TU WIEN Informatics

Advances in Distributed Randomness

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktor der Technischen Wissenschaften

by

Dipl. Ing. Philipp Schindler, BSc
Registration Number 1128993

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl

The dissertation has been reviewed by:

Aviv Zohar	Gerald Quirchmayr

Vienna, 25th January, 2022

Philipp Schindler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

TU WIEN Informatics

Advances in Distributed Randomness

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der Technischen Wissenschaften

eingereicht von

Dipl. Ing. Philipp Schindler, BSc
Matrikelnummer 1128993

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dipl.-Ing. Mag. Dr. techn. Edgar Weippl

Diese Dissertation haben begutachtet:

Aviv Zohar	Gerald Quirchmayr

Wien, 25 Jänner, 2022

Philipp Schindler

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Declaration of Authorship

Dipl. Ing. Philipp Schindler, BSc

I hereby declare that I have written this Doctoral Thesis independently, that I have completely specified the utilized sources and resources and that I have definitely marked all parts of the work - including tables, maps and figures - which belong to other works or to the internet, literally or extracted, by referencing the source as borrowed.

Vienna, 25th January, 2022

Philipp Schindler

Acknowledgements

I am very glad for the support I received from so many people doing the process towards this Ph.D. thesis. Certainly, without such great support composing this thesis would not have been possible.

First, I would like to express my gratitude towards my advisor Edgar Weippl. Since writing on my master thesis, during the work on many exciting research topics, and finally during the path of my Ph.D. Prof. Weippl was always available when I needed support or guidance while also providing me with the freedom needed to pursue my research ideas. Thank you.

Furthermore, a sincere thank you goes to all the wonderful people at SBA Research I worked with during last years. I really enjoyed the endless inspiring discussions with the different research teams as well as the opportunity to join the colleges from SBA's commercial services division on many interesting projects. I thank SBA Research not only for the financial support making this dissertation possible, but also the people and friends there for making this unique workplace feel like a little second home.

Aljosha and Nicholas, you really give me a hard time to now express how much I enjoyed collaborating with both of you during the last couple of years. I am so glad and honored to be tackling so many interesting and challenging topics with you guys. I deeply thank both of you, Nicholas and Aljosha, for the outstanding discussions, never ending support, essential feedback and the amazing time we spent and will spend together.

Finally, there is one person who joined not only the journey through my studies but my life's journey. Martha. Since we first met you have always been on my side. I cannot begin to express how much your support and your belief in me mean to me. And thank you not only for being there for me in difficult times but also for celebrating the good ones with me. I am so glad for all the countless awesome moments we experienced together and the many many more to come. I love you.

Abstract

Distributed randomness, the problem of how a network of computers can jointly and securely generate a sequence of verifiably random values, is a long standing research topic in computer science. Recently, the problem has seen a large amount of renewed interest, in particular due to technical innovations emerging in the field of public distributed ledgers. Whereas Bitcoin was the first to implement a digital currency which does not rely on a central party, we now see thousands of projects building platforms and applications far beyond the functionality of a payment system. Still, many large real-world systems in this domain are powered by Proof-of-Work based consensus algorithms, the core concept powering Bitcoin. Unfortunately, Proof-of-Work inherently requires a tremendous amount of electrical power to ensure the system's security. This is one of the key reasons why leading projects in this field have already switched to, or are considering, alternative designs, most prominently Proof-of-Stake. And distributed randomness, for example, used to implement leader selection, sharding, or the resolution of ties in the consensus algorithm turns out to be an essential component of these designs.

To support the development of these alternatives, as well as a number of other use cases, this thesis sets out to explore, analyze and improve upon existing approaches for distributed randomness. Our main results include two novel protocol designs named HydRand and RandRunner. With HydRand we not only improve upon the theoretical efficiency compared to other designs with similar guarantees but also demonstrate that these results are translatable into practice by providing our prototype implementation. RandRunner's design, despite being quite minimalistic, furthermore improves communication to a single message being propagated to produce a fresh random output. Additionally, it achieves a wide range of desirable security guarantees and properties, for example, being able to automatically recover from temporary network failures. These improvements are (in part) made possible by leveraging our novel construction of a trapdoor verifiable delay function with strong uniqueness.

Kurzfassung

Verfahren zur verteilten und manipulationssicheren Erzeugung von verifizierbaren Zufallszahlen in einem Computernetzwerk (distributed randomness) sind ein seit langem offenes Forschungsthema der Informatik. Insbesondere die jüngsten Entwicklungen aus dem Bereich der öffentlichen, verteilten Systeme (distributed ledgers) führen zu einem neuen starken Interesse an diesem Forschungsfeld. Zu den Vorreitern dieser Systeme zählt Bitcoin, eine digitale Währung, die unabhängig von zentralen Stellen erzeugt und verwaltet wird. Mittlerweile gibt es jedoch auch eine große Bandbreite an Projekten, Plattformen und Anwendungen, die in ihrer Funktionsweise weit über die Funktionalität eines Zahlungssystems hinausgehen. Eine Vielzahl dieser Systeme basiert trotz der fortschreitenden technischen Innovation immer noch auf dem Prinzip des Proof-of-Work, das bereits bei Bitcoin zum Einsatz kommt. Der enorme Ressourcenverbrauch, der in Proof-of-Work basierten Systemen notwendig ist um Sicherheit zu garantieren, wird jedoch zunehmend als problematisch angesehen. Dies ist einer der Hauptgründe warum führende Projekte in diesem Bereich bereits auf alternative Konstruktionen, insbesondere Proof-of-Stake, umgestellt haben oder diese in Betracht ziehen. Diesbezüglich stellt sich die verteilte Erzeugung von Zufallszahlen als Kernbestandteil heraus und findet unter anderem bei der Implementierung von Auswahlmechanismen für autoritative Knoten (leader selection), Sharding oder bei der Konfliktlösung in Entscheidungsprozessen der Konsensalgorithmen ihre Anwendung.

Um diese Entwicklungen, sowie zahlreiche weitere Anwendungsfälle zu unterstützen, setzt sich diese Dissertation zum Ziel bestehende Ansätze zu erforschen und entsprechend weiterzuentwickeln. Die wichtigsten Ergebnisse umfassen zwei neue Protokolle: HydRand und RandRunner. Mit HydRand wird nicht nur die theoretische Effizienz im Vergleich zu anderen Designs mit ähnlichen Garantien verbessert, sondern auch mit einer prototypischen Implementierung aufgezeigt, dass diese Ergebnisse in die Praxis überführbar sind. Darüber hinausgehend wird gezeigt, dass RandRunner mit Hilfe eines minimalistischen Designs den Kommunikationsaufwand weiter reduzieren kann und die Erstellung eines neuen zufälligen Werts mit nur einer einzelnen Nachricht, die durch das Netzwerk verteilt wird, sicherstellen kann. Zudem bietet das Protokoll zahlreiche wichtige Sicherheitsgarantien und gewünschte Eigenschaften, zum Beispiel die Fähigkeit der automatischen Betriebswiederaufnahme nach einem Netzwerkausfall. Diese Verbesserungen werden unter anderem durch den Einsatz eines weiterentwickelten kryptografischen Verfahrens, der sogenannten „Trapdoor Verifiable Delay Functions with Strong Uniqueness“, ermöglicht.

Contents

CHAPTER 1

Introduction

1.1 Motivation

A classic example of generating randomness is rolling a dice. In an idealized world, one would expect the dice to eventually land on one of its six sides with equal probability. In the real world the results may differ depending of a range of circumstances: A dice may roll of the table, raising the question if it should be counted anyway or rerolled, or, for example, physical imperfections, a deliberately manipulated dice, or a skilled dice roller may alter the odds of the dice landing on a particular number. Depending on the stakes at play, a range of countermeasures from using standardized transparent dice to specific rules on how to perform rolls are used to ensure the fairness of dice rolls.

The topic of how to generate randomness/random numbers has also been extensively studied in a range of (academic) disciplines. Within this work, we focus on the computer science aspects, although there are multiple touching points with other disciplines, for example, with physics and electronics considering the design and implementation of *hardware random number generators* (HRNGs). These HRNGs, often also referred to as *true random number generators* (TRNGs), are devices which exploit physical phenomena that fundamentally behave in a nondeterministic way (e.g., electrical and thermal noise, radioactive decay, or quantum effects such as photon arrival times) to provide a sequence of unpredictable random numbers for computer programs [113]. This approach is in great contrast to *pseudo random number generators* (PRNGs) – algorithms which essentially generate a sequence of "random-looking" numbers in a deterministic manner. Statistical samples, computer games, simulations or randomized algorithms are some of the many applications where PRNGs are used in favor of TRNGs, e.g., due to their superior performance characteristics or ease of implementation. For a range of other applications, in particular considering cryptographic protocols, the use of PRNGs introduces serious security vulnerabilities and must therefore be strictly avoided. For these kinds of security critical applications specifically designed *cryptographically secure pseudorandom number*

generators (CSPRNGs) are employed. CSPRNGs are initialized using a high-quality source of entropy. Modern operating systems use a combination of multiple sources (device drivers, keyboard/mouse interrupts, harddisk access pattern, inter-interrupt timings, HRNGs) to ensure a proper initialization of the used CSPRNGs [50]. Assuming the correct implementation and initialization of CSPRNGs – by itself a challenging but well understood task beyond the scope of this dissertation – modern computers are able to securely generate randomness for cryptographic use. Moreover, they are not only capable of doing so, but actually many day-to-day applications including secure messaging protocols, encrypted file storage, internet communication protocols, authentication solutions and digital signatures heavily rely on this capability.

Unfortunately, these *local* approaches to generate randomness fail to meet key criteria for applications which use randomness *collectively*. Reconsidering the dice example from the beginning of this introduction, imagine a multiplayer dice game played over the phone.[1] In this case, the person rolling the dice literally tells the other person the resulting number as the other party cannot directly observe the result. This approach however immediately raises the question of trust. Previously, i.e., when playing a dice game locally (e.g., by everyone sitting around a table), all players can observe/verify the dice roll and agree on the result. In the over-the-phone scenario, however, the "observing" players cannot actually verify the result of the dice roll – they have to go by the word of the rolling player, who may dishonestly claim the dice shows a six when it indeed landed on a different side. Considering distributed systems the same issue applies. Here, generating randomness on one computer and (securely) sending the result to another computer leaves the latter with no choice but to blindly trust that the randomness was indeed generated using a suitable method and not manipulated. For a large range of applications including, for example, Byzantine fault tolerant (BFT) and blockchain-based consensus protocols, e-voting, verifiable selection processes, online gaming and gambling, or parameter generation for cryptographic protocols, the introduction of these trust assumptions are undesirable at best or intolerable at worst. This raises the demand for solutions which are suitable for collective use in a distributed setting – typically discussed under the terms of *distributed randomness* or *randomness beacons*.

In particular, also the tremendous growth and new developments in the field of blockchain- and distributed ledger technologies since the emergence of Bitcoin [86] in 2008 sparked renewed interest in the topic of distributed randomness. It is an essential component of the core of Bitcoin, Ethereum [120] and related Proof-of-Work based consensus protocols. One essential function provided by the stochastic process of mining in these kinds of protocols is the randomized selection of an individual among the pool of participating miners, which is then allowed to advance the state of the blockchain by appending a new block. The underlying mechanism of Proof-of-Work is typically implemented as the computationally difficult task of finding specially formed inputs to cryptographic hash functions, which result in outputs with a large number of zero-bits. The advantages and

[1] The presented example is inspired by M. Blum's article "Coin flipping by telephone" [17].

drawbacks of such an approach as the basis for a consensus algorithm are heavily and controversially discussed among the different communities. Avoiding essential drawbacks such as the very high consumption of electricity, many leading projects in the field, including, for example, the second largest cryptocurrency and smart contract platform Ethereum are in the process of transitioning away from a Proof-of-Work based mechanism. Others, for example, Cardano, emerging from the academic research on the Ouroboros protocol family [78, 46, 7], Ripple [109], AlgoRand [42, 69] or Dfinity [72] are designed from the ground up based on alternative mechanisms. These alternatives to Proof-of-Work are commonly based on the idea of Proof-of-Stake or use traditional Byzantine fault tolerant (BFT) protocols and heavily rely on the secure generation of distributed randomness for their operation.

The need for distributed randomness manifests itself on many different levels. On the most fundamental level, Fischer, Lynch and Paterson [59] proved that reaching consensus with one faulty node is impossible for any deterministic algorithm under asynchronous network conditions. Consequently, already early on approaches were conceived that rely on randomization to overcome this impossibility and solve the underlying Byzantine agreement problem under such a system model. Notably, Ben-Or's solution [11] employs locally generated random bits to solve the problem. However, depending on the number of Byzantine nodes, the protocol is rather inefficient and may only reach agreement after an exponential number of rounds in expectation. Rabin [96] improved upon this result by assuming a lottery service – essentially a distributed randomness beacon – which regularly provides all protocol participants with an agreed upon random value. Similarly, in practice we now see many recently developed BFT and Proof-of-Stake based protocol designs employing distributed randomness. Here, the application of distributed randomness is not limited to the core of the consensus protocol, but it is also useful in the context of improving scalability (sharding), to ensure order-fairness, or on an application level where secure access to randomness is, for example, needed within the deterministic execution environment of smart contracts.

1.2 Aims

Already back in 1983, long before the emergence of modern distributed ledger technologies, the first methods of how to generate distributed randomness were being discussed by Blum [17] under the term *coin flipping protocols* and Rabin [97] introducing the notion of a (randomness) *beacon* (protocol), emitting a random integer within a chosen range at regularly spaced time intervals. Since then, and in particular in recent history, a range of new techniques and protocols addressing the problem have emerged. This work sets out to systematically collect, analyze, compare and improve upon the existing solutions within this domain. The aim of the work is extended to also include the design and implementation of supporting building blockings, in particular distributed key generation (DKG) protocols, which, in combination with threshold signature algorithms, serve as the basis of some modern approaches for generating distributed randomness. By pursuing

these research challenges, this thesis sets out to support the many current and future advancements in the larger field of distributed ledger technologies.

1.3 Results

Considering the design and implementation of state-of-the-art randomness beacon protocols, our first main result is the HydRand protocol. HydRand is a novel method for collectively generating a sequence of random values, emitted continuously at regular intervals. HydRand's design is based on well established cryptographic primitives, in particular digital signatures and publicly-verifiable secret sharing (PVSS), and avoids the need for distributed key generation (DKG) for the protocol setup. HydRand fulfills all key properties expected from a randomness beacon protocol. In particular, it ensures the property of guaranteed output delivery, which guarantees that a new value is produced timely, even if an adversary's leader or coalition of less than a third of the nodes does not follow the described protocol rules. Compared to other protocols based on similar assumptions, HydRand uses a pipelining approach which manages to reduce the communication complexity of the protocol from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ per round.

Alongside the design and implementation of the HydRand protocol itself, a range of other protocol designs in this field have been surveyed and compared. The analyzed designs include other PVSS-based protocols as well as protocols using threshold signatures, hash-chains or verifiable random functions (VRFs) as the main cryptographic building blocks. Furthermore, methods based on Proof-of-Work have been examined and compared. Our comparison results highlight the advantages, drawbacks and tradeoffs made by the different designs and suggest that there is no clear best protocol emerging. Nevertheless, we show that HydRand minimizes drawbacks and achieves the various desirable properties in a unique way. Both, the main protocol, its evaluation, as well as the survey and comparison of the other state-of-the-art protocols, have been published in the corresponding research paper accepted at the 2020 IEEE Symposium on Security and Privacy (SP 2020) [106].

Our survey conducted on the different methods for generating randomness distributedly highlights a range of efficient protocol designs, for example, the approach described by Cachin et al. [33] or Dfinity [72], which require the use of a DKG protocol for the trustless setup of the private keys used for threshold signing in the operational stage of the protocols.[2] Unfortunately, the availability of real world implementations for DKG, in particular considering the recent advancement in the field of public distributed ledgers, is rather limited. Leveraging these advancements enabled us to design, implement and evaluate EthDKG, a DKG protocol targeting the deployment on modern smart contract platforms such as Ethereum. EthDKG's core is based on the very well established line of research on DKG by Gennaro et al. [63, 64] (itself based on Pedersen's works [92,

[2] Although, alternatively, the use of a trusted dealer distributing key shares to the participants is possible in principle, this approach is highly undesirable in practise as it introduces a single point of failure/trust.

