as follows. The random group element that replaces $c_1^{2^t}$ can be written as g^r for some $r \xleftarrow{\$} [1, |\mathbb{G}|]$, and so we have:

$$\Omega = (g^r)^{\langle \mathbf{b},\mathbf{e} \rangle} \cdot h^{m_1} \cdot \hat{h}^{m_2} = (g^{\langle \mathbf{b},\mathbf{e} \rangle})^r \cdot h^{m_1} \cdot \hat{h}^{m_2}$$

Since $g^{\langle \mathbf{b},\mathbf{e} \rangle}$ is a generator by the previous hybrid transition, we have that $(g^{\langle \mathbf{b},\mathbf{e} \rangle})^r$ is a random group element, and so Ω is a random group element. Thus, $\mathcal{A}_{\text{rsw}}$ perfectly simulates $\mathcal{G}_3^b$ based on the challenge bit, simulating the setup and computing $\hat{h}$ with $2t$ work, and wins the RSW game with the same advantage as the distinguishing advantage between $\mathcal{G}_3^1$ and $\mathcal{G}_3^0$. ■

4.5 Security of Bicorn-ZK

Theorem 4.3 (t-Unpredictability of Bicorn-ZK)**.** *Let $\mathcal{A}_{\text{bzk}} = (\mathcal{A}_{\text{bzk},0}, \mathcal{A}_{\text{bzk},1})$ be an adversary against the t-unpredictability of* BZK *with random exponent space $\mathcal{B} = [2^{2\lambda} \cdot B]$. Then we construct adversaries $\mathcal{A}_{\text{rsw}} = (\mathcal{A}_{\text{rsw},0}, \mathcal{A}_{\text{rsw},1})$, $\mathcal{A}_{\text{zk}}$, and $\mathcal{A}_{\text{sound}}$ such that*

$$\mathsf{Adv}^{\text{unpred}}_{\mathcal{A}_{\text{bzk}},t,n,\mathsf{BZK}}(\lambda) \leq \mathsf{Adv}^{\text{C-RSW}}_{\mathcal{A}_{\text{rsw}},t,\mathsf{GGen}}(\lambda) + \mathsf{Adv}^{\text{zk}}_{\mathcal{A}_{\text{zk}},\mathsf{ZK\text{-}PoKE}}(\lambda) + \mathsf{Adv}^{\text{sound}}_{\mathcal{A}_{\text{sound}},\mathsf{ZK\text{-}PoKE},\mathsf{S},\mathsf{Ext}}(\lambda) + \frac{1}{2^{2\lambda+1}},$$

and where $\mathsf{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ *generates the group of unknown order used by* BZK*, and* S *and* Ext *are the simulator and extractor for* ZK-PoKE*. The running time of $T(\mathcal{A}_{\text{rsw},0}) \approx T(\mathcal{A}_{\text{bzk},0}) + t$ and*

$T(\mathcal{A}_{\text{rsw},1}) \approx T(\mathcal{A}_{\text{bzk},1})$.

Proof. We bound the advantage of $\mathcal{A}_{\text{bzk}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G} = \mathcal{G}^{\text{unpred}}_{\mathcal{A}_{\text{bzk}},t,n,\text{BZK}}(\lambda)$ and hybrids $\mathcal{G}_1$,$\mathcal{G}_2$,$\mathcal{G}_3$ for which we justify the following claims leading to the inequality above:

- $|\Pr[\mathcal{G}(\lambda) = 1] - \Pr[\mathcal{G}_1(\lambda) = 1]| \leq \text{Adv}^{\text{zk}}_{\mathcal{A}_{\text{zk}},\text{ZK-PoKE}}(\lambda)$
- $|\Pr[\mathcal{G}_1(\lambda) = 1] - \Pr[\mathcal{G}_2(\lambda) = 1]| \leq \frac{1}{2^{2\lambda+1}}$
- $|\Pr[\mathcal{G}_2(\lambda) = 1] - \Pr[\mathcal{G}_3(\lambda) = 1]| \leq \text{Adv}^{\text{sound}}_{\mathcal{A}_{\text{sound}},\text{ZK-PoKE,S,Ext}}(\lambda)$
- $\Pr[\mathcal{G}_3(\lambda) = 1] = \text{Adv}^{\text{C-RSW}}_{\mathcal{A}_{\text{rsw}},t,\text{GGen}}(\lambda)$

Hybrid $\mathcal{G}_1$ is defined the same as $\mathcal{G}$ except $\mathcal{G}_1$ simulates the zero-knowledge proof π_1 in Prepare. The distinguishing advantage is directly bounded by the zero-knowledge property of ZK-PoKE.

Hybrid $\mathcal{G}_2$ is defined the same as $\mathcal{G}_1$ except $\mathcal{G}_2$ samples c_1 in Prepare at random from $\mathbb{G}$ instead of through an exponent sampled from $\mathcal{B}$. By Lemma 2.8, the statistical distance between $\mathcal{G}_1$ and $\mathcal{G}_2$ is at most $1/2^{2\lambda+1}$.

Hybrid $\mathcal{G}_3$ extracts the discrete log $\{\alpha_i\}_{i=2}^n$ from the adversary-provided $\{\pi_i\}_{i=2}^n$ using the extractor from the knowledge soundness property of ZK-PoKE. We bound the probability of any extraction failure using an adversary against the simulation-extractability soundness of ZK-PoKE.

Game $\mathcal{G}_3$ outputs 1 when $\tilde{\Omega} = \Omega = (c_1)^{2^t} \cdot \prod_{i=2}^n h^{\alpha_i}$. We build an adversary $\mathcal{A}_{\text{rsw}}$ that wins the computational RSW game with the same advantage as $\mathcal{G}_3$ by replacing c_1 with challenge x and outputting $\tilde{y} = \frac{\tilde{\Omega}}{\prod_{i=2}^n h^{\alpha_i}}$. ∎

Theorem 4.4 (t-Indistinguishability of Bicorn-ZK)**.** *Let $\mathcal{A}_{\text{bzk}} = (\mathcal{A}_{\text{bzk},0}, \mathcal{A}_{\text{bzk},1}, \mathcal{A}_{\text{bzk},2})$ be an adversary against the t-indistinguishability of* BZK *with random exponent space $\mathcal{B} = [2^{2\lambda} \cdot B]$. Then we construct adversaries $\mathcal{A}_{\text{rsw}} = (\mathcal{A}_{\text{rsw},0}, \mathcal{A}_{\text{rsw},1})$, $\mathcal{A}_{\text{zk}}$, and $\mathcal{A}_{\text{sound}}$ such that*

$$\text{Adv}^{\text{indist}}_{\mathcal{A}_{\text{bzk}},t,n,\text{BZK}}(\lambda) \leq \text{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\text{rsw}},t,\text{GGen}}(\lambda) + 2 \cdot \text{Adv}^{\text{zk}}_{\mathcal{A}_{\text{zk}},\text{ZK-PoKE}}(\lambda) + 2 \cdot \text{Adv}^{\text{sound}}_{\mathcal{A}_{\text{sound}},\text{ZK-PoKE,S,Ext}}(\lambda) + \frac{1}{2^{2\lambda}},$$

and where $\mathsf{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ *generates the group of unknown order used by* BZK*, and* S *and* Ext *are the simulator and extractor for* ZK-PoKE*. The running time of* $T(\mathcal{A}_{\text{rsw},0}) \approx T(\mathcal{A}_{\text{bzk},0}) + t$ *and* $T(\mathcal{A}_{\text{rsw},1}) \approx T(\mathcal{A}_{\text{bzk},1}) + T(\mathcal{A}_{\text{bzk},2})$.

Proof. We bound the advantage of $\mathcal{A}_{\text{bzk}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G}^b = \mathcal{G}^{\text{indist}}_{\mathcal{A}_{\text{bzk}},t,n,b,\mathsf{BZK}}(\lambda)$ and hybrids $\mathcal{G}^b_1, \mathcal{G}^b_2, \mathcal{G}^b_3$ for which we justify the following claims leading to the inequality above:

- $\left|\Pr\left[\mathcal{G}^b(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_1(\lambda) = 1\right]\right| \leq \mathsf{Adv}^{\text{zk}}_{\mathcal{A}_{\text{zk}},\text{ZK-PoKE}}(\lambda)$
- $\left|\Pr\left[\mathcal{G}^b_1(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_2(\lambda) = 1\right]\right| \leq \frac{1}{2^{2\lambda+1}}$
- $\left|\Pr\left[\mathcal{G}^b_2(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_3(\lambda) = 1\right]\right| \leq \mathsf{Adv}^{\text{sound}}_{\mathcal{A}_{\text{sound}},\text{ZK-PoKE},\mathsf{S},\mathsf{Ext}}(\lambda)$
- $\left|\Pr\left[\mathcal{G}^1_3(\lambda) = 1\right] - \Pr\left[\mathcal{G}^0_3(\lambda) = 1\right]\right| = \mathsf{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\text{rsw}},t,\mathsf{GGen}}(\lambda)$

Hybrids $\mathcal{G}^b_1$, $\mathcal{G}^b_2$ and $\mathcal{G}^b_3$ are defined as in the unpredictability proof for Bicorn-ZK, simulating π_1, sampling a random c_1, and extracting $\{\alpha_i\}_{i=2}^n$, respectively.

In $\mathcal{G}^1_3$, the challenge output is computed to match the output of Recover as $\Omega = (c_1)^{2^t} \cdot \prod_{i=2}^n h^{\alpha_i}$. In $\mathcal{G}^0_3$, the challenge output is computed in the same way but by replacing $(c_1)^{2^t}$ with a random group element resulting in Ω to be a random group element. Thus, $\mathcal{A}_{\text{rsw}}$ perfectly simulates $\mathcal{G}^b_3$ based on the challenge bit (by setting c_1 equal to challenge input x and replacing $c_1^{2^t}$ with challenge input y) and wins the RSW game with the same advantage as the distinguishing advantage between $\mathcal{G}^1_3$ and $\mathcal{G}^0_3$. ∎

4.6 Security of Bicorn-PC

Theorem 4.5 (t-Unpredictability of Bicorn-PC)**.** *Let* $\mathcal{A}_{\text{bpc}} = (\mathcal{A}_{\text{bpc},0}, \mathcal{A}_{\text{bpc},1})$ *be an adversary against the* t*-unpredictability of* BPC *with random exponent space* $\mathcal{B} = [2^{2\lambda} \cdot B]$ *where hash function*

H is modeled as a random oracle. Then we construct an adversary $\mathcal{A}_{\text{rsw}} = (\mathcal{A}_{\text{rsw},0}, \mathcal{A}_{\text{rsw},1})$ such that

$$\mathsf{Adv}^{\text{unpred}}_{\mathcal{A}_{\text{bpc}},t,n,\text{BPC}}(\lambda) \leq \mathsf{Adv}^{\text{C-RSW}}_{\mathcal{A}_{\text{rsw}},t,\text{GGen}}(\lambda) + \frac{4n \cdot \mathsf{q}_{\text{ro}} + 1}{2^{2\lambda+1}},$$

and where $\text{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ *generates the group of unknown order used by* BPC*, n is the number of participants, and* q_{ro} *is the number of queries made to the random oracle. The running time of* $T(\mathcal{A}_{\text{rsw},0}) \approx T(\mathcal{A}_{\text{bpc},0}) + 2t$ *and* $T(\mathcal{A}_{\text{rsw},1}) \approx T(\mathcal{A}_{\text{bpc},1})$.