91] and Feldman's verifiable secret sharing [56]). Through the careful use of smart contract platforms as a base layer for secure and publicly-verifiable message exchange and cryptographic commitments, we obtain an efficient practical system, while keeping the simplicity of the original design by Gennaro et al. and incorporating more recent advancements such as the dispute resolution mechanism described by Neji et al. [87]. A key achievement of our design is that the correctness of the protocol's execution is verifiable in-band, i.e., within the smart contract platform, while tolerating minority attacks on the safety and liveness of the protocol. Our design can easily be augmented using the platform's features. For example, security deposits can be mandated for all joining participants. In this case, the safety of an honest party's deposit is ensured even considering a majority attack on the protocol. Furthermore, our design allows for dynamic participation rules specified upon the deployment of the respective smart contract as well as efficient on-platform verification of threshold signatures. We open sourced the protocol's implementation, demonstrated its practical performance via extensive benchmarks, and presented our results at the 2019 Cryptocurrency's Implementer Workshop (CIW 2019) at the 23rd International Conference on Financial Cryptography and Data Security conference (FC 2019). In addition, a further improved version of the paper is available on the Cryptology ePrint Archive [105].

Finally, we design a new novel randomness beacon protocol named RandRunner. RandRunner's central cryptographical building block is a special kind of trapdoor verifiable delay function (T-VDF) which does achieve the property of strong uniqueness we introduce in Section 4.3.2. While previous constructions of T-VDFs (see, e.g., the work by Wesolowski [118]) are able to guarantee uniqueness for a party without possession of the trapdoor, our new construction first demonstrates how uniqueness can be guaranteed despite an adversarial party with knowledge of the trapdoor. This is a key requirement for the use of T-VDFs within a distributed randomness beacon because it guarantees bias-resistance when computing the next beacon value using input(s) provided by an adversary. Leveraging the strong uniqueness property, we obtain not only a highly efficient but also very simplistic protocol: In essence, the protocol outputs a fresh random beacon value as the (regularly changing) protocol leader disseminates a single message, i.e., the result of evaluating the leader's T-VDF on the previous beacon output using the trapdoor. In case of an unresponsive or adversarial leader, all other nodes compute the leader's T-VDF without using the (to them unknown) trapdoor and obtain the same result after a short delay. Within our work published at the Network and Distributed System Security Symposium (NDSS 2021) [103], we furthermore highlight additional properties including the resilience of the protocol against temporary network delays or outages, responsiveness under good network conditions, or a non-interactively verifiable protocol setup without requiring a DKG protocol.

The main results summarized above, have been presented at a range of highly renowned international conferences:

P. Schindler, A. Judmayer, N. Stifter, and E. Weippl. HydRand: Efficient Continuous Distributed Randomness. In *2020 IEEE Symposium on Security and Privacy (SP 2020)*,

pages 32–48. IEEE, May 2020.

P. Schindler, A. Judmayer, N. Stifter, and E. Weippl. EthDKG: Distributed Key Generation with Ethereum Smart Contracts. Cryptology ePrint Archive, Report 2019/985, 2019. [3]

P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. Weippl. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness. In *28th Annual Network and Distributed System Security Symposium (NDSS 2021)*. The Internet Society, February 2021.

In the spirit of open science – to ensure reproducibility of our results and foster further improvements – we made the research artifacts developed during the work on the HydRand-, EthDKG- and RandRunner protocols publicly accessible. In particular, the source code for the prototypes and simulations, additional documentation, as well as the developed evaluation tools and benchmark results have been published on the respective Github repositories:

- https://github.com/PhilippSchindler/HydRand
- https://github.com/PhilippSchindler/EthDKG
- https://github.com/PhilippSchindler/RandRunner

In addition to the results presented within this Ph.D. thesis, a number of contributions on related research topics in the field of blockchain and distributed ledger technologies have been made during the course of this studies:

A. Judmayer, N. Stifter, **P. Schindler**, and E. Weippl. Estimating (Miner) Extractable Value is Hard, Let's Go Shopping! In *Financial Cryptography and Data Security FC 2022 International Workshop on Coordination of Decentralized Finance (CoDecFin)*. Springer, February 2022.

N. Stifter, A. Judmayer, **P. Schindler**, A. Kern, and W. Fdhila. What is Meant by Permissionless Blockchains? In *2021 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE, October 2021.

K. Pfeffer, A. Mai, A. Dabrowski, M. Gusenbauer, **P. Schindler**, E. Weippl, M. Franz, and K. Krombholz. On the Usability of Authenticity Checks for Hardware Security Tokens. In *30th USENIX Security Symposium (USENIX Security 2021)*, pages 37–54. USENIX Association, August 2021.

[3] An earlier revision of this work was presented at the Cryptocurrency Implementers' Workshop (CIW 2019) in association with the 23rd International Conference on Financial Cryptography and Data Security (FC 2019).

N. Stifter, A. Judmayer, **P. Schindler**, and E. Weippl. Opportunistic Algorithmic Double-Spending: How I learned to stop worrying and hedge the Fork. Cryptology ePrint Archive, Report 2021/1182, 2021.

A. Judmayer, **P. Schindler**, and N. Stifter. Blockchain-Technologie – Anwendungsformen. LexisNexis, April 2021.

A. Judmayer, **P. Schindler**, and N. Stifter. Blockchain-Technologie – technische Erklärung. LexisNexis, January 2020.

N. Stifter, **P. Schindler**, A. Judmayer, A. Zamyatin, A. Kern, and E. Weippl. Echoes of the Past: Recovering Blockchain Metrics From Merged Mining. In *International Conference on Financial Cryptography and Data Security (FC 2019)*, pages 527–549. Springer, February 2019.

A. Judmayer, **P. Schindler**, N. Stifter, and E. Weippl. Book chapter "Blockchain: Basics" in Business Transformation through Blockchain. pages 339–355. Springer, 2019.

A. Zamyatin, N. Stifter, A. Judmayer, **P. Schindler**, E. Weippl, and W. J. Knottenbelt. A Wild Velvet Fork Appears! Inclusive Blockchain Protocol Changes in Practice (Short Paper). In *Financial Cryptography and Data Security FC 2018 International Workshops, BITCOIN, VOTING, and WTSC*, pages 31–42. Springer, March 2018.

A. Judmayer, N. Stifter, **P. Schindler**, and E. Weippl. Pitchforks in Cryptocurrencies: Enforcing Rule Changes Through Offensive Forking and Consensus Techniques (Short Paper). In *Cryptocurrencies and Blockchain Technology ESORICS 2018 International Workshop (CBT 2018)*, pages 197–206. Springer, September 2018.

A. Zamyatin, N. Stifter, **P. Schindler**, E. Weippl, and W. J. Knottenbelt. Flux: Revisiting Near Blocks for Proof-of-Work Blockchains. Cryptology ePrint Archive, Report 2018/415, 2018.

N. Stifter, A. Judmayer, **P. Schindler**, A. Zamyatin, and E. Weippl. Agreement with Satoshi – On the Formalization of Nakamoto Consensus. Cryptology ePrint Archive, Report 2018/400, 2018.

1.4 Methodology

To achieve our stated goals, we base our methodological approach for conducting the work on this dissertation on the design science paradigm as described by Hevner et al. in their essay "Design Science in Information Systems Research" [73]. They describe design science as a problem solving process, where knowledge and understanding of a design problem with its corresponding solution(s) are attained by creating a design artifact and applying it to the problem domain. In order to support researchers in effectively conducting design science research, they derive the following set of seven guidelines:

- Design as an Artifact

- Problem Relevance
- Research Rigor
- Design Evaluation
- Design as a Search Process
- Research Contribution
- Communication of Research

We find that we can apply many of the stated principles throughout the work on this dissertation. The following subsections provide details on how the guidelines are implemented within the scope of this work. Thereby, we follow Hevner et al.'s recommendations which advocate for a creative and careful assessment of when, where, and in which way researchers should apply the guidelines, and against a strict mandatory use.

1.4.1 Design as an Artifact

Hevner et al. [73], in their first guideline, define the result of a design science research process to be a purposeful, innovative and effectively described IT artifact. They do not limit their definition of the term artifact to the instantiation (i.e., a software implementation in our context), but rather consider it in a broader way, including the underlying constructs, models and methods. Following this definition, the artifacts originating from the work on this dissertation are the three developed protocols: HydRand, EthDKG and RandRunner. The artifacts include the protocol descriptions of all three protocols, the implementation of HydRand and EthDKG as well as the simulation model created for RandRunner. As RandRunner's trapdoor verifiable delay function with strong uniqueness subcomponent is not only a viable building block for the protocol itself but also of independent interest, we additionally consider it as a design artifact according to Hevner et al.'s categorization. The artifacts are purpose-built to address open and relevant research problems, as described in Section 1.1 and the following subsection. Furthermore, the innovative nature of the designed protocols is highlighted by comparing them to existing state-of-the-art designs and showing improvements upon the prior state-of-the-art in many aspects. A summary of our main results is presented in Section 1.3, whereas we refer to the chapters 2, 3 and 4 for detailed descriptions of the specific artifacts.

1.4.2 Problem Relevance

The objective of design science research is the development of technological solutions to important and relevant business problems [73]. Although we typically do not characterize modern distributed ledger technologies as businesses, we see a tremendous amount of interest from users, research groups, foundations and traditional companies in these technologies. As we describe in detail in Section 1.1, solutions for the distributed randomness problem are highly relevant in this context. The necessity can be seen on the

most fundamental level, e.g., considering the Fischer, Lynch and Paterson impossibility result [59], or on higher levels, e.g., studying a variety of protocol designs as alternatives to Proof-of-Work, or on the application layer itself where secure distributed randomness is needed within the execution environments of smart contracts. Further highlighting the significance of the problem, we identify and compare a large body of recent research which also addresses the research problem at hand. An overview and comparison of these works are provided in Section 2.7.

1.4.3 Research Rigor

Following the guideline on research rigor given by Hevner et al. [73], we set out to apply rigorous methods in both the design and the evaluation of our design artifacts. We describe the methods used to ensure a rigorous design and description in this section. An overview of our methodology regarding the evaluation of the designs is given in the next subsection. The details regarding the used evaluation methods of the individual artifacts are provided in sections 2.6, 3.7 and 4.7 respectively.

To establish an effective knowledge base upon which rigor is derived, we conduct an extensive literature review as a first step. This review includes both state-of-the-art protocols for distributed randomness generation, as well as application scenarios, in particular distributed consensus protocols. To further expand upon this initially built knowledge base, the initial review of existing designs is extended to a detailed analysis and comparison. For this purpose, first the desired properties and characteristics of distributed randomness protocols are collected. Then, the drawbacks, advantages and tradeoffs of the individual designs are identified and compared with each other. The identified shortcomings and areas of potential improvements serve as the basis for the further design and prototypical implementation of new protocols in the next steps.

To rigorously describe our research artifacts, in particular the three protocol designs HydRand, EthDKG, RandRunner and their required subcomponents, we first informally describe the most fundamental design principles. The goal of this first step is to provide the reader with the key intuitions required to more easily apprehend the details of full protocol specifications given afterward. To allow for precise protocol specification a certain level of abstraction is needed. Following Hevner et al.'s cautious words regarding excessive formalism [73], and our own experience studying related work written with different levels of formalism, we attempt to find a good balance between the use of informal language and a highly formal mathematical notation for the description of our protocols. Furthermore, we explicitly state the assumptions we make in the respective system- and threat models of our designs, and wherever possible stick to assumptions widely known and used in related works.

1.4.4 Design Evaluation

In order to demonstrate the utility, quality, and efficiency of our design artifacts we perform an extensive evaluation for each of the artifacts. This evaluation is based

on two main pillars, formal reasoning and practical demonstration via prototypical implementations and simulations.

Regarding HydRand, we show it achieves its intended design goals by providing formal security proofs for each of the expected properties. A theoretical analysis shows that the per-round communication complexity can be reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. This result is complemented by extensively testing and benchmarking a prototype implementation of the protocol. The benchmark results, presented in Section 2.6, demonstrate that the approach does not only provide theoretical improvements but instead also highlights its applicability for practical use given the high throughput, fast verification times and low resource requirements.

EthDKG's quality is ensured by reducing its main security guarantees to the protocols it built upon and showing that any modification made does not negatively affect the original guarantees provided. Its practical utility is demonstrated by first implementing the protocol for the Ethereum platform and then deploying, running and documenting its execution on the Ethereum testnet Ropsten. Its efficiency is shown by executing the protocol under different configurations in a local environment and measuring key performance metrics such as execution time and computational costs in terms of gas usage. The details on our evaluation methods and results are presented in Section 3.7.

Similar to HydRand, we also formally analyze our protocol RandRunner. We first show the security of the underlying verifiable delay function (VDF) by providing a formal proof. This is accomplished by showing that our construction ensures all preconditions required to apply the security proof of the VDF designed by K. Pietrzak [94] which forms the base of our VDF variant. After establishing the security of this key component, we can use it (or any potential future construction with the required guarantees) in a black-box manner to formally prove the properties of the overall protocol. We refer to sections 4.3.6 and 4.7 for the details on the security proofs of the underlying VDF and the full protocol design respectively. This theoretical analysis is accompanied by developing a simulation model which is used to evaluate important protocol characteristics including, e.g., unpredictability and recoverability from periods of asynchrony, over a large range of parameter sets. The practical efficiency of the protocol immediately follows from the simplicity of the protocol, requiring only a single, small message being propagated in each round.

1.4.5 Design as a Search Process

Hevner et al. [73] describe design science as an inherently iterative search process towards discovering an effective solution for a given problem at hand. Ideally one would search for the best possible solution. However, they find the search for an optimal solution is often intractable, considering real world problems in information systems. Based on our findings and comparisons with the related literature, we can certainly confirm that their general statement applies to our specific problem domain. Despite the infeasibility of an exhaustive approach for exploring all possible solutions, design science can still

be applied by using an approach Hevner et al. describe as heuristic search strategies. In our case, conducting an extensive literature review helps us to explore the possible design space and identify shortcomings of existing solutions. For example, in regards to protocol designs for distributed randomness based on publicly-verifiable secret sharing, we find that a high communication complexity and the lack of an integrated agreement protocol limit the utility of the explored prior solutions – two key design limitations we are able to overcome with our protocol design HydRand. Similarly, our search for different possible solutions reveals that prior solutions based on the concept of verifiable delay functions have not received great attention by other researchers. Here, a closer look into this restricted design space leads us towards our second randomness beacon design named RandRunner.

To eventually arrive at the presented protocol designs, we iteratively change the design and/or implementation of our protocols, reevaluate them, and compare the results with the previously obtained ones. There is a wide range of factors contributing to this iterative design approach. These factors include findings from our performance tests, feedback received from various internal discussions and the academic peer reviewing process. All of these factors strengthen the deeper understanding of the problem and solution space and allow us to continuously improve our protocol designs.

Another interesting observation arises when we compare the heuristic search processes used for HydRand and EthDKG with RandRunner. In contrast to HydRand and EthDKG, iterative changing and reevaluating the RandRunner protocols reveals that certain design decisions are beneficial under a particular set of preconditions, whereas conflicting decisions provide better guarantees under other circumstances. In the search to improve HydRand and EthDKG we did not discover such an effect. To resolve the situation for RandRunner, we follow both "branches" of the search tree independently and obtain two variants of the RandRunner protocols. The variants share most properties but differ in regards to the unpredictability guarantees they provide for the generated sequence of randomness beacon values. A detailed discussion on this aspect is provided in Section 4.7.4.

1.4.6 Research Contribution

According to Hevner et al. [73] "effective design-science research must provide clear contributions in the areas of the design artifact, design construction knowledge (i.e., foundations), and/or design evaluation knowledge (i.e., methodologies)". We see our main contribution in the generated design artifacts themselves and refer to Section 1.3 for a summary of the key research contributions made. We follow best practices and make the source code for all prototypical implementations, the scripts required for setting up experimental environments as well as the simulations model developed for RandRunner publicly available on the Github platform. These additional contributions may turn out to be useful for other researchers which want to reproduce our results or adopt the provided resources for their own protocol evaluations.

1.4.7 Communication of Research

As we outline in more detail in Section 1.3, the research efforts within this dissertation have been presented to an expert audience at a range of renowned scientific conferences, including the IEEE Symposium on Security and Privacy and the Annual Network and Distributed System Symposium. To communicate the results to a broader audience, in-depth as well as higher level presentations about the works have been given at a range of other national and international venues. Recorded presentations on the HydRand and RandRunner protocols are also publicly available on the video sharing platform Youtube.

1.5 Structure

This thesis is structured as follows:

Chapter 1 (this chapter) provides an introduction to the topic of randomness and distributed randomness, motivates the need for further research in this field, outlines the goals of this thesis, describes the methodology used towards achieving these goals as well as summarizes the main contributions being made.

Then, chapters 2, 3, and 4 of this thesis's structure follow the corresponding scientific publications.

Chapter 2 introduces our first design of distributed randomness beacon, named HydRand. This chapter includes the design, implementation and evaluation of the protocol itself as well as a survey and comparison of other state-of-the-art designs in the field.

Chapter 3 describes the distributed key generation protocol EthDKG. The chapter covers the importance of distributed key generation in regards to distributed randomness and other applications, the related work in this field, and the design, implementation and evaluation of the protocol.

Chapter 4 introduces our trapdoor verifiable delay function (T-VDF) with strong uniqueness and demonstrates how we leverage this cryptographic building block to design our simple, yet highly efficient randomness beacon protocol RandRunner.

Finally, Chapter 5 concludes this thesis. In this chapter, key aspects of the field of distributed randomness and our results in this context are highlighted and potential directions for future research in this field are discussed.

CHAPTER 2

HydRand: Efficient Continuous Distributed Randomness*

A reliable source of randomness is not only an essential building block in various cryptographic, security, and distributed systems protocols, but also plays an integral part in the design of many new blockchain proposals. Consequently, the topic of publicly-verifiable, bias-resistant and unpredictable randomness has recently enjoyed increased attention. In particular *random beacon protocols*, aimed at *continuous* operation, can be a vital component for current Proof-of-Stake based distributed ledger proposals. In this chapter, we improve upon previous random beacon approaches by presenting *HydRand*, a novel distributed protocol based on publicly-verifiable secret sharing (PVSS). The protocol design ensures the key properties of unpredictability, bias-resistance, and public-verifiability for a *continuous* sequence of random beacon values. Furthermore, HydRand provides guaranteed output delivery of randomness at regular and predictable intervals in the presence of adversarial behavior and does not rely on a trusted dealer for the initial setup. Compared to existing PVSS based approaches that strive to achieve similar properties, our solution improves scalability by lowering the communication complexity from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$. Furthermore, we are the first to present a detailed comparison of recently described schemes and protocols that can be used for implementing random beacons.

2.1 Introduction to Distributed Randomness

The question of how to generate trustworthy random values among a set of mutually distrusting participants over a message passing network was first addressed by Blum in

*This chapter is an updated version of the equally-named research paper [106] published at the 41st IEEE Symposium on Security and Privacy (SP) 2020. Large text passages from the original work are used in verbatim form in this work.

1983, thereby introducing the notion of *coin tossing protocols* [17]. Distributed randomness also forms a key component of *asynchronous* consensus protocols in the form of local [11] and *common coin* designs [96, 33].

Lately, coin tossing protocols have received increased attention, in part because generating shared randomness is proving to be a vital component of most distributed ledger approaches (e.g. [13, 42, 78]) that aim to replace the computationally intensive *Proof-of-Work* (PoW) mechanism as found in Bitcoin [86] and similar cryptocurrencies. Specifically, *Proof-of-Stake* (PoS) blockchain proposals, which rely on virtual resources in the form of digital assets, call for manipulation resistant and unpredictable leader election as part of a secure protocol design [78]. The distributed generation of trustworthy random values can hence be considered a complementary problem to the development of such protocols.

Random beacon protocols aim to generate publicly-verifiable, bias-resistant and unpredictable randomness in distributed environments. The concept of a random *beacon* was first formalized by Rabin [97], where a service that emits a fresh random number at regular intervals is proposed. Potential application areas for random beacons are broad and, as described in [114, 38, 25], include:

- the secure generation of protocol parameters for cryptographic schemes [8, 80]
- privacy preserving messaging services [119, 117, 70]
- protocols for anonymous browsing, including Tor hidden services [48, 71, 68]
- electronic voting protocols [1]
- publicly-auditable selections [25]
- gambling and lottery services [25]

With the emergence of blockchain protocols additional areas that demand secure sources of public randomness, such as sharding approaches [45], were formed. In particular smart contracts often draw upon insecure sources of randomness or trusted third parties [4, 32] such as the NIST random beacon, Random.org or Oraclize.it.

The revealed backdoor in the Dual Elliptic Curve PRNG [14], the unreliability of proprietary beacons [25], and the possibility of a centralized provider buffering, manipulating, and benefiting from prior knowledge of the provided randomness [25] are only a few of many reasons in favor of distributed randomness beacons where trust is spread among participants.

Considering distributed approaches, the following properties, as outlined in [6, 25, 114], are desiderata of a random beacon protocol:

1. **Availability/Liveness:**

   Any single participant or colluding adversary should not be able to prevent progress.

2. **Unpredictability:** Correct, as well as adversarial nodes, should not be able to predict (precompute) future random beacon values.

3. **Bias-Resistance:** Any single participant or colluding adversary should not be able to influence future random beacon values to their advantage.

4. **Public-Verifiability:** Third parties not directly participating in the protocol should also be able to verify generated values. As soon as a new random beacon value becomes available, all parties can verify the correctness of the new value using public information only.

Additionally, we follow the notion of [78, 38] where **guaranteed output delivery** (G.O.D.) [98] i.e., the inability for an adversary to prevent correct participants of the protocol from obtaining an output, is also considered as an important property of random beacon protocols. In particular, if an adversary is not sufficiently restricted by how much it can affect the timing of the random beacon's output in system models with bounded delays, both unpredictability and bias-resistance are weakened because the adversary can influence if an application is able to receive the output before a certain time or not.

Another particular desirable property for random beacons in the context of (permissionless) distributed ledgers is the **avoidance of an initial trusted setup**, e.g., a trusted dealer [114].

Current random beacon protocols aim to provide solutions by employing different techniques, reaching from Proof-of-Delay [32, 30] and incentive based solutions [29, 99] over publicly-verifiable secret sharing (PVSS) [6, 38, 78, 114] and unique signatures [42, 72] to utilizing Bitcoin itself as a source of randomness [12, 25]. The diversity of these approaches, as well as the differences in their underlying assumptions and characteristics, make them difficult to compare and not equally suited for all use-cases. Furthermore, various recently described (PoS) blockchain schemes utilize or provide a random beacon as part of their protocol design and are therefore not easily comparable or deployable as a stand-alone protocol.

2.1.1 Contribution

In this chapter we present *HydRand*, a PVSS based distributed random beacon protocol geared towards the *continuous* provision of randomness at regular intervals in a Byzantine failure setting. HydRand provides *guaranteed output delivery*, i.e., it guarantees the generation of new, *bias-resistant* randomness in every round of the protocol. As a hybrid approach, HydRand provides both a probabilistic guarantee for *unpredictability*, which ensures that a successful prediction of future random beacon values becomes exponentially unlikely, as well as unpredictability with absolute certainty for applications which wait for at least $f + 1$ rounds before using a future protocol output. The protocol assumes a synchronous system model and $n = 3f + 1$ participants. In respect to previous approaches based on PVSS, the communication complexity is hereby lowered from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$

as HydRand only requires at most one PVSS distribution/recovery operation per round. Our protocol is described in a self contained manner and neither relies on a trusted dealer nor on a distributed key generation (DKG) protocol.

Moreover, to the best of our knowledge, we are the first to provide an extensive comparison of state of the art random beacon protocols in this field that considers and analyzes a variety of key characteristics and assumptions.

2.1.2 Structure of the Remainder of this Chapter

The remainder of this chapter is structured as follows: The assumed system model is presented in Section 2.2. Section 2.3 provides a high level overview of the HydRand protocol and highlights the employed variant of PVSS, which constitutes one of the main cryptographic primitives that is utilized. Then, the protocol's operation is further illustrated by discussing an exemplary execution in Section 2.3.3 and the details of the protocol in Section 2.4. An extensive analysis including proofs showing that HydRand achieves the desired properties is presented in Section 2.5. Our evaluation results obtained from measurements of our prototypical implementation are described in Section 2.6. Section 2.7 compares HydRand to other related schemes, while Section 2.8 and Section 2.9 discuss and conclude the chapter. A quick reference for the utilized symbols and notations can be found in the Appendix 2.A at the end of the chapter.

2.2 System and Threat Model of the HydRand Protocol

We assume a fixed set of known participants, hereby referred to as *nodes*, of size $n = 3f+1$, of which at most f nodes may exhibit Byzantine failures and can deviate arbitrarily from the specified protocol. A node is considered to be *correct* if it does not engage in any incorrect behavior during the entirety of the protocol execution, else it is considered to be *faulty*. The terms *Byzantine* or *malicious* are used synonymously to refer to faulty nodes. The set of all nodes is denoted by $\mathcal{P} = \{1, 2, ..., n\}$ and each node $i \in \mathcal{P}$ is assumed to have a private/public key pair $\langle sk_i, pk_i \rangle$. The public keys of these keypairs are known to all participants. A synchronous system model with a fully connected network of authenticated and reliable bidirectional point-to-point messaging channels is assumed. We argue that the chosen timing model is reasonable for a small to moderate set of participants, and defer an analysis of our protocol in other system models to future work. Further, for many application areas of random beacons, e.g., in the context of cryptocurrencies, partially synchronous and synchronous system models are prevalent. Here, a synchronous random beacon protocol that also provides guaranteed output delivery may be necessary if strong notions of bias-resistance are a requirement.

2.3 Overview of the HydRand Protocol

The aim of HydRand is to provide a bias-resistant, publicly-verifiable and unpredictable stand-alone random beacon which emits random values at a regular interval. We target

HydRand at a permissioned setting with a fixed set of participants and assume a known upper bound[1] on both computation and message transmission times.

For the protocol setup it is assumed that all participants exchanged their public keys and prepared an initial commitment using publicly-verifiable secret sharing (PVSS). The protocol operation itself is separated into *rounds*, where each round consists of three distinct *phases* – propose, acknowledge and voting. We describe these phases in detail in Section 2.4. In each round, the previously generated random value is used for uniquely determining the current round *leader*. This leader has two choices: (i) The leader *reveals* the correct secret value he has committed himself to the last time[2] he was leader and attaches his next commitment. (ii) The leader does not reveal his secret value and therefore cannot attach another commitment. In the latter case, the previously committed secret value is *reconstructed* by $f + 1$ other nodes, including at least one correct participant. The properties of the underlying PVSS scheme ensure that the random beacon value obtained by reconstruction is always equal to the value that is obtained when a leader reveals his secret – this establishes *bias-resistance*. Additionally, *guaranteed output delivery* follows because the protocol outputs a random beacon value at each round, independent of the actions of the (potentially adversarial) leader and other faulty nodes.

In case the leader's previous commitment is reconstructed, the leader is excluded from being eligible as leader in future rounds since no new valid commitment was provided. However, the presented protocol could also be adapted to facilitate that temporarily failed nodes may rejoin $f + 1$ rounds after a fresh commitment is provided and agreed upon. A correct leader constructs a new *dataset*, which includes: (i) the secret value they previously committed themselves to, (ii) a new commitment to a uniformly random sampled value and (iii) a reference to the dataset of the previous round. The leader signs this dataset using their private key and broadcasts this message and signature to all other nodes in the network. After receiving and verifying the dataset, each node can compute the new random value of the beacon.

In case a leader is faulty and does not broadcast any data, other participants can collaborate to reconstruct the missing secret value, i.e. the value the leader has previously committed himself to in (ii). The reconstructed value can be used by any node to obtain the new random beacon value and thereby advance the protocol to the next round and leader. This process is repeated until eventually a correct leader is selected that creates a new dataset that accounts for all reconstructed datasets in between.

To ensure that a correct node is selected as leader after (at most) $f + 1$ rounds, all previously selected leaders of the last f rounds are exempt from becoming leader in the current round. Since malicious nodes are unable to determine how an unrevealed commitment of an honest leader will influence future random beacon values, they cannot

[1] We assume that a message sent at the beginning of one phase is received within that same phase.

[2] The initial commitment from the protocol setup is used the first time.

precompute any future output once a correct node is selected as leader. Moreover, correct participants converge on a single history after a correct node is selected as leader, because correct leaders are required to build on top of a single dataset and never sign different datasets in the same round. The correct node acts as a barrier for *unpredictability* and anchor for agreement on the protocol state. Unpredictability is thereby ensured with certainty for any round after $f + 1$ rounds in the future. *Public-verifiability* is established by leveraging the properties of the underlying PVSS scheme.

2.3.1 Publicly-Verifiable Secret Sharing

We rely on PVSS as a primary building block in the HydRand protocol. More specifically, we make use of Scrape's PVSS protocol [38], which is an optimization of Schoenmakers' PVSS scheme [108], and allows a node (dealer) to efficiently share a secret value $s \in \mathbb{Z}_q$ among a set of n recipients, such that any subset of at least t recipients is able to recover/reconstruct the value $h^s \in \mathbb{G}_q$, where h is one of two independent generators of the group $\mathbb{G}_q$ and the prime q denotes the order of this group. The value of the reconstruction threshold t is set in a way that does not enable a colluding adversary to successfully recover a shared secret without requiring the collaboration of at least one correct node, i.e. $t = f + 1$. A key property of a *publicly-verifiable* secret sharing protocol is that, upon receiving the secret shares, not only the recipients but any third party with access to the public keys of the participants can verify the correctness of the shares prior to reconstruction of the secret. We use the term *PVSS commitment*, denoted by $Com(s)$, to refer to the result of the share distribution process of Scrape's PVSS. To form a PVSS commitment, a dealer provides:

- The encrypted shares for a secret s, i.e. one encrypted share $\hat{s_i}$ for each node i, encrypted with the receiver's public key.
- The commitments $v_1, v_2, ..., v_n$ to the shares for each node.
- A non-interactive zero-knowledge (NIZK) proof ensuring the correctness of the encrypted shares

For additional details regarding Scrape we refer the reader to [38].

2.3.2 Design Rationale

A malicious leader can try to construct and send different commitments, and hence different datasets, to other participants of the protocol or selectively withhold information to bias the resulting sequence of random beacon values. Hence, some form of (Byzantine) *consensus protocol* is necessary for participants to agree either on a single, valid commitment or the fact that the leader was faulty. In this respect, HydRand leverages on its intended application as a *continuous* random beacon to amortize the communication overhead of Byzantine agreement (BA) that is incurred at each round. Specifically,

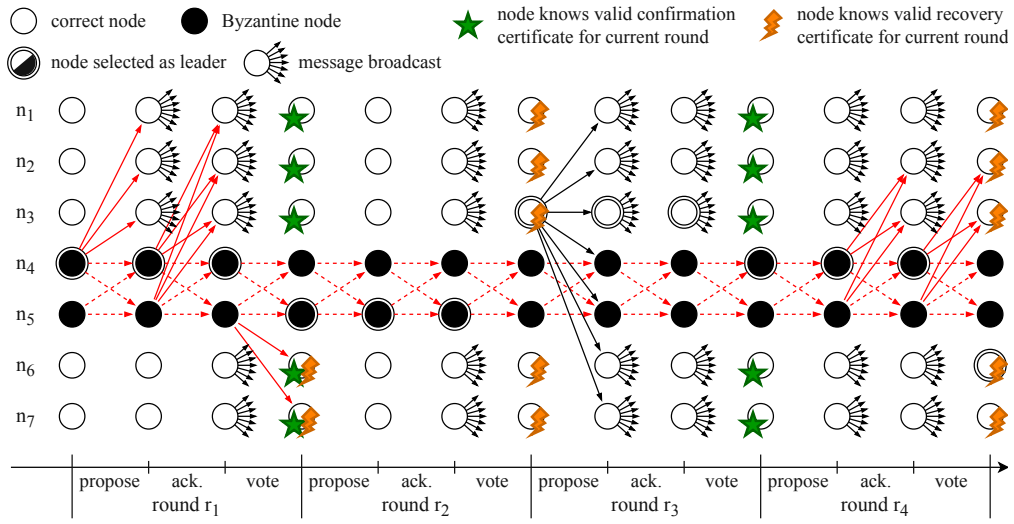

Figure 2.1: Example execution of four rounds of the HydRand protocol with $n = 3f+1 = 7$ nodes

HydRand introduces a variant of repeated Byzantine agreement that defers consensus decisions for up to $f+1$ rounds, and combines data from multiple consensus instances that are executed with every consecutive new round of the HydRand protocol. By exempting a current leader to be re-elected within the next f rounds, enough time is given to reach agreement if the leader was faulty or not. Thereby, the overall communication (bit) complexity in regard to PVSS based random beacon schemes with comparable guarantees is reduced from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

2.3.3 Example Protocol Execution

Figure 2.1 shows four rounds of an example execution of the HydRand protocol in a setting of $f = 2$ Byzantine nodes. The sequence of randomly selected leaders in this example execution includes a worst case scenario, where f distinct leaders were drawn from the set of Byzantine nodes (nodes n_4 and n_5), followed by a correct node and then again the first Byzantine node (n_4).

Round r_1: In this execution the first node that gets selected as the leader (i.e., node n_4) belongs to the set of Byzantine nodes. This leader selectively sends a *propose* message only to a subset of correct nodes. In our case the nodes n_1, n_2 and n_3. Moreover, the Byzantine node n_5 only sends *acknowledge* messages to the very same nodes (n_1, n_2, n_3). After that phase, the Byzantine node n_5 sends a *recover* message to the nodes n_6 and n_7.