Proof. We bound the advantage of $\mathcal{A}_{\text{bpc}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G} = \mathcal{G}^{\text{unpred}}_{\mathcal{A}_{\text{bpc}},t,n,\text{BPC}}(\lambda)$ and hybrids $\mathcal{G}_1$,$\mathcal{G}_2$,$\mathcal{G}_3$ for which we justify the following claims leading to the inequality above:

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

Hybrid $\mathcal{G}_1$ is defined the same as $\mathcal{G}$ except $\mathcal{G}_1$ samples c_1 in Prepare at random from $\mathbb{G}$ instead of through an exponent sampled from $\mathcal{B}$. By Lemma 2.8, the statistical distance between $\mathcal{G}$ and $\mathcal{G}_1$ is at most $1/2^{2\lambda+1}$.

Hybrid $\mathcal{G}_2$ is defined the same as $\mathcal{G}_1$ except we disallow collisions in the random oracle used for precommitments, i.e., we use sampling without replacement instead of sampling from $[2^{2\lambda}]$. We can apply a standard birthday analysis to bound the probability of collision among the n queries made to $n \cdot \mathsf{q}_{\text{ro}}/2^{2\lambda}$, which bounds the distinguishing advantage between $\mathcal{G}_1$ and $\mathcal{G}_2$.

We can view $\mathcal{G}_2$ as computing the beacon output Ω using the representations of $\{c_i\}_{i=2}^n$ provided by the algebraic adversary. Since $\mathcal{A}_{\text{bpc}}$ is algebraic, it will provide a representation for each c_i not in terms of (c_1, g, h), but in terms of (g, h). The reason, which is important to note, is that the adversary needs to precommit before being given c_1 in Bicorn-PC. Accordingly, we check if c_i was queried to the random oracle by $\mathcal{A}_{\text{bpc},0}$ for each $\{c_i\}_{i=2}^n$. Since we disallow collisions in

the random oracle in a prior game hop, there is only one possible c_i that maps to each d_i. If the random oracle was not queried on c_i then we do not have a representation for c_i in (g, h). The contribution c_i will affect the output if the sampling of a new value for $H(c_i)$ matches d_i provided by the adversary. If it does, this is a "bad" case, and we can bound the probability of this occurring for all n by $n \cdot \mathsf{q}_{\mathrm{ro}}/2^{2\lambda}$. We transition to $\mathcal{G}_3$ where this bad case does not occur.

Then we have that the adversary outputs $[(e_{i,1}, e_{i,2})]_{i=2}^{n}$ such that $c_i = g^{e_{i,1}} h^{e_{i,2}}$. Using this, and given a value $\hat{h} = h^{2^t}$, we can compute Ω as follows:

$$\begin{aligned}
\Omega &= \left(\prod_{i=1}^{n} c_i\right)^{2^t} = \left(c_1 \cdot \prod_{i=2}^{n} g^{e_{i,1}} h^{e_{i,2}}\right)^{2^t} \\
&= \left(c_1 \cdot g^{\sum_{i=2}^{n} e_{i,1}} \cdot h^{\sum_{i=2}^{n} e_{i,2}}\right)^{2^t} \\
&\text{By letting } m_1 = \sum_{i=2}^{n} e_{i,1} \text{ and } m_2 = \sum_{i=2}^{n} e_{i,2}, \\
&= (c_1 \cdot g^{m_1} \cdot h^{m_2})^{2^t} = (c_1^{2^t}) \cdot h^{m_1} \cdot \hat{h}^{m_2}
\end{aligned}$$

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

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

and we build $\mathcal{A}_{\mathrm{rsw}}$ to win the computational RSW game by setting c_1 equal to challenge element x and returning this value. The simulation is perfect, with $2t$ work to perform setup and compute $\hat{h}$, and thus the advantage of $\mathcal{A}_{\mathrm{rsw}}$ matches the advantage of $\mathcal{G}_3$. ■

Theorem 4.6 (t-Indistinguishability of Bicorn-PC)**.** *Let $\mathcal{A}_{\mathrm{bpc}} = (\mathcal{A}_{\mathrm{bpc},0}, \mathcal{A}_{\mathrm{bpc},1}, \mathcal{A}_{\mathrm{bpc},2})$ be an adversary against the t-indistinguishability of* BPC *with random exponent space $\mathcal{B} = [2^{2\lambda} \cdot B]$ where hash function H is modeled as a random oracle. Then we construct an adversary $\mathcal{A}_{\mathrm{rsw}} = (\mathcal{A}_{\mathrm{rsw},0}, \mathcal{A}_{\mathrm{rsw},1})$ such that*

$$\mathsf{Adv}^{\mathrm{indist}}_{\mathcal{A}_{\mathrm{bpc}},t,n,\mathsf{BPC}}(\lambda) \leq \mathsf{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\mathrm{rsw}},t,\mathsf{GGen}}(\lambda) + \frac{4n \cdot \mathsf{q}_{\mathrm{ro}} + 1}{2^{2\lambda}},$$

and where $\mathsf{GGen} \xrightarrow{\$} (\mathbb{G}, g, A, B)$ *generates the group of unknown order used by* BPC*, n is the number*

of participants, and $\mathrm{q_{ro}}$ *is the number of queries made to the random oracle. The running time of* $T(\mathcal{A}_{\mathrm{rsw},0}) \approx T(\mathcal{A}_{\mathrm{bpc},0}) + 2t$ *and* $T(\mathcal{A}_{\mathrm{rsw},1}) \approx T(\mathcal{A}_{\mathrm{bpc},1}) + T(\mathcal{A}_{\mathrm{bpc},2})$.

Proof. We bound the advantage of $\mathcal{A}_{\mathrm{bpc}}$ by bounding the advantage of a series of game hops, using the fundamental lemma of game playing and its identical-until-bad argument [Bellare and Rogaway, 2006]. We define $\mathcal{G}^b = \mathcal{G}^{\mathrm{indist}}_{\mathcal{A}_{\mathrm{bpc}},t,n,b,\mathrm{BPC}}(\lambda)$ and hybrids $\mathcal{G}^b_1$,$\mathcal{G}^b_2$,$\mathcal{G}^b_3$ for which we justify the following claims leading to the inequality above:

- $\left|\Pr\left[\mathcal{G}^b(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_1(\lambda) = 1\right]\right| \leq \frac{1}{2^{2\lambda+1}}$
- $\left|\Pr\left[\mathcal{G}^b_1(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_2(\lambda) = 1\right]\right| \leq \frac{n \cdot \mathrm{q_{ro}}}{2^{2\lambda}}$
- $\left|\Pr\left[\mathcal{G}^b_2(\lambda) = 1\right] - \Pr\left[\mathcal{G}^b_3(\lambda) = 1\right]\right| \leq \frac{n \cdot \mathrm{q_{ro}}}{2^{2\lambda}}$
- $\left|\Pr\left[\mathcal{G}^1_3(\lambda) = 1\right] - \Pr\left[\mathcal{G}^0_3(\lambda) = 1\right]\right| = \mathrm{Adv}^{\text{D-RSW}}_{\mathcal{A}_{\mathrm{rsw}},t,\mathrm{GGen}}(\lambda)$

Hybrids $\mathcal{G}^b_1$, $\mathcal{G}^b_2$ and $\mathcal{G}^b_3$ are defined as in the unpredictability proof for Bicorn-PC, sampling a random c_1, disallowing random oracle collisions, and disallowing precommitments that do not provide a representation in (g, h), respectively. In $\mathcal{G}^1_3$, the challenge output is computed to match Recover as $\Omega = (c_1)^{2^t} \cdot h^{m_1} \cdot \hat{h}^{m_2}$. In $\mathcal{G}^0_3$, the challenge output is computed in the same way but by replacing $(c_1)^{2^t}$ with a random group element resulting in Ω to be a random group element. Thus, $\mathcal{A}_{\mathrm{rsw}}$ perfectly simulates $\mathcal{G}^b_3$ based on the challenge bit (by setting c_1 equal to challenge input x and replacing $c_1^{2^t}$ with challenge input y) and wins the RSW game with the same advantage as the distinguishing advantage between $\mathcal{G}^1_3$ and $\mathcal{G}^0_3$. ∎

4.7 Implementation

We implemented all three variants of Bicorn in Solidity and measured the associated gas costs in Ethereum [Wood et al., 2014]. Our results are presented in Table 4.2. We instantiate $\mathbb{G}$ as an RSA group with a 2048-bit modulus (specifically, it is the quadratic residue subgroup $\mathbb{QR}^+_N$ [Pietrzak, 2018]). Multiplying two group elements costs ~90,000 gas and raising a group element to a power

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

Table 4.2: Ethereum gas costs and main operations involved for each Bicorn variant as well as Unicorn [Lenstra and Wesolowski, 2015] and Commit-Reveal DRBs. For Bicorn-PC, the Commit cost is split to show Precommit and Commit costs. The operations are: $\text{store}_{\mathbb{G}/2\lambda}$, storing a group element or 2λ-bit value; mul, multiplication of two group elements; exp, raising a group element to a power of size 2λ bits; poe.v and zk-poke.v, verifying a proof of exponentiation and proof of knowledge of exponent, respectively. Concrete costs are given with $\mathbb{G} = \mathbb{QR}_N^+$ within an RSA-2048 group and $\lambda = 128$.

of size 32 bytes costs ~150,000 gas. As mentioned in Section 2.6, we use the short exponent indistinguishability (SEI) assumption [Couteau et al., 2021] to reduce the size of the exponent required in practice from 288 to 32 bytes. The largest costs for each protocol are verifying a proof of exponentiation (PoE) for the VDF computation in the pessimistic Recover case and verifying a zero-knowledge proof of knowledge of exponent needed for each commitment in Bicorn-ZK. We implemented both proofs using non-interactive variants of Wesolowski proofs (ZKPoKRep from [Boneh et al., 2019] for the latter), which requires a prime challenge to be sampled. Verifying this "hash-to-prime" operation costs between 2.3–4 million gas, depending on the size of the Pocklington certificate used to test the primality of a number onchain. Table 4.2 reports costs with the smallest possible certificate. A recent work [Kemmoe and Lysyanskaya, 2024] shows it is possible to replace "hash-to-prime" with hashing to large odd integers, which may be an optimization.

Comparison to other DRBs. *Per-user Costs:* We find that the user operations for Bicorn-RX are practical on Ethereum with them costing 3× for Commit and 7× for Reveal when compared

to the standard Commit-Reveal and Unicorn protocols. In total, the sum of these operations per user per run comes to under 600,000 gas.

Pessimistic Costs: In the pessimistic case, a single call to Recover is required in all versions of Bicorn, costing millions of gas. This pessimistic case is roughly equivalent to *every* run of Unicorn. As the number of users grows large and the chances of Bicorn's optimistic case occurring decrease though, at some point it may make more sense to switch to Unicorn and avoid the overheads of Commit and Reveal that Bicorn protocols incur.

4.8 Discussion

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

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

Incentives and punishment. While all Bicorn variants ensure malicious participants cannot manipulate the output, they can waste resources by forcing the protocol into the more-expensive recovery mode. The protocol provides accountability as to which nodes published an incorrect

α_i value or other minor deviations which lead to removal (i.e. publishing an incorrect c_i such that $H(c_i) \neq d_i$ in Bicorn-PC or publishing an incorrect π_i in Bicorn-ZK). If signatures are added to each message, efficient fraud proofs are possible. In a blockchain setting, financial penalties can be used to punish incorrect behavior.

Batch verification optimization. In the optimistic case, the n exponentiations required to verify that $c_i = g^{\tilde{\alpha}_i}$ for each participant can be streamlined via batch verification [Bünz et al., 2018, Bellare et al., 1998]. The general idea is that $g^x = 1 \wedge g^y = 1$ can be verified more efficiently by checking $g^{r \cdot x + y} = 1$ for a random $r \xleftarrow{\$} \mathcal{R}$, as the latter equation implies the former with high probability given a large enough $\mathcal{R}$. In our case, to verify that $c_1 = g^{\tilde{\alpha}_1} \wedge c_2 = g^{\tilde{\alpha}_2} \wedge \ldots \wedge c_n = g^{\tilde{\alpha}_n}$, we generate random values $r_i \xleftarrow{\$} \mathcal{R}$ and verify that $g^{\sum r_i \cdot \tilde{\alpha}_i} = \prod c_i^{r_i}$. Thus, instead of computing n exponentiations each with an exponent of size $|\mathcal{B}|$, verification requires only one exponentiation with an exponent of size $n|\mathcal{B}||\mathcal{R}|$ and one n-way multi-exponentiation [Pippenger, 1980].

Lowering costs with rollup proofs. Practical costs can become significant if all users must post data to the blockchain to participate. An alternative solution is to perform Bicorn mediated via a *rollup server* (Rollup-Bicorn) which gathers every participant's c_i value and publishes:

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

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

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

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

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

To achieve the best of both worlds (the efficiency of rollup servers as well as robustness against censorship), we might design a *delegated* Bicorn protocol. In a delegated protocol, users can choose between multiple rollup servers or directly participate as an untrusted (possibly singleton) rollup server. This works like delegated proof-of-stake protocols: participants can delegate for efficiency if they want or participate individually if no server is considered trustworthy. This is straightforward for Bicorn-PC and Bicorn-ZK, as each rollup server can simply compute a partial product c_* which are multiplied together to obtain the final output Ω. Such a protocol for Bicorn-RX would require additional rounds of exponent randomization, to ensure each user's exponent is randomized by contributions from users at other rollup servers.

Bicorn Setup

Setup
input: λ, t
output: group $\mathbb{G}$, generators $g, h \in \mathbb{G}$, proof π_h, range $[A, B]$

1. Run $(\mathbb{G}, g, A, B) \xleftarrow{\$} \mathsf{GGen}(\lambda)$ to generate a group of unknown order
2. Compute $h \leftarrow g^{2^t}$, optionally with $\pi_h = \mathsf{PoE}(g, h, 2^t)$

Protocol 2: Bicorn setup routine (common to all protocol variants), where PoE is a proof of exponentiation [Boneh et al., 2019].

4.9 Individual protocol presentations

For reference, we present each protocol variant separately.

Bicorn-ZK

. . . . . *deadline* T_0

Commit
Each participant i runs:

1. Sample $\alpha_i \xleftarrow{\$} \mathcal{B}$
2. Compute $c_i \leftarrow g^{\alpha_i}$
3. Compute $\pi_i \leftarrow \mathsf{ZK\text{-}PoKE}(g, c_i, \alpha_i)$
4. Publish c_i, π_i

. . . . . *deadline* T_1

Reveal
Each participant i runs:

1. Publish α_i

Finalize
input: $c_i, \pi_i, \tilde{\alpha}_i$ for $i \in [1, n]$
output: Ω

1. For all users i, verify π_i using c_i
   (a) If verification fails for any π_i, remove participant i
2. Verify that $c_i = g^{\tilde{\alpha}_i}$ for all $i \in [1, n]$
   (a) If so, output $\Omega = \prod_{i \in [n]} h^{\tilde{\alpha}_i}$ // optimistic case
3. Output $\Omega = \left(\prod_{i \in [n]} c_i \right)^{2^t}$ // pessimistic case
   (a) Optionally, a proof π_Ω can be output to enable efficient verification of Ω

Protocol 3: Bicorn protocol with zero-knowledge proofs of knowledge of exponent (ZK-PoKE)

Bicorn-PC

Precommit
Each participant i runs:

1. Sample $\alpha_i \xleftarrow{\$} \mathcal{B}$
2. Compute $c_i \leftarrow g^{\alpha_i}$
3. Publish $d_i = H(c_i)$

. . . . . *deadline* T_0

Commit
Each participant i runs:

1. Publish c_i

. . . . . *deadline* T_1

Reveal
Each participant i runs:

1. Publish α_i

Finalize
input: $d_i, \tilde{c}_i, \tilde{\alpha}_i$ for $i \in [1, n]$
output: Ω

1. Verify that $d_i = H(\tilde{c}_i)$ for all $i \in [1, n]$
   (a) If any $d_i \neq H(\tilde{c}_i)$ or $\tilde{c}_i$ was not published by T_1, remove participant i
2. Verify that $\tilde{c}_i = g^{\tilde{\alpha}_i}$ for all $i \in [1, n]$
   (a) If so, output $\Omega = \prod_{i \in [n]} h^{\tilde{\alpha}_i}$ // optimistic case
3. Output $\Omega = \left(\prod_{i \in [n]} \tilde{c}_i \right)^{2^t}$ // pessimistic case
   (a) Optionally, a proof π_Ω can be output to enable efficient verification of Ω

Protocol 4: Bicorn protocol with precommitment round

Bicorn-RX

. . . . . *deadline* T_0

Commit
Each participant i runs:

1. Sample $\alpha_i \xleftarrow{\$} \mathcal{B}$
2. Compute $c_i \leftarrow g^{\alpha_i}$
3. Publish c_i

. . . . . *deadline* T_1

Reveal
Each participant i runs:

1. Publish α_i

Finalize
input: $c_i, \tilde{\alpha}_i$ for $i \in [1, n]$
output: Ω

1. Compute $b_* = H(c_1||c_2||\ldots||c_n)$
2. Verify that $c_i = g^{\tilde{\alpha}_i}$ for all $i \in [1, n]$
   (a) If so, output $\Omega = \prod_{i\in[n]} \left(h^{H(c_i\|b_*)}\right)^{\tilde{\alpha}_i}$ // optimistic case
3. Output $\Omega = \left(\prod_{i\in[n]} c_i^{H(c_i\|b_*)}\right)^{2^t}$ // pessimistic case
   (a) Optionally, a proof π_Ω can be output to enable efficient verification of Ω

Protocol 5: Bicorn protocol with randomized exponents using a random oracle H

5 | Cornucopia: Tolerating Dishonest Majority in Large-Scale Networks

5.1 Context

While Bicorn successfully introduces an optimistic case to the Unicorn protocol in a concretely efficient manner, it is realistically possible that, especially in large-scale networks, the pessimistic case will be triggered significantly more than the optimistic case. The reason is that Bicorn still inherits the downside of commit-reveal, which is that even one faulty node (honest or malicious) can interfere with the flow of the protocol. As a result, Bicorn is a protocol that shines when used in a "reliable" network where nodes are expected to be online most of the time and failures are expected to occur occasionally but not too frequently. If the pessimistic case of Bicorn is to be triggered every single epoch, it might as well be more practical to run Unicorn, with the beacon output computed as $\Omega = \mathsf{Delay}(\mathsf{Combine}(r_1, \ldots, r_n))$. In that case, the downside of Unicorn which motivates Cornucopia 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 (e.g. Merkle tree) to publish a succinct commitment to all users' contributions, retaining the security advantages of Unicorn in the strong $n - 1$ (just one honest node) security model 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 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 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 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 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. The coordinator does face at least linear

costs to compute the accumulator and per-user proofs, but for certain accumulators [Srinivasan et al., 2022a, Wang et al., 2023], the coordinator can efficiently batch compute all users' witnesses.

Related work. Unicorn [Lenstra and Wesolowski, 2015] introduces delay-based DRBs. Several extensions to Unicorn (of which Bicorn is one) work in a similar model. For instance, RandRunner [Schindler et al., 2021] 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 [Lee et al., 2022] 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 5.2).
- We prove (in Section 5.3) 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 5.4) 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 5.5). Since Cor-

nucopia 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 5.6 with discussion about some protocol extensions and open problems.

5.2 Timed DRBs: Definitions and Constructions

We first define timed DRBs using a generalized syntax.[1]

Definition 5.1 (Timed DRBs)**.** A timed DRB protocol consists of the following algorithms:

$\mathsf{Setup}(\lambda, t) \xrightarrow{\$} \mathsf{pp}$: 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.

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

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

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

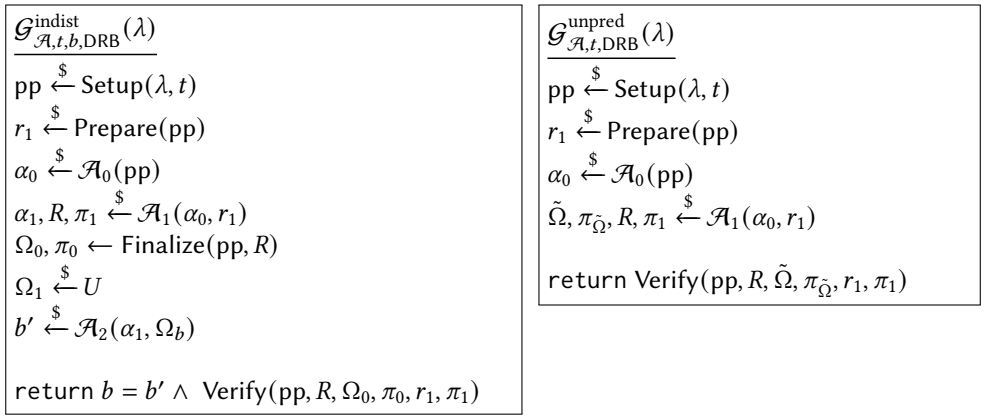

Figure 5.1: Security games for (p, σ)-indistinguishability and (p, σ)-unpredictability.