This leads to a situation where the correct nodes n_1, n_2 and n_3 receive $2f+1$ acknowledge messages. Therefore, those nodes (n_1, n_2 and n_3) broadcast *confirm* messages which together form a valid confirmation certificate known to every node. Further, the nodes n_6 and n_7 as well as the adversary are in possession of a valid recovery certificate $RC(r_1)$,

as nodes n_5, n_6 and n_7 sent out *recover* messages.

Round r_2: The next node (n_5) that is selected as leader is also in the set of Byzantine nodes and does not broadcast any message. Therefore, the secret value of the rounds leader gets reconstructed at the end of the *vote* phase and all nodes are only in possession of a *reconstruction certificate* $RC(r_2)$ for this round.

Round r_3: The leader (n_3) of this round belongs to the set of correct nodes and has received $f+1$ *confirm* messages in round r_1. Moreover, node n_3 is not in possession of a valid recovery certificate for r_1 since he has only received f *recover* messages, i.e. from node n_6 and n_7 but not from node n_5. Therefore, the leader broadcasts a new dataset D_3 containing a valid *confirmation certificate* $CC(D_1)$ for round r_1, as well as a *recovery certificate* $RC(r_2)$ for round r_2.

After receiving the *propose* message, all correct nodes, including n_6 and n_7, are safe to assume that at least $f+1$ correct nodes are in possession of dataset D_1. The justification for this assumption comes from the fact that the *propose* message contains a *confirmation certificate* composed of $f+1$ signed messages including the hash $H(D_1)$ of D_1. This necessarily includes at least one honest node which, per definition, only sends a confirm message if it has received $2f+1$ valid *acknowledge* messages in advance. Therefore, at least $f+1$ correct nodes have to be in possession of dataset D_1. As a result, all correct nodes accept this rounds new dataset D_3 containing $CC(D_1)$. This holds true, even for nodes n_6 and n_7 although they have not received dataset D_1.

If node n_6 or n_7 would have been selected as leader in round r_3, then this node would have constructed a dataset D_3 that contains a valid *recovery certificate* for round r_1 and r_2 as well. In that case the nodes n_1, n_2 and n_3 would have discarded their dataset D_1.

Round r_4: In this round node n_4 is again selected as leader. This is valid since f rounds have passed since this node has been selected as leader. Therefore, at least one correct node was selected as leader in between – in this case node n_3. Since there is no *recovery certificate* $RC(r_3)$ for round r_3 available, all further leaders have to include the *confirmation certificate* $CC(D_3)$ for round r_3 to extend upon the chain of valid datasets. Otherwise their future datasets would not be valid and rejected by all correct nodes. Therefore, all nodes including node n_4, have to accept the view of node n_3 in this case.

In our example, node n_4 attempts to stall the protocol by selectively releasing a new dataset D_4 only to the nodes n_2, n_3. But since those nodes are not able to reach the required number of $2f+1$ *acknowledge* messages (together with the Byzantine nodes n_4 and n_5), no correct node will send a *confirmation* message in the last phase of this round. As a result all correct nodes will send *reconstruct* messages leading to a total of $2f+1$ reconstruct messages, which is more than $f+1$ and hence enough to form a *reconstruction certificate* and to reconstruct the leader's secret for round r_4 given the decrypted shares of n_1, n_2 and n_3.

Note that, although possible, the PVSS reconstruction of the secret from r_1 would not be necessary here, since in this example the leader of r_4 selectively sent out a new dataset

and therefore revealed the secret to at least one correct node, namely n_2 and n_3. Per definition, correct nodes broadcast the revealed secret in their *acknowledge* messages. Therefore, all other correct nodes receive the revealed secret in round r_4 even if they have not received the dataset D_3 directly.

2.4 The HydRand Protocol

HydRand proceeds in rounds, where each round $r \geq 1$ consists of three phases: *propose*, *acknowledge* and *vote*. Further, each round has a uniquely associated leader $\ell_r \in \mathcal{P}$ that is selected through the randomness generated by the protocol. When referring to the current round's leader, we may omit the subscript and simply denote the leader by ℓ.

Each round, ℓ_r is selected uniformly at random from the set of all nodes that *were not leader* during the last $f+1$ rounds[3]. At the end of a round all nodes learn a new random beacon value R_r. For simplicity, we hereby assume that correct nodes agree on the initial random beacon value R_0 used to select the leader of round 1, as well as the set of initial commitments of all nodes. R_0 becomes public knowledge only after the set of initial commitments was defined during setup.[4]

To simplify our notation, we assume that the sender of a broadcast is also a recipient of that message. Similarly, the dealer in the PVSS protocol also provides a share for himself. We use $\langle m \rangle_i$ to denote the message m a node i cryptographically signed with its private key sk_i. We further assume, that each correct node discards invalidly signed messages and processes only messages for the current round and phase.

2.4.1 Propose Phase

During this phase the round leader ℓ reveals his previously committed value s_ℓ and provides a new commitment $Com(s_\ell^\star)$. For this purpose, it is the leader's task to propose a new dataset D_r for the current round r. As a performance optimization, we split a dataset into two parts: a header and a body. For certain operations, we only require sending the header of the dataset. The header $header(D_r)$ of dataset D_r contains:

- the hash of the dataset's body $H(body(D_r))$
- the current round index r
- the round's random beacon value R_r
- the revealed secret value s_ℓ

[3]The detailed leader selection mechanism is described in Section 2.4.4.

[4]In practice this initial random value can, for example, be obtained via *Proof-of-Delay* [32] or a *Proof-of-Work* [12].

- the round index $\overleftarrow{r}$ of the previous dataset $D_{\overleftarrow{r}}$
- the hash $H(D_{\overleftarrow{r}})$ of the previous dataset $D_{\overleftarrow{r}}$ if $\overleftarrow{r} > 0$
- a list of random beacon values $\{R_k, R_{k+1}, ...\}$ for all recovered rounds between $\overleftarrow{r}$ and r (if any)
- the Merkle tree root hash M_r over all encrypted shares in the new commitment $Com(s^\star_\ell)$

We use $H(D_r) = H(header(D_r))$ to denote the cryptographic hash of the dataset D_r. The body $body(D_r)$ of dataset D_r contains:

- a confirmation certificate $CC(D_{\overleftarrow{r}})$, which confirms that $D_{\overleftarrow{r}}$ was previously accepted as a valid dataset
- a recovery certificate $RC(k)$ for all rounds $k \in \{\overleftarrow{r}+1, \overleftarrow{r}+2, ..., r-1\}$, which confirms that there exists a recovery for all rounds between $\overleftarrow{r}$ and r If $\overleftarrow{r} = r-1$ then no such intermediate round exists and this value is omitted.
- the commitment $Com(s^\star_\ell)$ to a new randomly chosen secret $s^\star_\ell$

The leader selects $\overleftarrow{r} < r$ as the most recent regular round, for which the leader is not aware of any successful recovery. As we prove in Section 2.5.1, such a round always exists and the leader is in possession of the confirmation certificate $CC(D_{\overleftarrow{r}})$ required for the dataset's body.

After the construction of the above dataset, a correct leader ℓ broadcasts a signed *propose* message $\langle propose, \langle header(D_r)\rangle_\ell, body(D_r)\rangle_\ell$ to all nodes. Each node i, which receives such a message from the leader before the end of the propose phase, checks the validity of the dataset D_r. For this purpose i verifies that D_r is constructed as previously defined and properly signed. This includes a check that the revealed secret s_ℓ corresponds to the previous commitment $Com(s_\ell)$ of the current leader. Additionally, the validity of the confirmation and recovery certificates is checked. A *confirmation certificate* $CC(D_{\overleftarrow{r}})$ for dataset $D_{\overleftarrow{r}}$ is valid iff it consists of $f+1$ signed messages of the form $\langle confirm, \overleftarrow{r}, H(D_{\overleftarrow{r}})\rangle_i$ from $f+1$ different senders i. Similarly, a *recovery certificate* $RC(k)$ for some round k is a collection of $f+1$ signed messages of the form $\langle recover, k\rangle_i$ from $f+1$ different senders.

2.4.2 Acknowledge Phase

If a node i receives a valid dataset D_r from the round's leader ℓ during the propose phase, a signed acknowledge message $\langle\langle acknowledge, r, H(D_r)\rangle_i, \langle header(D_r)\rangle_\ell\rangle_i$ is constructed and broadcasted, thereby the node also forwards the revealed secret value s_ℓ as part of the header. Furthermore, each node i collects and validates acknowledge messages from other nodes.

2.4.3 Vote Phase

Each node i checks the following conditions:

- During the current propose phase a valid dataset D_r was received.
- During the current acknowledge phase at least $2f + 1$ valid acknowledge messages from different senders have been received.
- All acknowledge messages received refer to the dataset's hash $H(D_r)$. Valid acknowledge messages for more than one value of $H(D_r)$ form a cryptographic proof of leader equivocation.[5]

If all conditions are met, node i broadcasts a signed confirmation message of the form $\langle confirm, r, H(D_r)\rangle_i$. Otherwise node i, broadcasts a corresponding recover message $\langle\langle recover, r\rangle_i, s_\ell, Com(s_\ell)[s_i], \hat{s}_i, M_k[\hat{s}_i], R_{r-1}\rangle_i$. Here, $Com(s_\ell)[s_i]$ denotes i's decrypted share s_i and its share decryption proof according to Scrape's PVSS, which cryptographically proves that s_i is a valid decryption of $\hat{s}_i$ under i's secret key. Round k denotes the round in which ℓ has provided the commitment $Com(s_\ell)$ and a Merkle tree root hash M_k. The Merkle branch $M_k[\hat{s}_i]$ proves that the encrypted share $\hat{s}_i$ was previously distributed as part of $Com(s_\ell)$ and therefore also of D_k. The values $\hat{s}_i$ and $M_k[\hat{s}_i]$ are required to enable nodes which are not in possession of $Com(s_\ell)$ to verify the share decryption proof for s_i. R_{r-1} is included for efficient external verification.

Correct nodes always include values for s_ℓ, $Com(s_\ell)[s_i], \hat{s}_i$ and $M_k[\hat{s}_i]$ if they are in possession of the required data. Otherwise the unknown value(s) are omitted. This can happen if an adversary selectively sent the previous dataset D_k to a subset of nodes. Therefore, upon receiving recovery messages from other nodes, correct nodes accept messages with omitted values. The protocol guarantees that at least $f + 1$ correct nodes have received the dataset with a valid confirmation certificate, and hence are able to provide the necessary shares required for reconstructing the respective secret. An example is presented in Section 2.3.3.

At the end of this phase each node i can obtain the round's random beacon value R_r. We distinguish between the following two cases: (i) node i already knows the secret value s_ℓ, because it received the dataset D_r or an acknowledge message for D_r, and (ii) node i has received at least $f + 1$ valid recover messages which include at least $f + 1$ decrypted secret shares for s_ℓ. In the latter case the reconstruction procedure of Scrape's PVSS can be executed to produce the value h^{s_ℓ}. In both cases R_r is then obtained by computing:

$$R_r \leftarrow H(R_{r-1} \,||\, h^{s_\ell}) \qquad \text{(Definition 1)}$$

[5] In a (PoS) cryptocurrency setting, the protocol could be extended such that this equivocation proof is used to seize some form of security deposit from the leader.

2.4.4 Leader Selection

At the beginning of each round $r \geq 1$, a node i determines the round's leader ℓ_r based on the available local information it gathered so far. For this purpose node i uses the randomness R_{r-1} of the previous round to deterministically select ℓ_r from the set $\mathcal{L}_r$ of potential leaders. We denote the canonical representation of $\mathcal{L}_r$ as $\langle l_0, l_1, ..., l_{|\mathcal{L}_r|-1} \rangle$ and obtain ℓ_r as follows:

$$\ell_r \leftarrow l_{(R_{r-1} \bmod |\mathcal{L}_r|)} \qquad \text{(Definition 2)}$$

Let $D_{\bar{r}}$ denote the most recent valid dataset, for which node i is *not* in possession of a corresponding recovery certificate $RC(\bar{r})$. If no such dataset exists we set $\bar{r} = 0$. Now we introduce a method to determine the set of *recovered nodes* $rn(\cdot)$ as a component needed for the definition of $\mathcal{L}_r$. Intuitively, the set $rn(\cdot)$ contains all nodes, which have not provided valid datasets for some round where the node was selected as leader. We define the set of all leaders that were recovered in some round up to a referenced dataset as follows:

$$rn(D_x) = \begin{cases} \emptyset & \text{if } x = 0 \\ \{\ell_k \mid RC(k) \in D_x\} \cup rn(D_{\bar{x}}) & \text{otherwise} \end{cases} \qquad \text{(Definition 3)}$$

Here $D_{\bar{x}}$ denotes the previous dataset referenced by D_x. This function is used to construct the set of available nodes $\mathcal{P}_r$ for round r recursively by excluding all nodes which have been selected as leader in a round for which a valid reconstruction certificate exists:

$$\mathcal{P}_r = \mathcal{P} \setminus rn(D_{\bar{r}}) \qquad \text{(Definition 4)}$$

Based on this notion, the definition of the set of potential leaders $\mathcal{L}_r$ for round r follows:

$$\mathcal{L}_r = \mathcal{P}_r \setminus \{\ell_{r-f}, \ell_{r-f+1}, ..., \ell_{r-1}\} \qquad \text{(Definition 5)}$$

Intuitively, the set $\mathcal{L}_r$ only includes nodes that were not selected as leader for at least f rounds in the past and have not been reconstructed in any previous round, i.e., nodes that distributed valid datasets for all rounds in which they were selected as leader.

2.5 Analysis of HydRand's Protocol Properties

In the following, we show that HydRand achieves the desirable properties of a random beacon protocol as outlined in Section 2.1: *liveness*, *guaranteed output delivery*, *unpredictability*, *bias-resistance*, and *public-verifiability*. We furthermore show that our protocol also achieves *uniform agreement*. In our proofs we may refer to the definitions introduced in Section 2.4.

2.5.1 Liveness and Guaranteed Output Delivery

To show that HydRand satisfies liveness and guaranteed output delivery, we first introduce and prove several primary lemmas. We show that (i) correct nodes are always able to provide a valid dataset if they are selected as leader, (ii) correct nodes can never be recovered and (iii) the set of potential leaders always contains at least $f+1$ correct nodes. Using these results, we infer that correct nodes can always output the round's random beacon value by the end of the round, given that they know the value for the previous round. Finally, we use an inductive argument to prove liveness and guaranteed output delivery of our protocol.

Lemma 1. *(Possibility of construction of valid datasets) For each round r a correct leader ℓ_r can construct a valid dataset D_r.*

Proof. Implicit agreement by all correct nodes on the current round number r follows from the synchronous system model and fixed duration of phases. A correct leader is in possession of its own secret s_ℓ and thus knows R_r. Furthermore, the leader can always construct a new PVSS commitment for a new secret $Com(s^\star_\ell)$ and is able to provide a valid value for M_r. Therefore, it only remains to be shown that each correct node is able to provide the required confirmation certificate $CC(\cdot)$ and recovery certificates $RC(\cdot)$. During the vote phase of every previous round, correct nodes have either broadcast a recover or confirm message. As there are at least $2f+1$ correct nodes, each node is guaranteed to receive at least $f+1$ recover messages or at least $f+1$ confirm messages (or both) for each of these rounds. As $f+1$ recover messages form a recovery certificate and $f+1$ confirm messages form a confirmation certificate, each node is in possession of a recovery certificate or a confirmation certificate (or both) for every previous round, and is hence able to provide the required certificates for D_r. □

Lemma 2. *(No recovery of correct leaders) If leader ℓ_r is correct, there does not exist a node i, which is in possession of a valid recovery certificate $RC(r)$.*

Proof. A correct leader ℓ_r sends valid proposal D_r to all nodes during the propose phase. By Lemma 1, ℓ_r can always construct such a dataset. As all correct nodes consider D_r as valid, at least $2f+1$ nodes broadcast acknowledge messages for D_r during the acknowledge phase. All $2f+1$ correct nodes therefore receive at least $2f+1$ valid acknowledge messages for D_r. Since there cannot exist a valid acknowledge for a different dataset D'_r (a correct leader only provides his signature for D_r) all correct nodes broadcast *confirm* messages during the vote phase. As correct nodes only broadcast either confirm or recover messages, there are at most f recover messages (from Byzantine nodes). A valid recovery certificate $RC(r)$ however requires at least $f+1$ recover messages from different nodes, and therefore cannot exist. □

Lemma 3. *(Availability of leaders) For each round $r \geq 1$, the set of potential leaders $\mathcal{L}_r$ contains at least $f+1$ correct nodes.*

Proof. We first show that for each round r, the set of available nodes $\mathcal{P}_r$ contains at least $2f+1$ correct nodes. By Definition 3 and Definition 4 (see Section 2.4.4), we ensure that only leaders ℓ_k for some round k, in which a recovery certificate $RC(k)$ exists, are excluded from the set $\mathcal{P}$ to form $\mathcal{P}_r$. As we have shown in Lemma 2 there are no recovery certificates for rounds with correct leaders. Therefore correct nodes cannot be excluded from $\mathcal{P}$ to form $\mathcal{P}_r$, and thus $\mathcal{P}_r$ contains at least $2f+1$ correct nodes.