$\text{Verify}(\text{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \to \{0, 1\}$: 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 5.1):

Definition 5.2 ((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 $\text{poly}(\lambda, t)$. 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, exactly like the adversary for VDF sequentiality (Theorem 2.2). The adversary's advantage is: $\text{Adv}^{\text{unpred}}_{\mathcal{A},t,\text{DRB}}(\lambda) = \Pr\left[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\text{DRB}}(\lambda) = 1\right]$.

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 a DRB output from a random output, again allowing the adversary control of all but one participant.

Definition 5.3 ((p, σ)-indistinguishability)**.** The (p, σ)-indistinguishability game is exactly like the (p, σ)-unpredictability game, except with an extra input bit b. The challenger provides the adversary the genuine output of Finalize if $b = 0$ and a random output if $b = 1$. The adversary must, after running for at most $\sigma(t)$ steps on at most $p(t)$ parallel processors, output a guess b' for which output it received. We define the adversary's advantage as:

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

As observed by Boneh et al. [Boneh et al., 2018a], there is a generic transformation in the random oracle model in which a timed DRB which satisfies (p, σ)-unpredictability can be transformed generically into one with (p, σ)-indistinguishability by applying the random oracle to the output.

5.2.1 Unicorn

As a warm-up, we describe Unicorn [Lenstra and Wesolowski, 2015] succinctly as a timed DRB in our framework in Figure 5.2. Note that the the original Unicorn proposal used the delay function Sloth, which computes modular square roots modulo a prime. We describe Unicorn here using a modern VDF instead [Boneh et al., 2018a].

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 [Lenstra and Wesolowski, 2015]. We note that its security is also implied by our security proof for Cornucopia in Theorem 5.7, 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

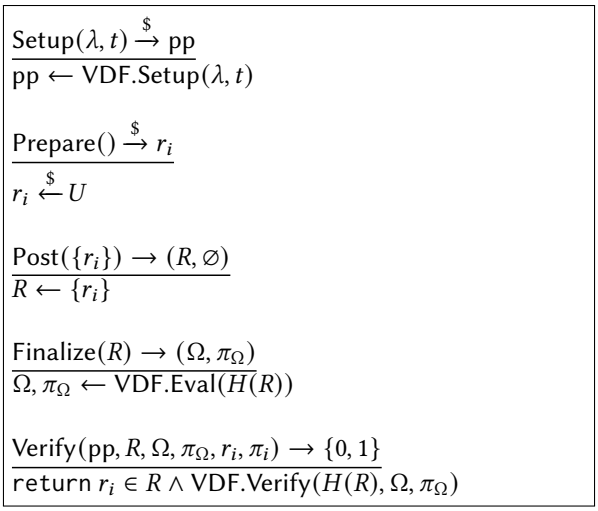

Figure 5.2: The Unicorn timed DRB protocol [Lenstra and Wesolowski, 2015]

achieve the same security as Unicorn while storing only $\Theta(1)$ data on the public bulletin board.

5.2.2 Cornucopia

Cornucopia, shown in Figure 5.3, 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 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.

5.3 Cornucopia Security

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

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

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

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

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

$\underline{\mathsf{Verify}(\mathsf{pp}, R, \Omega, \pi_\Omega, r_i, \pi_i) \rightarrow \{0, 1\}}$
$\mathtt{return}\ \mathsf{VDF.Verify}(H(R), \Omega, \pi_\Omega) \wedge \mathsf{Acc.MemVer}(R, r_i, \pi_i)$

Figure 5.3: The Cornucopia protocol

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

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

Figure 5.4: Insertion security game

Definition 5.4 (Insertion Security)**.** An accumulator is *insertion-secure* if for any PPT algorithm $\mathcal{A}$, the probability of $\mathcal{A}$ winning the insertion security game (Figure 5.4) is negligible:

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

To win the insertion security game (Figure 5.4), the adversary must produce an accumulator value A such that it can supply a membership proof for a randomly chosen element with non-negligible probability. Note that the adversary is not limited to producing A via the normal Accumulate function; it can produce A using any procedure at all. We will prove this property holds for concrete accumulators in Section 5.4, 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 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.

Lemma 5.5. *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* $\Pr[\mathcal{E}_1] \leq \text{negl}(\lambda)$.

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

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

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 $\frac{1}{\lambda^c}$, the probability that $\mathsf{MemVer}(R, x, w_1) = 1$ and $\mathcal{A}_0$ queried R to the random oracle is greater than or equal to $\frac{1}{\lambda^c}$. 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 $\mathcal{A}$, $\Pr[\mathsf{MemVer}(R, x, w_1) = 1 \wedge A = R] \geq \frac{1}{\lambda^c} \cdot \frac{1}{q(\lambda)}$ which is non-negligible. Thus, with non-negligible probability, $\mathcal{G}^{\text{insert}}_{\mathcal{B},\text{Acc}}(\lambda) = 1$. ■

Lemma 5.6. *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 verifiable delay function satisfying (p, σ)-sequentiality, then* $\Pr[\mathcal{E}_2] \leq \text{negl}(\lambda)$.

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

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

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 VDF.pp 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 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[2] 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

[2]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.

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. Recall that if $\mathcal{A}$ wins, 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, $\mathcal{A}_1$ must have output $\tilde{\Omega} = \mathsf{VDF.Eval}(H(R))$.

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

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

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

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.

■

Theorem 5.7 (Unpredictability of Cornucopia)**.** *Cornucopia is (p, σ)-unpredictable when instantiated with an insertion-secure accumulator, a verifiable delay function satisfying (p, σ)-sequentiality,*

and a hash function modeled as a random oracle.

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

Observe that $\Pr[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\text{CC}}(\lambda) = 1] = \Pr[\mathcal{E}_1] + \Pr[\mathcal{E}_2]$. By Theorem 5.5, $\Pr[\mathcal{E}_1] \leq \text{negl}(\lambda)$. By Theorem 5.6, $\Pr[\mathcal{E}_2] \leq \text{negl}(\lambda)$. Therefore, $\Pr[\mathcal{G}^{\text{unpred}}_{\mathcal{A},t,\text{CC}}(\lambda) = 1] \leq \text{negl}(\lambda)$. ■

Corollary 5.8. *Cornucopia is (p, σ)-indistinguishable when a random oracle is applied to its output.*

5.4 Insertion-secure accumulators

We now turn to the question of instantiating accumulators satisfying insertion security (Theorem 5.4).

5.4.1 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. 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 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. Bilinear accumulators have the same shortcut, which we remove with the same modification.

A second example, potentially of practical interest, is a *range accumulator*. A range accumulator can be defined from any accumulator scheme and for any data universe with a known total

ordering (for example, any fixed subset of the integers such as $\{0,1\}^k$). With a range accumulator, the value $H(x,y)$ can be accumulated, which is interpreted as adding a range $[x,y]$ (the value $H(x,x)$ can be accumulated to add a single element x). Given any value z, proving membership can be achieved by providing a witness $w' = (w,x,y)$ where $w = \mathsf{Acc.GetMemWit}(S,A,H(x,y))$ for $x \leqslant z \leqslant y$. This concept is quite natural and efficient, though it is also trivially not insertion-secure: an adversary can win $\mathcal{G}^{\text{insert}}_{\mathcal{A},\text{Acc}}(\lambda)$ 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.[3]

5.4.2 Merkle trees

Lemma 5.9. *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 $O : \{0,1\}^{2n} \to \{0,1\}^n$. Let A be the accumulator output by an adversary $\mathcal{A}$ in $\mathcal{G}^{\text{insert}}_{\mathcal{A},\text{Acc}}(\lambda)$. We show that for a uniform $x \in \{0,1\}^n$, the adversary can provide a verifying witness $w = (w_1,\ldots,w_k)$ for x with only negligible probability. For a verifying witness, it must hold that $O(w_k||\ldots O(w_2||O(w_1||x))) = A$. We'll show that with overwhelming probability (over choice of x), no query to O involved in the witness verification was made by the adversary in step 2 of $\mathcal{G}^{\text{insert}}_{\mathcal{A},\text{Acc}}(\lambda)$.

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, $b_i = w_i||O(w_{i-1}||\ldots)$. Let $p(\lambda)$ be a polynomial upper bound on the total number of queries made by the adversary to the random oracle throughout the game. Observe first that $\Pr[b_1 = a_j \text{ for some j}] = \frac{\ell}{2^\lambda}$ since $b_1 = w_1||x$ and x is chosen at random. Assume

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

that the probability that b_i is equal to any a_j is at most $\frac{i\ell \cdot p(\lambda)}{2^\lambda}$. If this event does not occur, then $O(b_{i+1}) = O(w_{i+1}||O(b_i))$ is a freshly random value, and the probability that $b_{i+1} = a_j$ for any j is at most $\frac{\ell \cdot p(\lambda)}{2^\lambda}$ (since $\mathcal{A}$ can try up to $p(\lambda)$ values for w_{i+1}).

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

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

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

5.4.3 RSA accumulators

In a standard RSA accumulator [Camenisch and Lysyanskaya, 2002, Lipmaa, 2012], $\mathsf{Setup}(\lambda)$ generates a random group of unknown order and a generator g for this group using some group generation algorithm 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 $B = g^b$. 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 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.

Definition 5.10 (Adaptive Root Assumption [Boneh et al., 2018b])**.**

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

Lemma 5.11. *Suppose a standard RSA accumulator is modified; the algorithm* 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* GenGroup*.*

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

Recall that if $\mathcal{A}$ wins $\mathcal{G}^{\text{insert}}_{\mathcal{A},\text{Acc}}(\lambda)$, it means that $\mathsf{MemVer}(A, x, w_x) = 1$. For RSA accumulators, $\mathsf{MemVer}(A, x, w_x) = 1$ 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.

■

Corollary 5.12. *The modified RSA accumulator is insertion-secure in the algebraic group model (AGM), since the Adaptive Root Assumption holds in the AGM [Feist, 2022].*

5.4.4 Bilinear accumulators

We show that bilinear accumulators [Nguyen, 2005, Srinivasan et al., 2022b] 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 [Nguyen, 2005], and we follow [Papamanthou, 2011] in its presentation. Let $\mathbb{G}, \mathcal{G}$ be cyclic multiplicative groups of prime order p, and let $e : \mathbb{G} \times \mathbb{G} \to \mathcal{G}$ be a bilinear pairing. Let $s \xleftarrow{\$} \mathbb{Z}_p^*$, and let g be a generator of $\mathbb{G}$. Let $\text{srs} = [g, g^s, \ldots, g^{s^q}]$ be the structured reference string, where q is an (polynomial in λ) upper bound on the number of accumulated elements. The public parameters are $(p, \mathbb{G}, \mathcal{G}, e, g, \text{srs})$. 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, the adversary is constrained to perform only algebraic operations within the given group. That is, the adversary is given some group elements as input, and for any element that it outputs, it must provide a description of the operations used to obtain that element. In our setting, the algebraic adversary is given as input $[1, g, g^s, \ldots, g^{s^q}]$. For any group element h that the adversary outputs, it must provide a scalar vector $v \in \mathbb{Z}_p^*$ such that $h = \prod_{i=0}^{q} g^{v_i \cdot s^i}$. We refer the reader to [Fuchsbauer et al., 2018, Gabizon et al., 2019] 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.

Definition 5.13 (q-Discrete Logarithm Assumption (q-DLOG) [Fuchsbauer et al., 2018])**.** The q-DLOG assumption holds in a group $\mathbb{G}$ if for every p.p.t. adversary $\mathcal{A}$,

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

Lemma 5.14. *The bilinear accumulator modified so that the $A = 1$ case is disallowed is insertion-secure in the algebraic group model, under the q-DLOG Assumption.*

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

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

5.4.5 From generic universal accumulators

We show how to construct an insertion-secure accumulator Acc′ from any universal accumulator Acc. The core idea is to map each element x to two pseudorandom sets (S_x^+, S_x^-), each a subset of the data universe U. Proving membership of x for Acc′ 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 in S_x^+ is wrongly excluded with probability $\frac{1}{2}$ and each item in S_x^- is wrongly included with probability $\frac{1}{2}$. By setting ensuring the sizes of S_x^+, S_x^-, we can amplify security to ensure such an adversary has only a negligible probability of correctly showing inclusion of a random element.

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

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

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

GetMemWit(S′, A, x)**:** Outputs a vector of witnesses $\mathbf{w}$ of length λ where:

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

MemVer$(A, x, \mathbf{w})$**:** Outputs 1 if and only if the following holds for all $i \in [\lambda]$:

- For $i \leq \frac{\lambda}{2}$, $\mathsf{Acc.MemVer}(A, H(i,x), w_i) = 1$.
- For $i > \frac{\lambda}{2}$, $\mathsf{Acc.NonMemVer}(A, H(i,x), w_i) = 1$.

Lemma 5.15. *If* Acc *is a secure universal accumulator and H is modeled as a random oracle,* Acc′ *is insertion-secure.*

Proof. Suppose for the sake of contradiction that Acc′ is not insertion-secure, and let $\mathcal{A}$ be an adversary that wins the insertion game with probability at least $\frac{1}{\lambda^c}$ for some constant $c > 0$,

conditioned on the event that it does not query x before it outputs A. (Since $\mathcal{A}$ is polynomially-bounded, this event fails to occur with only negligible probability). Thus, treating H as a random oracle, $H(x)$ is a λ-length tuple of truly random independent values $y_i \in U$, where $y_1, \ldots, y_{\frac{\lambda}{2}}$ should be included, and $y_{\frac{\lambda}{2}+1}, \ldots, y_\lambda$ should be excluded.

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

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

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

We now construct an adversary $\mathcal{B}$ that breaks the universal accumulator 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 $\mathsf{Acc.MemVer}(A, y_i, (w_1)_i) = 1$ and $\mathsf{Acc.NonMemVer}(A, y_i, (w_2)_i) = 1$. 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 the universal accumulator security.

■

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

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

5.4.6 From vector commitments

Vector commitments (VCs) [Catalano and Fiore, 2013] can be used to construct an insertion-secure accumulator for sets of bounded size $\leqslant k$ for any k polynomial in λ. Let the message space $\mathcal{M}$ underlying our VC have size exponential in λ, and assume there is some total ordering over $\mathcal{M}$. To accumulate a set $S \subseteq \mathcal{M}$, we order this set to obtain a vector and commit to this vector. The witness for an element $x \in S$ is an index $i \leq k$ and a VC opening proof for that index. To verify this witness, one verifies the opening proof. This scheme is detailed below:

$\mathsf{Setup}(\lambda)$: Output $\mathsf{pp} \leftarrow \mathsf{VC.Setup}(\lambda, k)$.

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

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

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

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

Theorem 5.16. *When constructed with a vector commitment over an exponentially large data universe, this accumulator scheme is insertion-secure.*

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

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

for some constants $c_1, c_2 > 0$.

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

Scheme	Trusted setup	$\|R\|$ (bytes)	Witness size (asymp.)	Witness size (bytes)	$\|\text{pp}\|$ (asymp.)	Witness gen. time (asymp.)
Merkle tree	no	32	$O(\log n)$	$32 \cdot \lceil \log n \rceil$	$O(1)$	$O(n \log n)$
RSA accumulator	yes†	384	$O(1)$	384	$O(1)$	$O(n^2)$
Bilinear accumulator	yes	48	$O(1)$	48	$O(n)$	$O(n \log n)$
Hyperproofs	yes	48	$O(\log n)$	$48 \cdot \lceil \log n \rceil$	$O(n)$	$O(n \log n)$

Table 5.1: Comparison of accumulator options for Cornucopia, at a security level of $\lambda = 128$ bits. Witness generation time is the time required to compute all n witnesses. †RSA accumulators can be instantiated using class groups [Long, 2018], which do not require trusted setup. We report numbers here for the classic RSA group $\mathbb{Z}_N^*$.

we have found a vector commitment A and proofs π_1, π_2 that the same component takes on two distinct values, contradicting position binding of the vector commitment. ■

5.5 Efficiency comparison of accumulator constructions

Cornucopia can be constructed from any insertion-secure accumulator. In Table 5.1 we compare efficiency trade-offs between Merkle trees, RSA accumulators, bilinear accumulators, and a construction from a vector commitment called Hyperproofs [Srinivasan et al., 2022a]. 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. 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 a layer-1 blockchain such as Ethereum, where every 32-byte word stored onchain costs around 20,000 gas [Wood et al., 2014]. 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 [Chen et al., 2021]. Deployments may also use class groups of imaginary quadratic order [Buchmann and Hamdy, 2011, Long, 2018], which avoid 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 [Pietrzak, 2018, Wesolowski, 2019], 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 [Kerber et al., 2021, Nikolaenko et al., 2024]. 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 [FileCoin, 2020]. Ethereum generated a smaller string with 2^{12} powers of tau in a community setup [Foundation, 2023]. 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 [Srinivasan et al., 2022a] 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 [Srinivasan et al., 2022a]. Verifying witnesses takes on the order of milliseconds. This efficiency is immediately inherited by the accumulator constructed using our approach in Section 5.4.6. 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.

5.6 Concluding Discussion

We introduce 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. Discussing the efficiency of common accumulator constructions in Section 5.5, 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 serves as motivation to revisit accumulator constructions with this goal in mind.

We discuss possible extensions to the Cornucopia framework, leaving a complete analysis to

future work.

Public verifiability. As proposed, Cornucopia only offers meaningful security guarantees to active participants (as opposed to passive observers) that contribute randomness to the protocol.

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 that commit to participating in the protocol regularly. Each notarized participant, after verifying the inclusion proof showing that its contribution is indeed 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 using a signature scheme such as BLS that supports succinct multi-signatures [Boneh et al., 2018c], resulting in only $O(1)$ additional overhead.

Any observer can then verify with the set of notarized participants that contribute 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. Using BLS multi-signatures, this would be about as efficient to verify in practice as a state-of-the-art honest-majority protocol like drand [drand, 2020], while offering much stronger security.

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 cannot 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 $\Omega = \mathsf{Delay}(\mathsf{Combine}(R_1, \ldots, R_n))$, passing the concatenation of these commitments to the VDF. Note that this idea can be extended

to a limit where every user might in fact be their own coordinator, in which case the protocol is exactly Unicorn. This makes it easy to see, informally, that extra malicious coordinators cannot undermine the 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 Ω 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.

Post-quantum security. As the current building blocks such as RSA-based VDFs, RSA accumulator, and bilinear accumulator do not achieve post-quantum security, Cornucopia may be vulnerable to quantum attacks unless explicitly instantiated with building blocks (VDF and accumulator) that are post-quantum secure. Basing cryptographic hardness on post-quantum assumptions is an active area of research in general. For VDFs, the only post-quantum construction that is implemented for practical parameters is a lattice-based one [Osadnik et al., 2025], but the concrete numbers for VDF proof size and prover time still need improvement. Others exist theoretically (e.g. isogeny-based [Chavez-Saab et al., 2021]). For accumulators, Merkle trees seem to suffice for the purpose of Cornucopia while lattice-based constructions exist in other settings [Papamanthou et al., 2013, Yu et al., 2018, Kemmoe et al., 2025].

Incentives. Finally, 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 nontrivial 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 using 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 use 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.

6 | Conclusion

We conclude by identifying the following areas which we consider most promising for further research:

- While VDFs are a promising tool, practical deployment requires good estimates of the lower bound of wall-clock VDF evaluation time. More research is needed to gain confidence in the security of underlying VDF primitives (such as repeated modular squaring), and hardware implementations must be built to provide practical assurance.
- VDFs might be useful as a modular layer in strengthening other DRBs in a "belt-and-suspenders" approach.
- With the exception of VDF-based protocols like Unicorn++, all other DRBs assume a permissioned setting requiring some initial setup (e.g. PKI or DKG) to establish participants' identities. It is an open question to extend non-VDF-based protocols to enable ad hoc, permissionless participation (so that the overhead of reconfiguration or dynamic participation is minimal).
- Most existing DRBs assume synchronous communication, which may fail in practice. Extending protocols to handle asynchrony is an important challenge.
- Most papers today use game-based security definitions. Universal Composability (UC) security proofs [Canetti, 2001] could be a useful tool for proving more robust and modular security results.

- The post-quantum story of DRBs is yet to unfold. We observe in Table 3.1 that most current DRB solutions are not post-quantum secure. While progress is expected at the level of cryptographic building blocks such as VDF, VRF, PVSS, and DKG, there may be many practical gaps to fill toward post-quantum DRBs over time alongside incremental deployment of post-quantum cryptography in general.

- Moreover, leveraging quantum algorithms directly for the sake of (distributed) randomness is a different line of work that can be supplementary while likewise assuming a landscape of quantum computers.