Using the above result and Definition 5, which excludes at most $f+1$ nodes from $\mathcal{P}_r$ to form $\mathcal{L}_r$, $\mathcal{L}_r$ contains at least $f+1$ correct nodes. □

Lemma 4. *(Liveness) If a correct node knows the random beacon value R_{r-1}, it can output the random beacon value R_r by the end of round r (independent of the actions of the round's leader ℓ_r).*

Proof. Following Lemma 3 we guarantee the existence of a leader ℓ_r. Since $\ell_r \in \mathcal{L}_r$ and $\mathcal{L}_r \subset \mathcal{P}_r$, we know that $\ell_r \in \mathcal{P}_r$. By applying Definition 4 we get $\ell_r \notin rn(D_{\overleftarrow{r}})$. Hence, there exists some history of datasets with head $D_{\overleftarrow{r}}$, in which there does not exist a recovery certificate $RC(k)$ for any round $k < \overleftarrow{r}$ in which ℓ_r was also leader. Such a history for any valid dataset D_k can only exist if at least one correct node confirmed that D_k was correctly distributed and acknowledged by $2f+1$ nodes by providing a confirm message. Hence, at least $f+1$ correct nodes know a common dataset D_k for all rounds k where ℓ_r was previously selected as leader. In addition, all nodes know the shares for ℓ_r's first commitment as defined in the protocol setup. Thus, at least $f+1$ correct nodes can broadcast the decrypted share in case a recovery of the leader ℓ_r in round r is necessary. Hence all nodes learn the value h^{s_ℓ} corresponding to ℓ_r's last commitment $Com(s_\ell)$, and thus obtain R_r using h^{s_ℓ} and R_{r-1} via Definition 1. □

Theorem 1. *(Guaranteed Output Delivery) For each round r all correct nodes output a new random beacon value R_r.*

Proof. We use lemmas 3 and 4 and prove the theorem by induction on the round index r. For the base case we have an agreed random beacon value R_0 as given by the protocol setup. For the induction step, we assume that R_{r-1} is known by all correct nodes. Lemma 3 ensures that the set of potential leaders $\mathcal{L}_r$ contains at least $f+1$ correct nodes. Therefore, Definition 2 can always be applied to a selected leader ℓ_r using $\mathcal{L}_r$ and R_{r-1}. Hence, we can use Lemma 4, to show that by the end of round r each correct node outputs a value R_r. □

2.5.2 Agreement

In the following, we show that all correct nodes agree on a common sequence of random beacon values. We start by showing that (i) within $f+1$ rounds a correct node is selected as leader and (ii) all correct nodes agree on a common set of potential leaders and use this two results to prove that uniform agreement is satisfied for the random beacon values in HydRand.

Lemma 5. *(Selection of correct leaders) In each interval* $\{k, k+1, k+2, ..., k+f\}$ *of* $f+1$ *consecutive rounds there is at least one round* $\bar{k} \in \{k, k+1, k+2, ..., k+f\}$*, such that the leader* $\ell_{\bar{k}}$ *of that round is correct.*

Proof. We assume that there is no correct leader in $\{\ell_k, \ell_{k+1}, \ell_{k+2}, ..., \ell_{k+f}\}$ and derive a contradiction. We apply the definition of the set of potential leaders for round $k+f$:

$$\mathcal{L}_{k+f} = \mathcal{P}_{k+f} \setminus \{\ell_k, \ell_{k+1}, ..., \ell_{k+f-1}\}$$

Notice that $\{\ell_k, \ell_{k+1}, ..., \ell_{k+f-1}\}$ denotes a set of f Byzantine nodes. As there are only f Byzantine nodes in total, $\mathcal{L}_{k+f}$ cannot contain any Byzantine nodes. However, the Byzantine node ℓ_{k+f} is assumed to be leader of round $k+f$ and therefore $\ell_{k+f} \in \mathcal{L}_{k+f}$, which completes the contradiction. □

Lemma 6. *(Agreement on potential leaders) If a node constructs a valid set of potential leaders* $\mathcal{L}_r$ *in round* r*, then every correct node constructs the same value for* $\mathcal{L}_r$*.*

Proof. Using Lemma 5, for the interval $\{r-f-1, r-f, ..., r-1\}$, we know that there is some round $\bar{r}$ with a correct leader $\ell_{\bar{r}}$ in this interval. Using Lemma 1, we know that $\ell_{\bar{r}}$ is able to construct a valid dataset $D_{\bar{r}}$ in round $\bar{r}$. As $\ell_{\bar{r}}$ is correct, it has distributed this dataset to all nodes during the propose phase of round $\bar{r}$. All correct nodes therefore acknowledge $D_{\bar{r}}$ in the acknowledge phase of round $\bar{r}$. Since there are at least $2f+1$ correct nodes, all correct nodes receive at least $2f+1$ valid acknowledge messages for $D_{\bar{r}}$ by the end of the acknowledge phase. No node can receive a valid acknowledge for some different dataset $D'_{\bar{r}}$, because the correct leader $\ell_{\bar{r}}$ does not provide a signature for a different value. Therefore, all correct nodes broadcast confirm messages for $D_{\bar{r}}$. As all correct nodes broadcast either one confirm or one recovery message, there are at most f recover messages (by Byzantine nodes). Therefore, no valid recovery certificate $RC(\bar{r})$ exists for round $\bar{r}$. Thus, any valid future dataset needs to (indirectly) reference the common and unique dataset $D_{\bar{r}}$. Consequently, we established agreement on $D_{\bar{r}}$ and its common history provided by the references to the predecessor datasets.

As the set of available nodes $\mathcal{P}_{\bar{r}}$ for round $\bar{r}$ is defined using only the agreed set of all nodes $\mathcal{P}$ and $D_{\bar{r}}$, $\mathcal{P}_{\bar{r}}$ is also agreed upon. Since the definition of $\mathcal{L}_r$ does not depended on whether or not leaders are recovered during the rounds $\{r-f, r-f+1, ..., r-1\}$ and $\bar{r} \geq r-f-1$, agreement on the set $\mathcal{L}_r$ follows. □

Theorem 2. *(Uniform Agreement) If a node outputs a valid random beacon value* R_r *in round* r*, then every node that outputs a valid beacon value in round* r *outputs the same* R_r*.*

Proof. We prove the theorem by induction on the round index r. For the base case we have an agreed common random beacon value R_0 as defined by the protocol setup.

For the induction step, we assume that every node that outputs a valid beacon value in round $r-1$ outputs the same R_{r-1}. We have agreement on R_{r-1} by the induction

hypothesis and show agreement on the set of potential leaders $\mathcal{L}_r$ in Lemma 6. As the leader selection mechanism given in Definition 2 only depends on those two arguments, all correct nodes agree on a common unique leader ℓ_r. By applying Lemma 4 we obtain that each correct node learns the leader's previously committed secret h^{s_ℓ}. By either checking the revealed value of s_ℓ against the leaders commitment or verifying the validity of the share decryption proof according to Scrape's PVSS description [38], uniqueness of a valid h^{s_ℓ} and consequently of R_r is ensured. □

2.5.3 Unpredictability

Intuitively, the prediction of a future random beacon value by the adversary is only possible if the adversary is selected as leader for that particular round, as well as all rounds before that point, because each round's random beacon value depends on a secret value only known to the round leader. As we prove below, this is impossible for $f + 1$ consecutive rounds.

However, even before this bound is reached, the possibility of successful prediction decreases exponentially in the number of rounds to predict. The probability of successful prediction of ω future random beacon values, where $\omega < f + 1$, can be characterized by a hypergeometric distribution with population size n, ω draws (the number of values to predict) and f success states (adversarial nodes) in the population. The prediction is possible, iff all of the ω draws pick one of the success states. Figure 2.2 shows the probabilities for different values of n, under the $n = 3f + 1$ security assumption. For large values of n, the probability converges to a geometric distribution.

Theorem 3. *(Unpredictability) At the beginning of round r, no node can predict the outcome R_{r+f} of the random beacon protocol in round $r + f$.*

Proof. By applying Lemma 5 we show that there is at least one correct leader during the interval of $f + 1$ consecutive rounds $\{r, r + 1, r + 2, ..., r + f\}$. Let k denote any round during this interval in which the leader ℓ_k is correct. As ℓ_k follows the protocol, it has not distributed its secret value s_{ℓ_k} to any node at the beginning of round r. Additionally, no correct node will provide a decrypted secret share, which could be used in the recovery process of the secret value. Therefore only f secret shares are available for an adversary to try and recover the secret in order to compute R_k (and potentially consecutive random beacon values). However, the protocol defines the reconstruction threshold t used by the PVSS scheme to be $f + 1$. Therefore, an adversary cannot obtain the underlying secret before it is revealed or recovered during round k. Consequently, R_k and all consecutive random beacon values (including R_{r+f}) are unpredictable at the start of round r. □

2.5.4 Bias-Resistance

Theorem 4. *(Bias-Resistance) No node i can, for any round r, influence the value R_r of the random beacon protocol in a meaningful (i.e. predictable) way.*

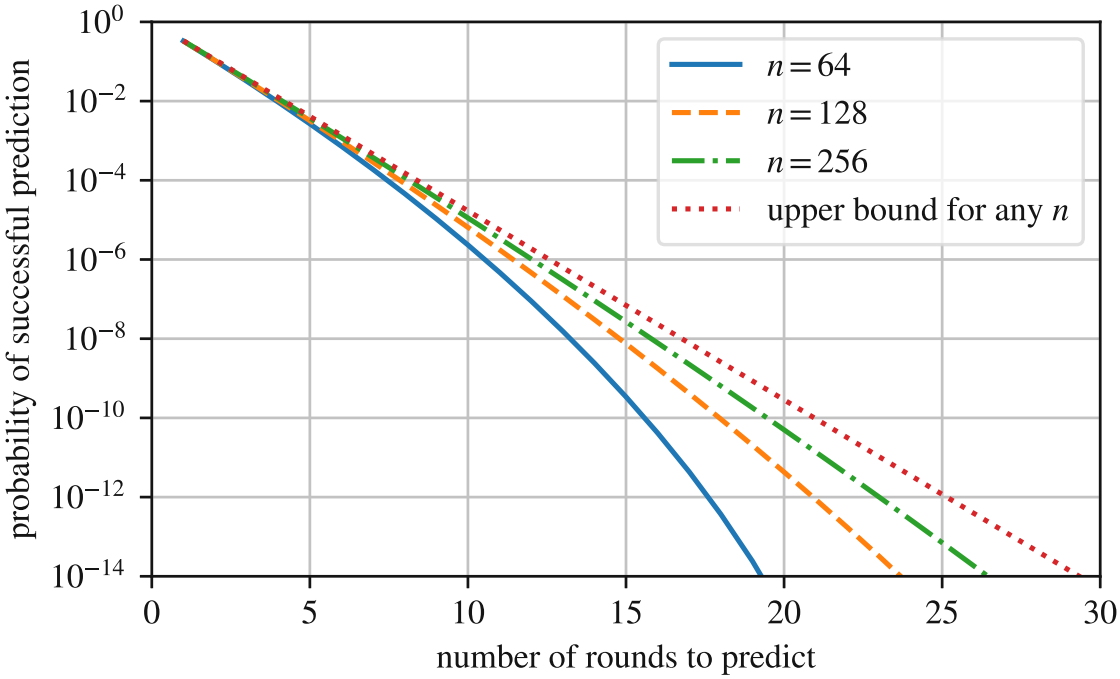

Figure 2.2: HydRand's unpredictability guarantees for different numbers of nodes (n), assuming a 33% adversary, i.e. $f = \lceil \frac{n}{3} \rceil$ - 1

Proof. This property follows from unpredictability and the fact that the protocol is constructed in a way that ensures that any action a (Byzantine) nodes takes in some round r, can only influence the value of the random beacon at round $r + f + 1$ or later. In Theorem 3 we have shown that the random beacon value at round $r + f$ is unpredictable at the beginning of round r. Therefore, a node cannot influence the random beacon values for rounds r to $r + f$, and may only influence values at round $r + f + 1$ or later in an unpredictable manner. □

2.5.5 Public-Verifiability

Theorem 5. *(Public-Verifiability) For each round r, an external verifier can check the correctness of the random beacon value R_r, at the end of round r.*

Proof. The external verifier receives from any correct node (i.e. after querying at most $f + 1$ nodes) its history up to and including round r. The verifier can, by following the protocol rules, only obtain a random beacon value R_r iff the provided data is correct. Additionally, an external verifier can obtain and verify recovered random beacon values between the last valid dataset $D_{\bar{r}}$ and the current dataset D_r for all rounds $k \in \{\bar{r} + 1, \bar{r} + 2, ..., r - 1\}$. □

Lemma 7. *(Efficient-Verification) For each round r, an external verifier can check the correctness of the random beacon value R_r in $\mathcal{O}(n)$, at the end of round r (without validation of all previous rounds).*

Proof. We distinguish two cases: (i) the leader of round r provided a valid dataset D_r in time. The confirmation certificate $CC(D_r)$ is hereby available at the end of round r and can be used to verify the correctness of D_r (and hence the included R_r) by verifying $f+1$ signatures; (ii) the leader of round r was recovered. In this case, an external verifier requests the necessary information to simulate a node's execution of the recovery step of round r, i.e. $f+1$ recover messages from round r as well as the header $header(D_{r'})$ and confirmation certificate $CC(D_{r'})$ of the failed leader's previously distributed dataset $D_{r'}$. The simulation of a node's recovery requires $f+1$ validations of decrypted PVSS shares, combining the shares via Lagrange interpolation, and checking $f+1$ signatures to verify $CC(D_{r'})$, and can thus be performed in $\mathcal{O}(n)$. □

2.6 Evaluation of the HydRand Protocol

To outline the feasibility and practicability of HydRand, we develop a fully functional protocol prototype in Python, and make our code publicly available on Github [107]. The evaluation was performed by executing the HydRand protocol on Amazon EC2 *t2.micro* instances (1 GiB of RAM, one virtual CPU core, 60-80 Mbit/s network bandwidth). To simulate an execution over the internet, instances were spread equally over multiple data centers in eight AWS regions (Canada, London, Ireland, N. California, N. Virginia, Paris, Singapore and Tokio).

Executions were performed both, with correct nodes only, as well as considering up to f simulated node failures. The synchronous round duration was derived experimentally and leaves room for improvement, as both the resource and network capacity of the instances can be adjusted, and the protocol code may be further optimized. Figure 2.3 presents the throughput our protocol achieves within the aforementioned setup conditions for different numbers of nodes (n) where failures can occur.

Figure 2.4 outlines the average bandwidth used per node over the duration of 30 minutes of protocol execution, with and without simulated failures. It should be noted that the protocol parametrization, specifically the round duration, is the same only for executions with the same n, i.e. for faulty and correct executions for a particular n, and corresponds to that of Figure 2.3. In the presented data it appears executions with simulated failures grow linearly with the size of n, whereas the average amount of data transferred per node in failure-free runs appears almost constant for different sizes of n. This is expected, given that executions with simulated failures induce larger message exchanges between participants if a leader has to be recovered and therefore consume more bandwidth than runs without failures.

In regard to verification performance, we measure that an (external) client can publicly verify the correctness of a round's random beacon in ≈ 57ms, considering the worst case

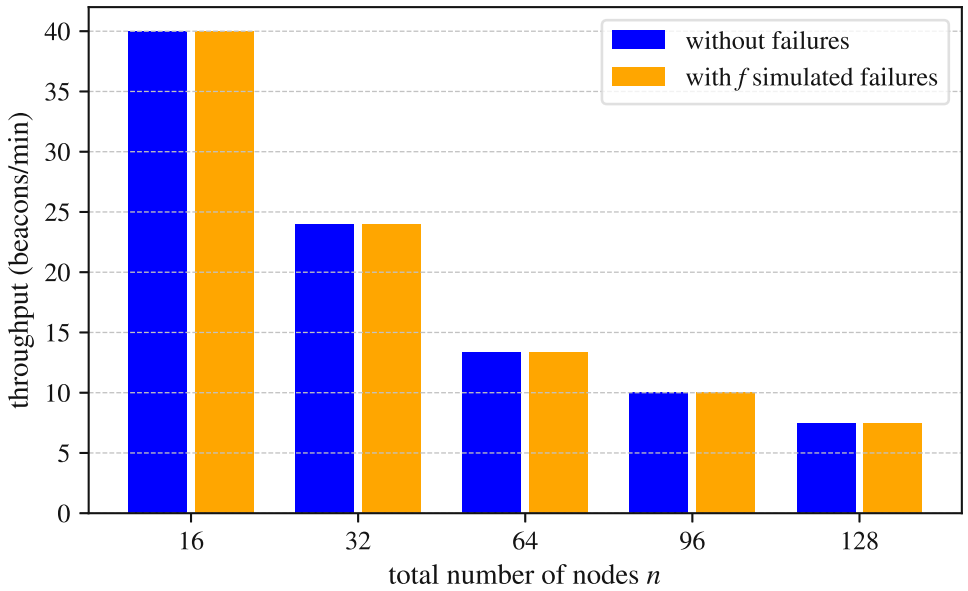

Figure 2.3: Measured throughput for the HydRand protocol, considering different numbers of nodes (n) with and without simulated failures

in a setting with with 128 nodes. The corresponding proof, which enables non-interactive verification, is $\approx$ 26kB in size.

The results of our research prototype evaluation highlight that the presented HydRand protocol is practicable for realistic deployment scenarios. Data from our performed executions suggests that the beacon throughput for large n was restricted by the computational capacity of the AWS instances. An evaluation of the effects of different parameterizations, including the utilization of more powerful instances, as well as an analysis of resource consumption under more complex adversarial behavior, is deferred to future work.

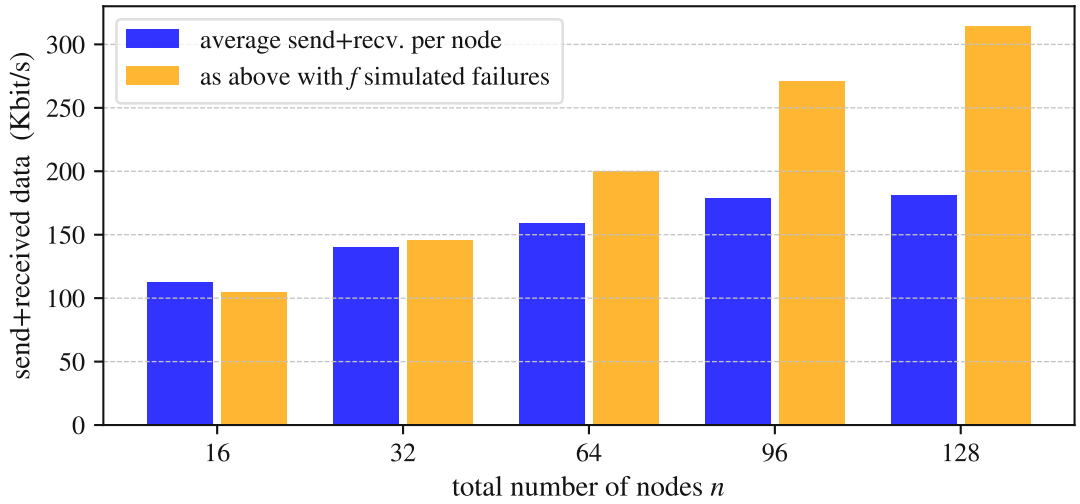

Figure 2.4: Measured average per node network bandwidth for the HydRand protocol, considering different total number of nodes (n), with and without simulated failures

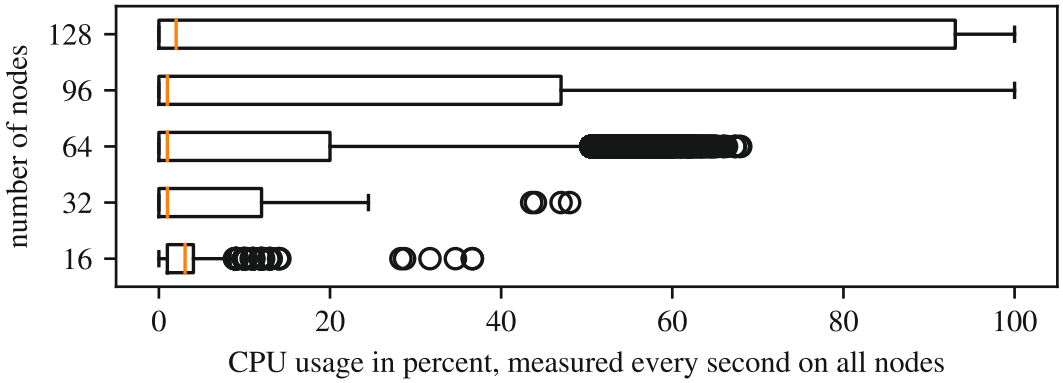

Figure 2.5: Total CPU utilization (in %), measured once every second on all nodes for different total number of nodes (n)

2.7 Comparison of Random Beacon Protocols

Recent years have seen a substantial amount of new research related to the generation of publicly-verifiable (distributed) randomness being published in academia as well as the industry. Thereof, we distinguish between the following types of protocols:

1. Stand-alone protocols, that are specifically designed to provide randomness. This includes the approach described within the first Ouroboros Proof-of-Stake protocol [78], the Scrape protocol [38], the Rand* protocol family [114], as well as our HydRand protocol.

2. Protocols designed for the purpose of generating randomness, leveraging resources of existing systems, namely Caucus [6] and Proof-of-Delay [32, 30].

3. Protocols that produce randomness as a byproduct of their operation, including Algorand [42], the BA protocol by Cachin et al., Dfinity [72] and Ouroboros Praos [46].

Additionally, we include Proof-of-Work blockchains, as first described by S. Nakamoto [86], as a source of public-verifiable randomness [25] in our comparison.

Proof-of-Work and Proof-of-Delay inherently require a substantial amount of computational resources to ensure security. When directly relying on the block hashes of a PoW blockchain as a source of randomness, bias-resistance can generally not be ensured. A miner can, with non-negligible probability, pick/reject random beacon values which suit him by choosing to withhold a valid PoW solution in favor of some other block(s). Hence, random beacon values derived by this mechanism may not be guaranteed to be uniformly distributed.

Proof-of-Delay, as described by Bünz et al. [32], addresses this problem by employing a delay function on top of the PoW block hash. Here, a cryptographic hash function or symmetric encryption algorithm is applied iteratively to the block hash to produce the randomness. The number of iterations Δ is a security parameter of the protocol and must be selected in a way that ensures that no adversary can finish this inherently sequential computation within the typical confirmation time of a block. While the adversary can still withhold its value and influence the protocol's output, they can only do so blindly without knowing the effects at the time of the decision, which ensures bias-resistance. However, full verification of a random beacon value is slow, as it requires the same sequential recomputing.

Algorand [42], Ouroboros Praos [46] and Caucus [6] are comparable in their approach of combining the previous public randomness with a (verifiable) source of private randomness, i.e., in the form of a VRF or hash chain, from an eligible leader to form the next random value. However, leader uniqueness by itself is not guaranteed and additional consensus rules are necessary to reach agreement. In this respect Algorand implements a Byzantine agreement protocol with finality, whereas Ouroboros Praos is a Proof-of-Stake blockchain protocol with eventual agreement, and Caucus is implemented within a smart contract that leverages the consensus protocol provided by the underlying Ethereum blockchain.

Cachin et al. [33] and Dfinity [72] both employ *unique* threshold signatures as a core primitive in their construction. The BLS signature scheme of Boneh et al. [24, 23], is a suitable candidate as its signatures are unique and both the signing process as well as the aggregation process are non-interactive. The main idea is that all nodes (i) provide a signature share on some common value (e.g. a round number), (ii) verify the received signatures shares and (iii) combine the valid shares to obtain the next random beacon value. As long as a threshold of nodes contributes valid signature shares the aggregation succeeds. Both approaches require the secure distribution of a shared private key as a

precondition. While a trusted dealer is assumed in [33], Dfinity uses a distributed key generation protocol to establish this key.

The approaches described in the initial Ouroboros protocol [78], Scrape [38] and RandShare [114] all rely on PVSS as an underlying primitive. The general idea is that each node first privately generates a random secret value, and then sends out a publicly-verifiable commitment and shares of this secret using PVSS to all nodes. After verification and filtering out invalid commitments, the nodes begin to reveal their respective secrets. If a node fails to reveal or maliciously withholds its value, the other nodes step in and collectively recover the secret from the shares they received previously. Finally, all revealed/reconstructed secrets are combined to form the randomness.

RandHound [114] and RandHerd [114] are also protocols based on PVSS, but operate in a different manner. RandHound is a one-shot protocol, where a client divides nodes into multiple smaller groups and combines the randomness generated by these groups to form a random beacon value, whereas RandHerd is tailored towards continuous operation. The latter uses RandHound to establish a fair division of nodes, executes a distributed key generation protocol within these groups, and leverages on *collective signing* [115] to produce a sequence of random beacon values.

2.7.1 Comparison Overview

In this section, the results of our comparison of the herein presented and discussed approaches for generating publicly-verifiable distributed randomness are outlined. We highlight that a broad comparison was performed by not only considering protocols specifically targeted at implementing random beacons, but also by including approaches that can readily provide a random beacon functionality as a byproduct of their intended application, such as the provision of a distributed public ledger. Consequently, the underlying models, assumptions, notations, as well as the context differ from protocol to protocol and render an evaluation of the herein presented approaches a non-trivial task. We conducted the comparison to the best of our knowledge, contacted the respective protocol authors to try and clarify ambiguities and explicitly state whenever we were unable to adequately determine certain properties or had to estimate them.

The main results are presented in Table 2.1 and the various protocol properties are discussed in greater detail in the following subsections. For the presented complexity evaluations, n refers to the number of protocol participants, and c denotes the size of some subset of nodes, if one is assumed in the specific protocol. Notice that the subset size c is protocol dependent and, although typically constant, a non-negligible factor for the resulting communication complexity in practice (see Section 2.7.4 for a more detailed discussion).

2.7.2 Communication Model

We classify the communication model of the analyzed protocols into three categories, namely synchronous, semi-synchronous and asynchronous protocols. We call a protocol

Table 2.1: Comparison of approaches for generating publicly-verifiable randomness

		Communication model	Liveness / failure probability[◊]	Comm. complexity (overall protocol)	Unpredictability	Bias-Resistance	Comp. complexity (per node)	Verif. complexity (per passive verifier)	Characteristic cryptographic primitive(s)	Trusted dealer or DKG required
[42]	Algorand	semi-syn.	10^{-12}	$\mathcal{O}(cn)^*$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(c)^*$	$\mathcal{O}(1)^*$	VRF	no
[33]	Cachin et al.	asyn.	✓	$\mathcal{O}(n^2)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(1)$	uniq. thr. sig.	yes
[6]	Caucus	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$	hash func.	no
[72]	Dfinity	syn.	10^{-12}	$\mathcal{O}(cn)$	✓	✓	$\mathcal{O}(c)$	$\mathcal{O}(1)$	BLS sig.	yes[#]
[78]	Ouroboros	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[46]	Ourob. Praos	semi-syn.	✓	$\mathcal{O}(n)^*$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)^*$	$\mathcal{O}(1)^*$	VRF	no
[86]	Proof-of-Work	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	very high[§]	$\mathcal{O}(1)$	hash func.	no
[32]	Proof-of-Delay	syn.	✓	$\mathcal{O}(n)$	✓	✓	very high[§]	$\mathcal{O}(\log \Delta)^\circ$	hash func.	no
[114]	RandShare	asyn.	✗[†]	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[114]	RandHound	syn.	0.08%	$\mathcal{O}(c^2 n)^\ddagger$	✓	✗	$\mathcal{O}(c^2 n)$	$\mathcal{O}(c^2 n)$	PVSS	no
[114]	RandHerd	syn.	0.08%	$\mathcal{O}(c^2 \log n)^\ddagger$	✓	✓	$\mathcal{O}(c^2 \log n)$	$\mathcal{O}(1)$	PVSS/CoSi	yes[#]
[38]	Scrape	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	no
	HydRand	syn.	✓	$\mathcal{O}(n^2)$	$\overset{t}{\nearrow}$✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS	no

◊ For the failure probability we give the upper bound for the parameterization of the protocol as suggested by the respective authors.

* The approach for generating randomness is not described in a stand-alone matter and requires additional communication and verification steps for the underlying consensus protocol or the implementation of e.g. a bulletin board. The herein presented values do not account for this additional complexity.

‡ In contrast to Algorand and Dfinity, the parameter c in RandHound/RandHerd actually depends on n and is thus not constant. This is a direct consequence of sharding n nodes into groups of size c, as the protocols fail to provide availability if any single group fails. Keeping c constant while increasing n leads to a higher number of groups m, and thus increases the probability of a liveness failure. To counter this effect requires a security/performance tradeoff where c also has to be increased as n grows. In Section 2.7.3 we further outline that c is a relevant factor in practice, in particular if one wants to achieve a similar liveness guarantee as e.g. Algorand or Dfinity.

† The protocol only provides liveness with additional synchrony assumptions. See Section 2.7.3 for a detailed discussion.

§ The complexity is not dependent on the number of nodes n, but the involved Proof-of-Work is inherently computationally demanding. For Proof-of-Delay the computational complexity depends on the chosen input for the delay function. For the typical choice of using the blockhashes of the underlying Proof-of-Work system as inputs, the cost of the mining process is inherited.

In Dfinity's and RandHerd's approach nodes are split into smaller groups. Within each of these groups a distributed key generation protocol is run.

$\overset{t}{\nearrow}$ The protocols provide probabilistic guarantees for unpredictability, which quickly (exponentially in the waiting time) get stronger the longer a client waits after it commits to use a future protocol output. For HydRand, we indicate that unpredictability with absolute certainty is reached after f rounds using the additional ✓symbol.

° We refer to the verification executed within the Smart Contract via an *interactive* challenge/response protocol. It has logarithmic complexity $\mathcal{O}(\Delta)$ in the security parameter Δ, which describes how many iterations of the hash function are applied to the seed.

synchronous, if a fixed known upper bound on message propagation delay is assumed. If no assumption on this delay is imposed by the protocol and messages are only eventually delivered, we categorize the protocol as asynchronous. If some weaker assumptions in regard to synchrony are made, we informally use the term semi-synchronous. This applies for instance to Algorand and Ouroboros Praos, where the underlying assumptions are outlined in detail, but are not readily comparable to other definitions of partial-synchrony, such as those first established in the context of distributed consensus [49, 53].

Dfinity [72] is aimed at a semi-synchronous setting, however security proofs are currently only published for the synchronous case.

We inferred the synchrony assumption from the protocol description or the underlying protocol whenever they have not been stated explicitly. Currently deployed Proof-of-Work blockchains such as Bitcoin and Ethereum assume a synchronous communication model. As Proof-of-Delay [32] and Caucus [6] are built on top of such Proof-of-Work blockchains, these protocols are also classified as synchronous. In [114], RandShare is described within an asynchronous setting[6]. For RandHound and RandHerd synchrony is indicated in various paragraphs, e.g. III. A. for RandHound and IV. B. 1) for RandHerd [114].

2.7.3 Liveness/Availability

In regard to liveness, we distinguish between three different protocol types:

1. protocols which achieve liveness unconditionally (in the respective system model)
2. protocols which have a (configurable) but non-zero probability of a liveness failure
3. protocols which do not provide liveness (in the respective system model)

We mark protocols of the first type with a ✓ symbol in our comparison table. This category also includes HydRand, which achieves liveness and guaranteed output delivery in the respective system model unconditionally. For protocols of the second type, namely Algorand, Dfinity, RandHound and RandHerd, we give a typical failure probability as described by the respective authors. The authors of Algorand and Dfinity consider failure probabilities of at most 10^{-12} [42] and $2^{-40} \approx 10^{-12}$ [72] as suitable for the respective setting, whereas a typical failure probability of 0.08% [114] is stated for the exemplary configuration in the RandHound and RandHerd protocols.

For all of the above protocols, the failure probability can be adjusted through a security parameter. For example, to lower the failure probability of RandHound and RandHerd to a level of 10^{-12}, the group size c can be increased. By applying the formula given in Syta et al. [114], we observe an increase in group size from $c = 32$ to $c = 125$ to achieve this failure rate against an adversary controlling less than 1/3 of the nodes. Consequently,

[6] see Section 2.7.3 for detailed discussion on liveness problems in this setting

performance is decreased, as the communication complexities of both protocols contain c as a quadratic factor.

The RandShare protocol is described in an asynchronous communication model (under an $n = 3f + 1$ adversary assumption). However, a closer analysis of the protocol shows that further synchrony assumptions are required and therefore RandShare does not guarantee liveness under full asynchrony. The problem arises in paragraph II. D. 2. 1) [114], where $s_j(i)$ is used to denote the secret share of the secret $s_j(0)$, which node j sends privately to node i.