- Finally, there is a gap between the systems-based literature on DRBs and the traditional cryptographic literature on randomness extractors [Trevisan and Vadhan, 2000, Trevisan, 2001], with DRBs simply assuming cryptographic primitives such as hash functions work as extractors in practice. Utilizing the existing theory of extractors could prove useful in scenarios where high-quality DRB outputs are required directly.

Bibliography

[Abraham et al., 2022] Abraham, I., Ben-David, N., and Yandamuri, S. (2022). Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In *ACM PODC*.

[Abraham et al., 2023] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., and Stern, G. (2023). Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation. In *Annual International Cryptology Conference*.

[Adida, 2008] Adida, B. (2008). Helios: Web-based open-audit voting. In *USENIX security symposium*.

[Ajtai and Linial, 1993] Ajtai, M. and Linial, N. (1993). The influence of large coalitions. *Combinatorica*, 13(2).

[Al-Bassam et al., 2017] Al-Bassam, M., Sonnino, A., Bano, S., Hrycyszyn, D., and Danezis, G. (2017). Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*.

[Alon and Naor, 1993] Alon, N. and Naor, M. (1993). Coin-flipping games immune against linear-sized coalitions. *SIAM Journal on Computing*, 22(2).

[Andrychowicz et al., 2014] Andrychowicz, M., Dziembowski, S., Malinowski, D., and Mazurek, L. (2014). Secure Multiparty Computations on Bitcoin. In *IEEE Security & Privacy*.

[Arun et al., 2022] Arun, A., Bonneau, J., and Clark, J. (2022). Short-lived zero-knowledge proofs and signatures. In *Asiacrypt*.

[Arun and Setty, 2024] Arun, A. and Setty, S. (2024). Nebula: Efficient read-write memory and switchboard circuits for folding schemes. *Cryptology ePrint Archive*.

[Aspnes, 1998] Aspnes, J. (1998). Lower bounds for distributed coin-flipping and randomized consensus. *Journal of the ACM (JACM)*, 45(3).

[Azouvi et al., 2018] Azouvi, S., McCorry, P., and Meiklejohn, S. (2018). Winning the caucus race: Continuous leader election via public randomness. *arXiv preprint arXiv:1801.07965*.

[Bacho et al., 2024a] Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., and Papachristoudis, D. (2024a). Grandline: adaptively secure dkg and randomness beacon with (log-) quadratic communication complexity. In *ACM CCS*.

[Bacho and Loss, 2022] Bacho, R. and Loss, J. (2022). On the adaptive security of the threshold BLS signature scheme. In *ACM CCS*.

[Bacho and Loss, 2023] Bacho, R. and Loss, J. (2023). Adaptively secure (aggregatable) pvss and application to distributed randomness beacons. In *ACM CCS*.

[Bacho et al., 2024b] Bacho, R., Loss, J., Stern, G., and Wagner, B. (2024b). Harts: High-threshold, adaptively secure, and robust threshold schnorr signatures. In *Asiacrypt*.

[Baigneres et al., 2015] Baigneres, T., Delerablée, C., Finiasz, M., Goubin, L., Lepoint, T., and Rivain, M. (2015). Trap me if you can-million dollar curve. *IACR Cryptol. ePrint Arch.*

[Bailey et al., 2022] Bailey, B., Miller, A., and Sattath, O. (2022). General partially fair multi-party computation with vdfs. *Cryptology ePrint Archive*.

[Bandarupalli et al., 2024] Bandarupalli, A., Bhat, A., Bagchi, S., Kate, A., and Reiter, M. K. (2024). Random beacons in monte carlo: Efficient asynchronous random beacon without threshold cryptography. In *ACM CCS*.

[Beaver et al., 2023] Beaver, D., Chalkias, K., Kelkar, M., Kokoris-Kogias, L., Lewi, K., de Naurois, L., Nikolaenko, V., Roy, A., and Sonnino, A. (2023). Strobe: Streaming threshold random beacons. In *Advances in Financial Technologies*.

[Beaver and So, 1993] Beaver, D. and So, N. (1993). Global, unpredictable bit generation without broadcast. In *Eurocrypt*.

[Bellare et al., 1998] Bellare, M., Garay, J. A., and Rabin, T. (1998). Fast batch verification for modular exponentiation and digital signatures. In *Eurocrypt*.

[Bellare and Rogaway, 1993] Bellare, M. and Rogaway, P. (1993). Random oracles are practical: A paradigm for designing efficient protocols. In *ACM CCS*.

[Bellare and Rogaway, 2006] Bellare, M. and Rogaway, P. (2006). The security of triple encryption and a framework for code-based game-playing proofs. In *Eurocrypt*.

[Ben-Or and Linial, 1985] Ben-Or, M. and Linial, N. (1985). Collective coin flipping, robust voting schemes and minima of banzhaf values. In *FOCS*.

[Ben-Or and Linial, 1989] Ben-Or, M. and Linial, N. (1989). Collective coin flipping. *Advances in Computing Research*.

[Benaloh and De Mare, 1993] Benaloh, J. and De Mare, M. (1993). One-way accumulators: A decentralized alternative to digital signatures. In *Eurocrypt*.

[Bentov et al., 2016] Bentov, I., Gabizon, A., and Zuckerman, D. (2016). Bitcoin beacon. *arXiv preprint arXiv:1605.04559*.

[Bentov and Kumaresan, 2014] Bentov, I. and Kumaresan, R. (2014). How to use bitcoin to design fair protocols. In *Annual International Cryptology Conference*.

[Bentov et al., 2014] Bentov, I., Lee, C., Mizrahi, A., and Rosenfeld, M. (2014). Proof of activity: Extending bitcoin's proof of work via proof of stake. *ACM SIGMETRICS Performance Evaluation Review*.

[Bhat et al., 2023] Bhat, A., Shrestha, N., Kate, A., and Nayak, K. (2023). OptRand: Optimistically responsive distributed random beacons.

[Bhat et al., 2021] Bhat, A., Shrestha, N., Luo, Z., Kate, A., and Nayak, K. (2021). Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *ACM CCS*.

[Blum, 1983] Blum, M. (1983). Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*.

[Boldyreva, 2003] Boldyreva, A. (2003). Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*.

[Boneh et al., 2018a] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. (2018a). Verifiable Delay Functions. In *Annual International Cryptology Conference*.

[Boneh et al., 2019] Boneh, D., Bünz, B., and Fisch, B. (2019). Batching techniques for accumulators with applications to IOPs and stateless blockchains. In *Annual International Cryptology Conference*.

[Boneh et al., 2018b] Boneh, D., Bünz, B., and Fisch, B. (2018b). A Survey of Two Verifiable Delay Functions. Cryptology ePrint Archive, Paper 2018/712.

[Boneh et al., 2018c] Boneh, D., Drijvers, M., and Neven, G. (2018c). Compact multi-signatures for smaller blockchains. In *Asiacrypt*.

[Boneh et al., 2020] Boneh, D., Eskandarian, S., Hanzlik, L., and Greco, N. (2020). Single secret leader election. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*.

[Boneh et al., 2001] Boneh, D., Lynn, B., and Shacham, H. (2001). Short signatures from the weil pairing. In *Asiacrypt*.

[Boneh and Naor, 2000] Boneh, D. and Naor, M. (2000). Timed commitments. In *Annual International Cryptology Conference*.

[Bonneau et al., 2025] Bonneau, J., Bünz, B., Christ, M., and Efron, Y. (2025). Good things come to those who wait: Dishonest-majority coin-flipping requires delay functions. In *Eurocrypt*.

[Bonneau et al., 2015] Bonneau, J., Clark, J., and Goldfeder, S. (2015). On Bitcoin as a public randomness source. *IACR Cryptol. ePrint Arch.*

[Boppana and Narayanan, 2000] Boppana, R. B. and Narayanan, B. O. (2000). Perfect-information leader election with optimal resilience. *SIAM Journal on Computing*, 29(4).

[Brorsson and Gunnarsson, 2023] Brorsson, J. and Gunnarsson, M. (2023). Dipsauce: efficient private stream aggregation without trusted parties. In *Nordic Conference on Secure IT Systems*.

[Buchmann and Hamdy, 2011] Buchmann, J. and Hamdy, S. (2011). A survey on IQ cryptography. In *Public-Key Cryptography and Computational Number Theory*.

[Bünz et al., 2018] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. (2018). Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Security & Privacy*.

[Bünz and Fisch, 2022] Bünz, B. and Fisch, B. (2022). Schwartz-zippel for multilinear polynomials mod n. Cryptology ePrint Archive, Paper 2022/458.

[Bünz et al., 2017] Bünz, B., Goldfeder, S., and Bonneau, J. (2017). Proofs-of-delay and randomness beacons in Ethereum. *IEEE Security and Privacy on the blockchain (IEEE S&B)*.

[Cachin et al., 2005] Cachin, C., Kursawe, K., and Shoup, V. (2005). Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*.

[Camenisch et al., 2022] Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y.-A., Shoup, V., and Williams, D. (2022). Internet computer consensus. In *ACM PODC*.

[Camenisch and Lysyanskaya, 2002] Camenisch, J. and Lysyanskaya, A. (2002). Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Annual International Cryptology Conference*.

[Canetti, 2001] Canetti, R. (2001). Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*.

[Cascudo and David, 2017] Cascudo, I. and David, B. (2017). SCRAPE: Scalable randomness attested by public entities. In *ACNS*.

[Cascudo and David, 2020] Cascudo, I. and David, B. (2020). Albatross: publicly attestable batched randomness based on secret sharing. In *Asiacrypt*.

[Cascudo et al., 2021] Cascudo, I., David, B., Shlomovits, O., and Varlakov, D. (2021). Mt. random: Multi-tiered randomness beacons. *Cryptology ePrint Archive*.

[Castro and Liskov, 1999] Castro, M. and Liskov, B. (1999). Practical byzantine fault tolerance. In *OSDI*.

[Catalano and Fiore, 2013] Catalano, D. and Fiore, D. (2013). Vector commitments and their applications. In *PKC*.

[Chatzigiannis and Chalkias, 2021] Chatzigiannis, P. and Chalkias, K. (2021). Proof of assets in the diem blockchain. In *Applied Cryptography and Network Security*.

[Chaum and Pedersen, 1992] Chaum, D. and Pedersen, T. P. (1992). Wallet databases with observers. In *Annual International Cryptology Conference*.

[Chavez-Saab et al., 2021] Chavez-Saab, J., Rodríguez-Henríquez, F., and Tibouchi, M. (2021). Verifiable isogeny walks: Towards an isogeny-based postquantum vdf. In *International Conference on Selected Areas in Cryptography*.

[Chen et al., 2024] Chen, J., Zhao, Z., Messou, F. J. A., Katabarwa, R., Alfarraj, O., Yu, K., and Guizani, M. (2024). A byzantine-fault-tolerant federated learning method using tree-decentralized network and knowledge distillation for internet of vehicles. In *IEEE Vehicular Technology Conference*.