> Initialize a bit-vector $V_i = (v_{i1}, ..., v_{in})$ to zero, to keep track of valid secrets $s_j(0)$ received. Then wait until a message with share $s_j(i)$ from each $j \neq i$ has arrived.

In an asynchronous setting a node i cannot wait to receive a message from *each* other node j, as Byzantine nodes might never send such a message. Similarly, a node should not broadcast a negative vote in case no value $\hat{A}_j$ is received, as described in paragraph II. D. 2. 3), because this would imply a time bound for being able to send valid votes.[7]

2.7.4 Communication Complexity

In Table 2.1, we outline the communication complexity of different approaches that provide randomness either as a stand-alone service or by deriving it from the characteristics of the underlying protocol. Thereby we consider the overall bits transmitted for all nodes per round, i.e. per produced random beacon value.

The different approaches exhibit a wide range of communication complexities. In the simplest scenario, where a Proof-of-Work blockchain forms the basis for the random beacon, a successful miner only has to perform one broadcast, leading to a complexity of $\mathcal{O}(n)$. This also applies for the Proof-of-Delay approach. Caucus also provides a low communication complexity of $\mathcal{O}(n)$ by leveraging the properties of the underlying Smart Contract platform.

For the Algorand and Ouroboros Praos protocols, an analysis of the communication complexity is not provided in the respective publications [42, 46]. We infer that Ouroboros Praos has a communication complexity in $\mathcal{O}(n)$, because the protocol only provides guarantees for eventual consensus and is based upon many of the design principles of Proof-of-Work blockchains, whereas protocols like Algorand, which provide consensus finality, generally operate at a higher per round communication cost. Both protocols use a similar approach based on private randomness, where a verifiable random function

[7] Even if this issue is corrected, i.e., by modifying the protocol to only wait for $2f + 1$ shares, and broadcasting negative votes only after receiving $2f + 1$ valid messages, the protocol can not guarantee liveness, as the threshold of $2f + 1$ positive votes as described in paragraph II. D. 2. 4) may never be reached, and consequently the protocol aborts as stated in paragraph II. D. 3. 2).

(VRF) is used to compute and verify a local source of randomness. The leader's local randomness is then combined with the previous global randomness to obtain the next global randomness. Used in this way, the communication complexity is only dependent on the underlying agreement protocol and does not incur any additional overhead.

To optimize the amount of data transmitted, the Algorand and Dfinity protocols perform certain communication-heavy operations only within a single subset of nodes. RandHound and RandHerd employ sharding to split nodes into multiple smaller groups, where some operations are performed independently within all groups, and the results from individual groups are then combined in a final step. The required sizes for these subgroups typically depend on the assumptions in regard to the adversarial power and the described failure probability. Algorand is designed for a very large number of nodes, and the group size is $c \approx 1000$ [69]. Dfinity outlines a group size of $c = 405$ under a $n = 3f + 1$ security assumption and a failure probability of $2^{-40} \approx 10^{-12}$. As the authors outline, for small values of n, Dfinity's random beacon protocol may also be executed by all nodes, i.e. without selecting a committee as a subset of all nodes. In this case n nodes broadcast a signature share to all other nodes, leading to a complexity of $\mathcal{O}(n^2)$.

For RandHound and RandHerd, group sizes of 16, 24, 32 and 40 are considered by the authors. As we outline in Section 2.7.3, a group size of $c \geq 125$ is required to establish a failure probability similar to Algorand or Dfinity.

The approaches employed by Ouroboros, RandShare and Scrape are similar, where each node in the protocol employs a PVSS scheme to commit to a secret value. This involves the distribution of the PVSS shares, i.e. each node has to broadcast a message of size $\mathcal{O}(n)$ to all other nodes. The resulting communication complexity of $\mathcal{O}(n^3)$ is a major drawback of these approaches, however in this context (P)VSS can also help to achieve guaranteed output delivery [97].

HydRand is similar in this respect, as it also uses PVSS as an underlying primitive, but improves efficiency by a factor of $\mathcal{O}(n)$ because only a single node has to perform the distribution of PVSS shares per round. HydRand's communication complexity of $\mathcal{O}(n^2)$ includes all messages required to establish Byzantine agreement. The communication complexity is reduced by shifting the transmission of messages of size n to the leader and employing cryptographically signed *conformation/recovery certificates* to converge on a history of datasets. Messages that need to be broadcast by all nodes are always of constant size. In our evaluation (see Section 2.6), we provide information on the produced network traffic for different numbers of nodes (n) in practice.

2.7.5 Unpredictability

Unpredictability is a key property related to randomness that is provided by all compared protocols. We distinguish between the following two types of unpredictability that the protocols achieve:

1. all future random beacon values are fully unpredictable for all participants, and

2. the probability of predicting future random beacon values decreases exponentially with the number of rounds to predict.

Protocols, where each round's random beacon is dependent on the input of a (Byzantine) quorum of participants, namely the protocols by Cachin et al., Ouroboros, Dfinity, RandShare, RandHound, RandHerd and Scrape, fall into the first category. For Proof-of-Work, Algorand, Caucus, Ouroboros Praos this is not the case and the next random beacon depends on a single node's (i.e. the miner's or the leader's) secret value. Clearly, since this node knows the next random number in advance it is able to predict the next random beacon value. In case adversarial nodes mine a sequence of blocks or are selected repeatedly as leader, prediction of more than one value is possible if they collude. This issue is typically addressed by a random selection of the respective leader, rendering prediction unlikely quickly. As long as the leader selection process ensures that honest nodes are selected with non-negligible probability, the probability of successful prediction decreases exponentially with the number of rounds to predict.

Proof-of-Delay can, in principle, achieve full unpredictability or unpredictability with high probability even though the next random beacon value depends on the output of a single node, because the leader (e.g. the miner who finds a valid PoW) does not immediately know the resulting random number that is derived from their output. If the leader tries to predict a future value, it has to withhold their output until it is able to finish the sequential computation required for the delay function. Depending on the underlying synchrony assumptions and consensus protocol, withholding the solution (e.g. block) for too long will either exclude the leader's output with certainty or high probability, as the delay parameter can be set much greater than the time bounds used for consensus.

In the context of unpredictability, HydRand offers both unpredictability with exponentially increasing probability for at most f rounds, as well as *full unpredictability* after $f + 1$ rounds. We provide a detailed analysis in Section 2.5.3, outlining the necessary waiting times to achieve an error margin of 10^{-12} for different participant numbers when waiting less than $f + 1$ rounds to achieve guaranteed unpredictability.

2.7.6 Bias-Resistance

Bias-resistance is the property that ensures a protocol's output cannot be manipulated by a (colluding) adversary, i.e. each random beacon value should be uniformly drawn from the set of possible values. Following the work in Cascudo at el. [38], we observe that bias-resistance is closely related to the property of guaranteed output delivery. In case an adversary can learn a candidate output and subsequently prevent the random beacon protocol from producing that output, the resulting beacon values are no longer guaranteed to be uniformly distributed. Even if an adversary is only able to prevent the output of a random beacon value to be available at some specific time, without having previously gained knowledge of the candidate value itself, bias resistance may not be guaranteed. Here, the synchrony requirements of the application(s) toward the delivery of new random beacon values determine biasability. In either cases, further security

assumptions and additional primitives (e.g. PVSS or threshold signatures and $n > 2$ participants) are necessary if bias-resistance is to be guaranteed.

For all (of the compared) protocols, where the last interacting party can influence the random beacon value, this strong form of bias-resistance can not be ensured. This does not necessarily imply that an adversary can arbitrarily manipulate the probability distribution or, even worse, select a specific output. For example, the respective publications for Algorand and Ouroboros Praos show techniques to efficiently use this somewhat biasable form of randomness for the purpose of leader selection.[8] However, if the specific application requires a true uniform distribution of random beacon values, only protocols that provide the previously outlined strong notion of bias-resistance should be considered, namely the protocols by Cachin et al., Dfinity, Ouroboros, Proof-of-Delay, RandShare, RandHerd, Scrape and HydRand.

2.7.7 Computation and Verification Complexity

For our analysis we distinguish between (i) computation complexity, which describes the amount of operations an active protocol participant has to perform during one round of the protocol, and (ii) verification complexity, referring to the amount of computation an external (passive) observer of the protocol has to perform in order to verify the correctness of one random beacon value.

A main drawback of using Proof-of-Work and Proof-of-Delay as a source of randomness is the high computational complexity, as both approaches inherently rely on solving cryptographic puzzles as part of their security model. The other protocols herein considered have a computational complexity of at most $\mathcal{O}(n^3)$. The protocols RandShare and Ouroboros, which require $\mathcal{O}(n^3)$ due to the involved PVSS instances, may be optimized by updating the employed PVSS scheme to the variant introduced by Scrape [38]. The VRF based approaches from Algorand, Ouroboros Praos as well as Caucus (after the initial setup) are very efficient, because they only require the verification of a VRF or hash preimage. In regard to verification complexity, when applying the optimization of the PVSS protocol introduced by Scrape, all protocol outputs can be verified reasonably efficiently, i.e. within $\mathcal{O}(n^2)$.

For Proof-of-Delay, the drawbacks of a high verification complexity, and consequently the disadvantages of an interactive verification process for the use within Smart Contracts, may also be addressed by employing verifiable delay functions (VDF) [18, 118, 94, 19]. While VDFs are not sufficient to provide the functionally of a random beacon on their own, they enable efficient verification of the involved sequential computation steps and can be used in combination with a consensus protocol for agreement on the VDF inputs to form a random beacon. On the contrary, the high computation complexity is inherent in Proof-of-Delay protocols and cannot be reduced using VDFs.

[8] Both publications are aware of, and analyze the fact that the distribution of random numbers produced by their approaches is not uniform and consider the potential implications [42, 46].

The most efficient protocols in regard to computation and verification complexity are based on threshold signatures, VRFs and Proof-of-Work. We consider these approaches most suitable for verification within smart contracts or embedded devices, if fast implementations of the required cryptographic primitives are available within the specific platform.

2.8 Discussion of HydRand and Existing Approaches for Generating Distributed Randomness

The comparison in Section 2.7 outlines that there exist a variety of different approaches for implementing random beacon protocols. Improvements in one characteristic or aspect are often met with negative trade-offs in others, providing no clear candidate that is suitable for all applications. In the following, we discuss defining characteristics of the herein considered protocol designs, to aid in the selection process for particular use case scenarios.

2.8.1 Key Characteristics of Existing Protocol Designs

Both Proof-of-Work [86] and Proof-of-Delay [32] based random beacon approaches are well suited for larger and dynamic sets of participants and can easily leverage on existing Proof-of-Work blockchains. While Proof-of-Work alone is not sufficient to establish bias-resistance, Proof-of-Delay can serve as an augmentation to achieve this guarantee with high probability. However, both approaches require a very high amount of computational resources. Proof-of-Delay may also serve as a suitable bootstrapping mechanism for generating an initial random value to be used in other protocols.

Ouroboros [78], RandShare [114], and Scrape [38] are PVSS based protocols. While the produced randomness of these approaches satisfies strong notions of unpredictability and bias-resistance, their high communication overhead significantly impacts scalability. Consequently, these protocols seem most suitable for a small scale setting (e.g. a private/consortium blockchain) or as an alternative for a Proof-of-Delay bootstrapping mechanism without the computational requirements.

Caucus [6] is an approach that can be deployed and efficiently verified within Smart Contracts but unfortunately cannot ensure bias-resistance.

Algorand [42] targets a large set of nodes while still being able to provide consensus finality without requiring strong synchrony assumptions. As a trade-off, the protocol can not ensure a strong notion of bias-resistance. In this regard Ouroboros Praos [46] makes a similar trade-off to achieve better scalability at the cost of consensus finality and also weakening bias-resistance.

The randomness produced by the threshold signature based protocols of Cachin et al. [33] and Dfinity [72] provide strong bias-resistance. Additionally, Cachin et al. is the only protocol in our comparison that is proven secure in an asynchronous communication

model. Dfinity's approach scales to a larger number of nodes, but security is only proven in a synchronous system model. The drawback of both protocols is their reliance on cryptographic primitives that are based on elliptic curve pairings, which are not yet well-established. E.g. Menezes et al. [84] and subsequently Barbulescu et al. [9] showed the security level of a commonly used pairing-friendly curve is in fact 2^{110} or 2^{100} instead of the targeted 2^{128}. Also these protocols require a trusted dealer or distributed key generation protocol.

RandHound [114] and RandHerd [114] employ a sharding approach to achieve good scalability for a large number of participants. RandHound does not provide a strong notion of bias-resistance while RandHerd requires additional view-change and agreement protocols when a leader is Byzantine or non-available.

2.8.2 Advantages and Limitations of HydRand

HydRand is a dedicated random beacon protocol tailored towards continuous operation and assumes a small to medium set of nodes. The protocol provides strong properties that are comparable to other PVSS-based approaches, while reducing the communication overhead by $\mathcal{O}(n)$. A resulting trade-off is the need to wait for $f+1$ rounds for guaranteed unpredictability, however strong probabilistic unpredictability is ensured within a few rounds (see Figure 2.2), and bias-resistance is always achieved.

An evaluation (see Section 2.6) of our open-source Python implementation of the HydRand protocol outlines the practicability for a wide range of participant configurations, while requiring minimal hardware resources. The protocol design is simple, and its design goals are achieved without requiring a trusted dealer or DKG in the initial setup, thereby avoiding the introduction of additional security assumptions and implementation complexity. Moreover, a detailed analysis and security proofs of the protocol's properties and guarantees are provided.

HydRand furthermore ensures *guaranteed output delivery*: A new random beacon value is guaranteed to be produced at each round, i.e. in regular intervals, regardless of the adversary's actions. This is of particular importance for application scenarios in which strong synchrony requirements or gapless delivery of new random beacon values is required. To achieve the design goal of producing random beacon values at regular intervals, HydRand implicitly requires synchronous round-to-round communication. A resulting drawback is that any leader which (temporarily) fails to deliver required messages is excluded from further participation. Consequently, in systems where synchrony guarantees may have a probability of being temporarily violated, the round duration parameter has to be carefully selected to avoid any resulting liveness failures. In future work, we envision an extension of the HydRand protocol to also consider and tolerate crash-recovery failures, which may be able to address the current limitations in this regard.

Requiring strong synchrony can also prove advantageous for a public randomness beacon, as an external validator with knowledge of the setup parameters and the protocol start time cannot be tricked into accepting outdated random beacon values. Further, unlike

protocols aimed at a dynamic set of participants (e.g. Algorand, Dfinity or Ouroboros Praos), a static set can also render the validation of random beacon values simpler. For a static validator set, no additional proofs need to be provided to convince any third party, which has not observed the entire protocol execution, that the current set of validators has legitimately evolved from some initial configuration.

Although excellent performance results were obtained when testing our implementation with up to 128 globally distributed nodes, scalability to a much larger set of participants is limited due the inherent communication complexity of $\mathcal{O}(n^2)$. In such a scenario, approaches where the consensus algorithm is only executed by a subset of the participating nodes, or Proof-of-Delay based protocols may prove advantageous.

2.9 Summary of our Findings on the HydRand Protocol

In this chapter we present HydRand, a synchronous random beacon protocol that tolerates up to one third Byzantine failures and show that it provides *liveness*, *public-verifiability*, *bias-resistance*, and probabilistic as well as hard bounds for *unpredictability*. HydRand ensures *guaranteed output delivery*, namely that randomness is produced at regular intervals, even under adversarial conditions. The protocol is designed for stand-alone use, but could also find utility in the context of current and future Proof-of-Stake and permissioned blockchain or consensus protocols.

Additionally, we provide the first in-depth comparison and discussion of novel approaches for generating publicly-verifiable randomness, which enables researchers to compare current as well as future designs objectively with each other. We highlight the different trade-offs these approaches make and provide a detailed discussion on the protocol properties, which supports future researchers and application engineers during the selection process for their specific use case. Thereby, we highlight that HydRand achieves various desirable properties in a unique way without incurring major drawbacks: (i) it is a stand-alone protocol that can be readily adapted for different use-cases, (ii) it neither requires a trusted dealer nor a distributed key generation protocol, and (iii) it offers strong guarantees for the produced randomness while improving upon the performance and scalability of previous solutions with comparable guarantees.

Furthermore, we develop and evaluate a fully functional protocol prototype in Python to demonstrate the feasibility and practicability of HydRand. The source code and additional information on the implementation details are publicly available on Github [107].