[Chen et al., 2021] Chen, M., Hazay, C., Ishai, Y., Kashnikov, Y., Micciancio, D., Riviere, T., Shelat, A., Venkitasubramaniam, M., and Wang, R. (2021). Diogenes: Lightweight Scalable RSA Modulus Generation with a Dishonest Majority. In *IEEE Security & Privacy*.

[Cherniaeva et al., 2019] Cherniaeva, A., Shirobokov, I., and Shlomovits, O. (2019). Homomorphic encryption random beacon. *Cryptology ePrint Archive*.

[Christ et al., 2024] Christ, M., Choi, K., McKelvie, W., Bonneau, J., and Malkin, T. (2024). Accountable secret leader election. In *Advances in Financial Technologies*.

[Cini et al., 2023] Cini, V., Lai, R. W., and Malavolta, G. (2023). Lattice-based succinct arguments from vanishing polynomials. In *Annual International Cryptology Conference*.

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

[Cleve, 1986] Cleve, R. (1986). Limits on the security of coin flips when half the processors are faulty. In *TOC*.

[Cleve and Impagliazzo, 1993] Cleve, R. and Impagliazzo, R. (1993). Martingales, collective coin flipping and discrete control processes. *Manuscript*.

[Couteau et al., 2021] Couteau, G., Klooß, M., Lin, H., and Reichle, M. (2021). Efficient range proofs with transparent setup from bounded integer commitments. In *Eurocrypt*.

[Damgård, 1998] Damgård, I. (1998). Commitment schemes and zero-knowledge protocols. In *School organized by the European Educational Forum*.

[Damgård, 2002] Damgård, I. (2002). On σ-protocols. *Lecture Notes, University of Aarhus*.

[Das et al., 2022] Das, S., Krishnan, V., Isaac, I. M., and Ren, L. (2022). SPURT: Scalable distributed randomness beacon with transparent setup. In *IEEE Security & Privacy*.

[Das et al., 2025] Das, S., Pinkas, B., Tomescu, A., and Xiang, Z. (2025). Distributed randomness using weighted vufs. In *Eurocrypt*.

[Das and Ren, 2024] Das, S. and Ren, L. (2024). Adaptively secure bls threshold signatures from ddh and co-cdh. In *Annual International Cryptology Conference*.

[David et al., 2018] David, B., Gaži, P., Kiayias, A., and Russell, A. (2018). Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Eurocrypt*.

[David et al., 2022] David, B., Magri, B., Matt, C., Nielsen, J. B., and Tschudi, D. (2022). Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In *ACM CCS*.

[De Feo et al., 2019] De Feo, L., Masson, S., Petit, C., and Sanso, A. (2019). Verifiable delay functions from supersingular isogenies and pairings. In *Eurocrypt*.

[Desmedt and Frankel, 1990] Desmedt, Y. and Frankel, Y. (1990). Threshold cryptosystems. In *Advances in Cryptology — CRYPTO' 89 Proceedings*.

[Dodis, 2000] Dodis, Y. (2000). Impossibility of black-box reduction from non-adaptively to adaptively secure coin-flipping. In *ECCC*.

[Dodis and Yampolskiy, 2005] Dodis, Y. and Yampolskiy, A. (2005). A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*.

[drand, 2020] drand (2020). Drand. https://drand.love.

[Duan et al., 2023] Duan, S., Wang, X., and Zhang, H. (2023). Fin: Practical signature-free asynchronous common subset in constant time. In *ACM CCS*.

[Edgington, 2023] Edgington, B. (2023). A technical handbook on ethereum's move to proof of stake and beyond. In *ETH2 Book*.

[Etesami et al., 2020] Etesami, O., Mahloujifar, S., and Mahmoody, M. (2020). Computational concentration of measure: Optimal bounds, reductions, and more. In *SODA*.

[Feige, 1999] Feige, U. (1999). Noncryptographic selection protocols. In *FOCS*.

[Feist, 2022] Feist, D. (2022). RSA Assumptions. rsa.cash/rsa-assumptions.

[Feldman, 1987] Feldman, P. (1987). A practical scheme for non-interactive verifiable secret sharing. In *IEEE 28th Annual Symposium on Foundations of Computer Science*.

[Feng et al., 2024] Feng, H., Lu, Z., and Tang, Q. (2024). Dragon: Decentralization at the cost of representation after arbitrary grouping and its applications to sub-cubic dkg and interactive consistency. In *ACM Symposium on Principles of Distributed Computing*.

[Fiat and Shamir, 1986] Fiat, A. and Shamir, A. (1986). How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*.

[FileCoin, 2020] FileCoin (2020). Trusted setup complete! https://filecoin.io/blog/posts/trusted-setup-complete.

[Fischer et al., 2011] Fischer, M. J., Iorga, M., and Peralta, R. (2011). A public randomness service. In *Proceedings of the International Conference on Security and Cryptography*. IEEE.

[Foundation, 2023] Foundation, E. (2023). Proto-danksharding. https://www.eip4844.com.

[Fuchsbauer et al., 2018] Fuchsbauer, G., Kiltz, E., and Loss, J. (2018). The algebraic group model and its applications. In *CRYPTO*.

[Gabizon et al., 2019] Gabizon, A., Williamson, Z. J., and Ciobotaru, O. (2019). PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Cryptology ePrint Archive, Paper 2019/953.

[Gainsbury and Blaszczynski, 2017] Gainsbury, S. M. and Blaszczynski, A. (2017). How blockchain and cryptocurrency technology could revolutionize online gambling. *Gaming Law Review*.

[Galindo et al., 2021] Galindo, D., Liu, J., Ordean, M., and Wong, J.-M. (2021). Fully distributed verifiable random functions and their application to decentralised random beacons. In *Euro S&P*.

[Garay et al., 2015] Garay, J., Kiayias, A., and Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*.

[Garg et al., 2024a] Garg, S., Jain, A., Mukherjee, P., Sinha, R., Wang, M., and Zhang, Y. (2024a). hints: Threshold signatures with silent setup. In *2024 IEEE Symposium on Security and Privacy (SP)*.

[Garg et al., 2024b] Garg, S., Kolonelos, D., Policharla, G.-V., and Wang, M. (2024b). Threshold encryption with silent setup. In *Annual International Cryptology Conference*.

[Gennaro et al., 1999] Gennaro, R., Jarecki, S., Krawczyk, H., and Rabin, T. (1999). Secure distributed key generation for discrete-log based cryptosystems. In *International Conference on the Theory and Applications of Cryptographic Techniques*.

[Gentry et al., 2021] Gentry, C., Halevi, S., Krawczyk, H., Magri, B., Nielsen, J. B., Rabin, T., and Yakoubov, S. (2021). Yoso: you only speak once. In *Annual International Cryptology Conference*.

[Gilad et al., 2017] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. (2017). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*.

[Goldwasser et al., 2015] Goldwasser, S., Kalai, Y. T., and Park, S. (2015). Adaptively secure coin-flipping, revisited. In *ICALP*.

[Groth, 2021] Groth, J. (2021). Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Paper 2021/339.

[Guo et al., 2020] Guo, Z., Shi, L., and Xu, M. (2020). SecRand: A Secure Distributed Randomness Generation Protocol With High Practicality and Scalability. *IEEE Access*.

[Gurkan et al., 2021] Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. (2021). Aggregatable distributed key generation. In *Eurocrypt*.

[Haahr, 2010] Haahr, M. (2010). Random.org: True random number service. www.random.org.

[Haitner and Karidi-Heller, 2020] Haitner, I. and Karidi-Heller, Y. (2020). A tight lower bound on adaptively secure full-information coin flip. In *FOCS*.

[Haitner and Tsfadia, 2014] Haitner, I. and Tsfadia, E. (2014). An almost-optimally fair three-party coin-flipping protocol. In *ACM STOC*.

[Han et al., 2020] Han, R., Lin, H., and Yu, J. (2020). Randchain: A scalable and fair decentralised randomness beacon. *Cryptology ePrint Archive*.

[Johnson et al., 2024] Johnson, S., Mengersen, K., O'Callaghan, P., and Madsen, A. L. (2024). Proposer selection in eip-7251. *arXiv preprint arXiv:2404.12657*.

[Kahn et al., 1989] Kahn, J., Kalai, G., and Linial, N. (1989). The influence of variables on boolean functions. *FOCS*.

[Kalai et al., 2021] Kalai, Y. T., Komargodski, I., and Raz, R. (2021). A lower bound for adaptively-secure collective coin flipping protocols. *Combinatorica*, 41(1).

[Karthikeyan and Polychroniadou, 2024] Karthikeyan, H. and Polychroniadou, A. (2024). Picaso: Secure aggregation for federated learning with minimal synchronization. In *Advancements In Medical Foundation Models: Explainability, Robustness, Security, and Beyond*.

[Katz et al., 2020] Katz, J., Loss, J., and Xu, J. (2020). On the Security of Time-Lock Puzzles and Timed Commitments. In *TCC*.

[Kavousi et al., 2024] Kavousi, A., Wang, Z., and Jovanovic, P. (2024). Sok: Public randomness. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[Kelsey et al., 2019] Kelsey, J., Brandão, L. T., Peralta, R., and Booth, H. (2019). A reference for randomness beacons: Format and protocol version 2. Technical report, National Institute of Standards and Technology.

[Kemmoe and Lysyanskaya, 2024] Kemmoe, V. Y. and Lysyanskaya, A. (2024). Rsa-based dynamic accumulator without hashing into primes. In *ACM CCS*.

[Kemmoe et al., 2025] Kemmoe, V. Y., Lysyanskaya, A., and Nguyen, N. K. (2025). Lattice-based accumulator and application to anonymous credential revocation. *Cryptology ePrint Archive*.

[Kerber et al., 2021] Kerber, T., Kiayias, A., and Kohlweiss, M. (2021). Mining for Privacy: How to Bootstrap a Snarky Blockchain. In *Financial Crypto*.

[Khovratovich et al., 2022] Khovratovich, D., Maller, M., and Tiwari, P. R. (2022). MinRoot: candidate sequential function for Ethereum VDF. *Cryptology ePrint Archive*.

[Kiayias et al., 2017] Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*.

[Kokoris-Kogias et al., 2018] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. (2018). Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*.

[Kothapalli and Setty, 2024] Kothapalli, A. and Setty, S. (2024). Hypernova: Recursive arguments for customizable constraint systems. In *Annual International Cryptology Conference*.

[Kothapalli et al., 2022] Kothapalli, A., Setty, S., and Tzialla, I. (2022). Nova: Recursive zero-knowledge arguments from folding schemes. In *Annual International Cryptology Conference*.

[Lai and Malavolta, 2023] Lai, R. W. and Malavolta, G. (2023). Lattice-based timed cryptography. In *Annual International Cryptology Conference*.

[Langellotti, 2025] Langellotti, D. (2025). Cypherpunk cosmic randomness: ctrng beta now live. https://blog.spacecomputer.io/cypherpunk-cosmic-randomness-ctrng-beta-now-live.

[Lee et al., 2022] Lee, H., Hsu, Y., Wang, J.-J., Yang, H. C., Chen, Y.-H., Hu, Y.-C., and Hsiao, H.-C. (2022). HeadStart: Efficiently Verifiable and Low-Latency Participatory Randomness Generation at Scale. In *NDSS*.

[Lenstra and Wesolowski, 2015] Lenstra, A. K. and Wesolowski, B. (2015). A random zoo: sloth, unicorn, and trx. *Cryptology ePrint Archive*.

[Li et al., 2007] Li, J., Li, N., and Xue, R. (2007). Universal accumulators with efficient nonmembership proofs. In *ACNS*.

[Libert, 2025] Libert, B. (2025). Simplified adaptively secure threshold bls signatures. In *Cryptographers' Track at the RSA Conference*.

[Lichtenstein et al., 1989] Lichtenstein, D., Linial, N., and Saks, M. (1989). Some extremal problems arising from discrete control processes. *Combinatorica*, 9(3).

[Lipmaa, 2012] Lipmaa, H. (2012). Secure accumulators from Euclidean rings without trusted setup. In *ACNS*.

[Liu-Zhang et al., 2025] Liu-Zhang, C.-D., Masserova, E., Ribeiro, J., Soni, P., and Thyagarajan, S. A. (2025). Efficient distributed randomness generation from minimal assumptions where pa rties s peak s equentially o nce. In *Eurocrypt*.

[Long, 2018] Long, L. (2018). Binary quadratic forms. https://github.com/Chia-Network/vdf-competition/blob/master/classgroups.pdf.

[Ma et al., 2024] Ma, Y., Guo, Y., Karthikeyan, H., and Polychroniadou, A. (2024). Armadillo: Robust secure aggregation for federated learning with input validation. In *Advancements In Medical Foundation Models: Explainability, Robustness, Security, and Beyond*.

[Ma et al., 2023] Ma, Y., Woods, J., Angel, S., Polychroniadou, A., and Rabin, T. (2023). Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *IEEE Symposium on Security and Privacy (SP)*.

[Mahloujifar and Mahmoody, 2019] Mahloujifar, S. and Mahmoody, M. (2019). Can adversarially robust learning leveragecomputational hardness? In *Algorithmic Learning Theory*.

[Meng et al., 2025] Meng, X., Sui, X., Yang, Z., Rong, K., Xu, W., Chen, S., Yan, Y., and Duan, S. (2025). Rondo: Scalable and reconfiguration-friendly randomness beacon. In *NDSS*.

[Micali et al., 1999] Micali, S., Rabin, M., and Vadhan, S. (1999). Verifiable random functions. In *IEEE 40th Annual Symposium on Foundations of Computer Science*.

[Micciancio, 2005] Micciancio, D. (2005). The RSA group is pseudo-free. In *Annual International Cryptology Conference*.

[Michalevsky, 2022] Michalevsky, Y. (2022). Cryptosat launched crypto1 — the first cryptographic root-of-trust in space. https://medium.com/cryptosatellite/cryptosat-launches-crypto1-the-first-cryptographic-root-of-trust-in-space-37dcc324fe65.

[Moran et al., 2009] Moran, T., Naor, M., and Segev, G. (2009). An optimally fair coin toss. In *Theory of Cryptography Conference*.

[Nakamoto, 2008] Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

[Nguyen, 2005] Nguyen, L. (2005). Accumulators from bilinear pairings and applications. In *CT-RSA*.

[Nguyen-Van et al., 2019] Nguyen-Van, T., Nguyen-Anh, T., Le, T.-D., Nguyen-Ho, M.-P., Nguyen-Van, T., Le, N.-Q., and Nguyen-An, K. (2019). Scalable distributed random number generation based on homomorphic encryption. In *IEEE International Conference on Blockchain*.

[Nikolaenko et al., 2024] Nikolaenko, V., Ragsdale, S., Bonneau, J., and Boneh, D. (2024). Powers-of-tau to the people: Decentralizing setup ceremonies. In *ACNS*.

[Ongaro and Ousterhout, 2014] Ongaro, D. and Ousterhout, J. (2014). In search of an understandable consensus algorithm. In *USENIX annual technical conference (USENIX ATC)*.

[Osadnik et al., 2025] Osadnik, M., Kaviani, D., Cini, V., Lai, R. W., and Malavolta, G. (2025). Papercraft: Lattice-based verifiable delay function implemented. In *IEEE Security & Privacy*.

[Oshitani and Drake, 2025] Oshitani, L. and Drake, J. (2025). Eip-7917: Deterministic proposer lookahead. https://eips.ethereum.org/EIPS/eip-7917.

[Papamanthou, 2011] Papamanthou, C. (2011). *Cryptography for efficiency: new directions in authenticated data structures*. PhD thesis, Brown University.

[Papamanthou et al., 2013] Papamanthou, C., Shi, E., Tamassia, R., and Yi, K. (2013). Streaming authenticated data structures. In *Eurocrypt*.

[Pedersen, 1991a] Pedersen, T. P. (1991a). Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual International Cryptology Conference*.

[Pedersen, 1991b] Pedersen, T. P. (1991b). A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*.

[Pietrzak, 2018] Pietrzak, K. (2018). Simple Verifiable Delay Functions. In *ITCS*.

[Pippenger, 1980] Pippenger, N. (1980). On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250.

[Rabin, 1983] Rabin, M. O. (1983). Transaction protection by beacons. *Journal of Computer and System Sciences*.

[Raikwar and Gligoroski, 2022] Raikwar, M. and Gligoroski, D. (2022). SoK: Decentralized randomness beacon protocols. In *Australasian Conference on Information Security and Privacy*.

[RANDAO, 2016] RANDAO (2016). Randao: A dao working as rng of ethereum. https://github.com/randao/randao.

[Rivest et al., 1996] Rivest, R. L., Shamir, A., and Wagner, D. A. (1996). Time-lock puzzles and timed-release crypto.

[Russell et al., 1999] Russell, A., Saks, M., and Zuckerman, D. (1999). Lower bounds for leader election and collective coin-flipping in the perfect information model. In *TOC*.

[Saks, 1989] Saks, M. (1989). A robust noncryptographic protocol for collective coin flipping. *SIAM Journal on Discrete Mathematics*, 2(2).

[Scharfman, 2023] Scharfman, J. (2023). Decentralized finance (defi) fraud and hacks: Part 2. In *The Cryptocurrency and Digital Asset Fraud Casebook*.

[Schindler et al., 2021] Schindler, P., Judmayer, A., Hittmeir, M., Stifter, N., and Weippl, E. (2021). RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In *NDSS*.

[Schindler et al., 2020] Schindler, P., Judmayer, A., Stifter, N., and Weippl, E. (2020). HydRand: Efficient continuous distributed randomness. In *IEEE Security & Privacy*.

[Schoenmakers, 1999] Schoenmakers, B. (1999). A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*.

[Schwartz, 1980] Schwartz, J. T. (1980). Fast probabilistic algorithms for verification of polynomial identities. *Journal of the ACM (JACM)*, 27(4):701–717.

[Shamir, 1979] Shamir, A. (1979). How to share a secret. *Communications of the ACM*.

[Shasha et al., 2023] Shasha, D., Kim, T., Bonneau, J., Michalevsky, Y., Shotan, G., and Winetraub, Y. (2023). High performance, low energy, and trustworthy blockchains using satellites. *Foundations and Trends® in Networking*.

[Srinivasan et al., 2022a] Srinivasan, S., Chepurnoy, A., Papamanthou, C., Tomescu, A., and Zhang, Y. (2022a). Hyperproofs: Aggregating and maintaining proofs in vector commitments. In *USENIX Security*.

[Srinivasan et al., 2022b] Srinivasan, S., Karantaidou, I., Baldimtsi, F., and Papamanthou, C. (2022b). Batching, aggregation, and zero-knowledge proofs in bilinear accumulators. In *ACM CCS*.

[Syta et al., 2017] Syta, E., Jovanovic, P., Kogias, E. K., Gailly, N., Gasser, L., Khoffi, I., Fischer, M. J., and Ford, B. (2017). Scalable bias-resistant distributed randomness. In *IEEE Security & Privacy*.

[Thyagarajan et al., 2021a] Thyagarajan, S. A. K., Castagnos, G., Laguillaumie, F., and Malavolta, G. (2021a). Efficient CCA timed commitments in class groups. In *ACM CCS*.

[Thyagarajan et al., 2021b] Thyagarajan, S. A. K., Gong, T., Bhat, A., Kate, A., and Schröder, D. (2021b). OpenSquare: Decentralized repeated modular squaring service. In *ACM CCS*.

[Trevisan, 2001] Trevisan, L. (2001). Extractors and pseudorandom generators. *Journal of the ACM*.

[Trevisan and Vadhan, 2000] Trevisan, L. and Vadhan, S. (2000). Extracting randomness from samplable distributions. In *FOCS*.

[van Kempen et al., 2023] van Kempen, E., Li, Q., Marson, G. A., and Soriente, C. (2023). Lisa: Lightweight single-server secure aggregation with a public source of randomness. *arXiv preprint arXiv:2308.02208*.

[Wang et al., 2019] Wang, G., Shi, Z. J., Nixon, M., and Han, S. (2019). Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61.

[Wang et al., 2023] Wang, W., Ulichney, A., and Papamanthou, C. (2023). {BalanceProofs}: Maintainable Vector Commitments with Fast Aggregation. In *USENIX Security*.

[Wesolowski, 2019] Wesolowski, B. (2019). Efficient Verifiable Delay Functions. In *Eurocrypt*.

[Wood et al., 2014] Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger.

[Yakira et al., 2020] Yakira, D., Asayag, A., Grayevsky, I., and Keidar, I. (2020). Economically viable randomness. *CoRR*.

[Yakira et al., 2019] Yakira, D., Grayevsky, I., and Asayag, A. (2019). Rational threshold cryptosystems.

[Yu et al., 2018] Yu, Z., Au, M. H., Yang, R., Lai, J., and Xu, Q. (2018). Lattice-based universal accumulator with nonmembership arguments. In *Australasian Conference on Information Security and Privacy*.

[Zhang and Duan, 2022] Zhang, H. and Duan, S. (2022). Pace: Fully parallelizable bft from reproposable byzantine agreement. In *ACM CCS*.

[Zippel, 1979] Zippel, R. (1979). Probabilistic algorithms for sparse polynomials. In *Symbolic and algebraic manipulation*.