2.A Appendix: HydRand Notation Reference

Table 2.2: HydRand notation reference – symbols

Symbol	Description
f	number of Byzantine nodes
n	number of all nodes, defined as $n = 3f + 1$
t	reconstruction threshold for PVSS, defined as $t = f + 1$
i	a node as defined by context
r, k, x	some round as defined by context
ℓ	leader of the current round r
ℓ_x	leader of round x
$H(\cdot)$	cryptographic hash function
$\langle sk_i, pk_i \rangle$	private/public keypair of node i
$\langle m \rangle_i$	some message m signed using the secret key sk_i of node i
$\|\|$	string/list concatenation
R_x	randomness of round x
D_x	dataset of some round x, consists of a $header(D_x)$ and $body(D_x)$
$H(D_x)$	cryptographic hash of the $header(D_x)$
$\overleftarrow{x}$	previous round of round x, such that there exists a valid dataset for round $\overleftarrow{x}$
$D_{\overleftarrow{x}}$	previous dataset referenced in dataset D_x
$\mathcal{P}$	set of all nodes (processes), $\mathcal{P}$ is of size n
$\mathcal{P}_x$	set of available nodes for some round x, i.e., set of all nodes excluding recovered nodes till round x
$\mathcal{L}_x$	set of potential leaders for some round x, i.e., set of all nodes excluding recovered nodes till round x and excluding nodes that have been selected as leader within the last f rounds
$rn(D_x)$	set of recovered nodes up to block D_x
q	prime number q
$\mathbb{Z}_q$	ring of integers modulo q
$\mathbb{G}_q$	multiplicative group of order q, in which the discrete log problem hard
h	generator for the group $\mathbb{G}_q$
s	underlying secret value, a dealer wants to share with PVSS, $s \in \mathbb{Z}_q$
$Com(s)$	PVSS commitment to the value s, includes commitments to the coefficients of the underlying polynomial, encrypted shares and a NIZK correctness proof.
h^s	result of the reconstruction process for a commitment $Com(s)$
$\hat{s}_i$	encrypted share for node i, part of the commitment $Com(s)$
$Com(s)[s_i]$	node i's decrypted share for the commitment $Com(s)$, result of decrypting $\hat{s}_i$ using i's private key

Table 2.3: HydRand notation reference – symbols continued

Symbol	Description
s_ℓ	current leader's previously committed secret value.
$s_\ell^\star$	current leader's new randomly selected secret value.
$Com(s_\ell)$	current leader's previous commitment
$Com(s_\ell^\star)$	current leader's new commitment
$CC(D_x)$	*commit certificate* of dataset D_x that contains at least $f+1$ valid *confirmation* messages.
$RC(x)$	*recovery certificate* of round x that contains at least $f+1$ valid *recover* messages.
M_x	root of a Merkle tree for the shares $\hat{s_1}, \hat{s_2}, ..., \hat{s_n}$ for ℓ_x's commitment $Com(s_{\ell_x})$ in round x
$M_x[\hat{s_i}]$	merkle branch for $\hat{s_i}$, showing that $\hat{s_i}$ is under the Merkle root M_x (and thus part of D_x)

Table 2.4: HydRand notation reference – message formats

Message	Description
$\langle propose, \langle header(D_r)\rangle_\ell, body(D_r)\rangle_\ell$	The message that is broadcasted by correct leaders in the *propose* phase of each round.
$\langle\langle acknowledge, r, H(D_r)\rangle_i, \langle header(D_r)\rangle_\ell\rangle_i$	The message that is broadcasted by correct nodes that received a valid *propose* messages from the leader of the current round. Broadcasting this messages ensures that the leader cannot equivocate.
$\langle confirm, r, H(D_r)\rangle_i$	The message that is broadcasted by correct nodes that received $2f+1$ valid *acknowledge* messages from other nodes during this round. Any node which received $f+1$ of these messages can construct a valid *confirmation certificate* for round r.
$\langle\langle recover, r\rangle_i, s_\ell, Com(s_\ell)[s_i], \hat{s_i}, M_k[\hat{s_i}], R_{r-1}\rangle_i$	The message that is broadcasted by correct nodes that did not receive a valid *propose* message from the leader at the beginning of this round. Any node which received $f+1$ of these messages can reconstruct a valid *recovery certificate* for round r.
$(f+1) \times \langle confirm, r, H(D_r)\rangle_i$	commitment certificate $CC(D_r)$ for dataset D_r with hash $H(D_r)$ (valid if it contains correctly signed messages from $f+1$ different nodes i)
$(f+1) \times \langle recover, r\rangle_i$	recovery certificate $RC(r)$ for round r (valid if it contains correctly signed messages from $f+1$ different nodes i)

CHAPTER 3

EthDKG: Distributed Key Generation with Ethereum Smart Contracts*

In the previous chapter we described HydRand, a novel protocol for generating distributed randomness. HydRand's used cryptographic primitives (i.e., digital signatures, hash functions, and publicly-verifiable secret sharing) rely upon well established cryptographic assumptions. The protocol's setup is simple and does not require interaction between the participants. Together with the strong guarantees provided, HydRand is suitable for a wide range of application scenarios. Comparing HydRand with protocols which rely on threshold signatures as their main cryptographic primitive, we show that HydRand asymptotically achieves the same communication complexity (see Section 2.7). However, we also observe that the use of unique threshold signatures for implementing a randomness beacon, in particular the Boneh-Lynn-Shacham (BLS) signature scheme [24], leads to small message sizes which, together with the used aggregation techniques, reduce the communication overhead in practice. This advantage comes with the drawback of additional cryptographic assumptions (pairing based cryptography) and the question on how to securely generate and distribute the key shares required for the threshold signing operations. Still, the increased communication efficiency and constant computational costs for verifying a (pre-aggregated) random beacon output can outweigh the drawbacks in many circumstances. An prominent application example where is fact becomes particularly clear is the provisioning of distributed randomness for a smart contract

*This chapter is an extended version of the equally-named research paper initially presented at the 2019 Cryptocurrency's Implementer Workshop (CIW 2019) hosted at the 23rd International Conference on Financial Cryptography and Data Security conference (FC 2019). Large text passages from the original work are used in verbatim form in this work.

platform such as Ethereum. In this case, computational costs directly relate to monetary costs and are particularly high in case of network congestion, which mandates the reduction of the use of computational resources for a cost effective operation. While one has to study but eventually accept the additional cryptographic assumptions when opting for a threshold signature based solution, the remaining question of how to securely generate the key shares is addressed by distributed key generation (DKG) protocols – the main topic covered in this chapter.

Distributed key generation (DKG) is a fundamental building block for a variety of cryptographic schemes and protocols, such as threshold cryptography, multi-party coin tossing schemes, public randomness beacons and consensus protocols. More recently, the surge in interest for blockchain technologies, and in particular the quest for developing scalable protocol designs, has renewed and strengthened the need for efficient and practical DKG schemes. Surprisingly, given the broad range of applications and available body of research, fully functional and readily available DKG protocol implementations still remain limited. With our work on the *EthDKG* protocol, we aim to close this gap by tailoring Gennaro et al.'s [63] well known protocol design towards being efficiently implementable within public cryptocurrency ecosystems such as Ethereum. Our theoretical improvements are supported by an open source, fully functional, well documented DKG implementation[1] that can employ any Ethereum Virtual Machine (EVM) compatible smart contract platform as a communication layer. We evaluate the efficiency of our protocol and demonstrate its practicability through the deployment and successful execution of our DKG contract in the Ethereum Ropsten testnet. Given the current Ethereum blocks gas limit, all steps required for the key generation process, even in demanding scenarios tested with up to 256 nodes, can be verified at the smart contract level.

3.1 Introduction to Distributed Key Generation Protocol

Distributed key generation (DKG) protocols serve as a key building block for threshold cryptography. The goal of a DKG scheme is to agree on a common secret/public key pair such that the secret key is shared among a set of n participants. Only a subset of $t+1 \leq n$ parties can use or reveal the generated secret key, while t collaborating parties cannot learn any information about it. In this regard DKG is related to secret sharing protocols, as first introduced by Shamir [110] and Blakley [16]. However, in contrast to secret sharing, DKG protocols do not rely on a (trusted) dealer which generates, knows and distributes the secret key, and hence avoid this single point of failure. Instead, the key pair is generated using a multi-party computation in a way that no single party learns the secret that is being shared.

Distributed key generation has been studied and discussed for over two decades [92, 22, 63, 64, 76, 77, 87]. However, the extensive body of literature is currently not matched

[1]The source code, documentation, and logs of a successful execution in the Ropsten testnet are publicly available at https://github.com/PhilippSchindler/EthDKG/.

by a single clear, succinct, and practical protocol design template that reflects the state of the art and leverages on recent technical developments such as distributed ledgers. Moreover, real-world open source implementations of DKG protocols are still rare, and often not well documented.

We aim to close this gap by providing and evaluating a lightweight, scalable, and well-documented protocol design and open source implementation of a DKG protocol. Our design is based on the Joint-Feldman DKG protocol [63] and incorporates the enhancements proposed by Neji et al. [87] to address biasing attacks [63], without requiring two distinct secret sharing rounds. Additionally, we describe and implement a new mechanism that handles disputes during the protocol execution more efficiently. The resulting protocol design is described in its generality for any discrete logarithm based cryptosystem, and we demonstrate that our protocol improvements enable the verification of the key generation process within Ethereum, and similar smart contract platforms.

Leveraging the capabilities provided by distributed ledger-based smart contract platforms, our DKG protocol allows the set of participating entities to be dynamically defined and can incentivize participation as well as penalize adversarial behavior. Further, we are able to ensure that any security deposits provided by participants following the protocol rules always remain safe, even if the DKG protocol itself is executed by a majority of adversarial participants. This design approach can help address the issue of Sybil nodes in settings where *open* participation for better decentralization [116] in the DKG is desirable.

3.1.1 Structure of the Remainder of this Chapter

We continue this chapter by introducing and comparing related work to our approach in Section 3.2. We describe our system model, including assumptions concerning the network infrastructure, the capabilities of the adversary as well as the security properties expected from DKG protocols in Section 3.3. Our generalized protocol design for discrete logarithm based cryptosystems is presented and analyzed in Section 3.4 and Section 3.5. Section 3.6 provides implementation specific details, while Section 3.7 provides our evaluation results. Finally, we discuss and conclude the chapter in Section 3.8 and Section 3.9. Additionally, a notation reference for the symbols used in our descriptions of the EthDKG protocol is provided in Appendix 3.A at the end of this chapter.

3.2 Related Work in Distributed Key Generation

The first protocol for DKG was introduced by Pedersen [92] in 1991, and was subsequently built upon within a wide range of publications in the field of threshold cryptography. A popular variant is the so called Joint-Feldman DKG protocol, introduced by Gennaro et al. [63] as a simplification of Pedersen's work. The core idea of the Pedersen (and the Joint-Feldman) protocol is that each party executes Feldman's [56] verifiable secret

sharing (VSS) protocol, acting as a dealer in order to share a randomly chosen secret among all parties. After a verification step, ensuring the participants shared their secrets correctly, the resulting group private key is defined by the sum of the properly shared secrets. This private key is unknown to the individual participants, but may be obtained by a collaborating group of parties. The corresponding public key can be computed using the commitments published during the sharing phase with Feldman's VSS protocol and is the public result of executing the DKG protocol.

However, as described in great detail by the works of Gennaro et al. [63, 64], keys generated using (a wide range of variants of) the Pedersen protocol, are not guaranteed to be uniformly distributed over the respective keyspace. An adversary can bias bits of the resulting key by selectively denouncing the validity of shares of one or more of the parties it controls. Consequently, the set of parties which properly shared their secrets, and thus define the resulting key, is influenced as the denounced parties are excluded. The issue in this case is that honest parties have provided all the information required to compute the resulting public key *before* agreement on the set of shares that are used to create the master key is reached, allowing the adversary to influence the final outcome.[2]

Gennaro et al. [63] presents mitigation strategies against these kind of attacks. However, their approach adds complexity as it requires an additional secret sharing step using Pedersen's VSS protocol [91]. Canetti et al. [36] extend the solution from Gennaro et al. to cope with adaptive adversaries, which may corrupt parties based on prior knowledge gathered during the protocol execution. More recently, Neji et al. [87] describe a different countermeasure which we adopt in this work, avoiding these drawbacks.

Kate and Goldberg [76] were the first to study DKG in an asynchronous communication model, whereas synchronous message delivery was previously assumed. In order to support these weaker assumptions, they require a network of $n \geq 3t + 2f + 1$ participants, out of which t are controlled by the adversary and thus considered Byzantine and f parties may fail in the crash-stop model. This is in contrast to works in the style of Gennaro et al. and our protocol, which require synchrony but can tolerate ($n \geq 2t + 1$) Byzantine adversaries. In a subsequent extension of their work [77], Kate et al. provide an implementation, tested with up to 70 parties distributed over multiple continents. A crucial distinction between Kate and Goldberg's work and the approach followed by Gennaro et al. and this work, is that the former also implement a Byzantine agreement protocol alongside the DKG, whereas the consensus protocol is not part of the DKG specification in the latter. We outline the advantages and drawbacks of both design decisions in our discussion (see Section 3.8).

To the best of our knowledge, the DKG protocol developed by the Orbs Network team [88] is the only publicly available protocol targeted at a similar deployment scenario, namely, an implementation of a DKG protocol using the Ethereum platform. However, the presented prototypical implementation appears to be incomplete and has not been

[2]We refer to the works of Gennaro et al. [64, 65] for an in-depth discussion of the implications of a non-uniform distribution.

updated since 5th August, 2018. A peer-reviewed publication outlining the details of the protocol is also not available at the time of writing. Further, this approach, in comparison to the works of Gennaro et al., Kate and Goldberg, and our work, fails to guarantee liveness under adversarial behavior. It requires a protocol restart even if only a single adversarial participant sends an invalid share – a major drawback we can avoid.

3.3 System Model and Threat Model of the EthDKG Protocol

Using our protocol, a set of n participants $\mathcal{P} = \{P_1, P_2, ..., P_n\}$ wish to jointly generate a master secret/public key pair of the form $mpk = g^{msk}$ for a discrete logarithm based threshold cryptosystem. We use g and h to denote two independently[3] selected generators of the group $\mathbb{G}_q$ with prime order q and assume that computing discrete logarithms in $\mathbb{G}_q$ is hard and the Computational Diffie-Hellman (CDH) assumption holds for this group. The master public key mpk is the (public) output of the protocol. The corresponding (virtual) secret key msk is shared among the participants, and may be obtained by pooling the shares from $t + 1$ collaborating parties. Depending on the use case scenario, it may not be desirable or even necessary to ever obtain msk. For instance, by employing BLS threshold signatures [24], a signature verifying under the master public key mpk can be obtained by aggregating signature shares without recovering msk first.

3.3.1 Communication Model

We assume all parties can monitor and broadcast messages on a shared public and authenticated communication channel. Further, all participants are in agreement on a common view and ordering of these broadcast messages. We assume synchrony in the sense that, any message that is broadcast by a participant during some protocol phase is received by all other parties before the next phase starts. In this regard, our communication model is closely related to the notion of public bulletin boards [43].

In contrast to Gennaro et al. [63], we do not consider pairwise private communication links between parties. Instead, we assume that each participant $P_i \in \mathcal{P}$ generates a *fresh* secret/public key pair $\langle sk_i, pk_i \rangle$ of the form $pk_i = g^{sk_i}$ prior to the protocol start and knows the public keys of all other participants.[4] Note that these keypairs are independent of the keys used to establish the authenticated communication channel, and are only used to derive a symmetric encryption key for each sender/receiver pair of nodes. These symmetric keys are then used *once* to ensure the secrecy of the key shares being transmitted in the sharing phase of the protocol.

[3] I.e., the discrete logarithm $dlog_g(h)$ between g and h is unknown.

[4] Instead of assuming a priori knowledge of the other parties' keys, an additional registration phase (see Section 3.6.6) can be used to exchange the public keys.

Blockchain protocols, which allow inclusion of arbitrary data, and other BFT state machine replication and distributed ledger protocols present suitable candidates for such communication channels. In practice, we leverage the Ethereum blockchain as a public authenticated communication channel and consensus protocol, where agreement on message ordering is ensured through the common prefix property [62]. Together with our client software, which enforces appropriate stabilization times to ensure agreement with high probability, the desired guarantees can be achieved. We refer to Section 3.6.1 for additional details on how the communication channel is instantiated.

3.3.2 Adversarial Model

To ensure secrecy of the generated secret key msk, we assume that an adversary controls at most t participants, whereas a collaboration of $t+1$ participants is required to derive msk. A node controlled by the adversary may deviate arbitrarily from the specified protocol. We consider an *adaptive* adversary, in the sense that it can decide which parties to corrupt based on prior observations. However, the adversary is *not mobile*, once a party is corrupted it is considered compromised for the entire protocol execution. To guarantee both, secrecy of the generated key as well as liveness, i.e., that the protocol completes successfully, the adversary must not control more than $t < n/2$ parties. These are the optimal bounds one can hope to achieve in this setting [63].

3.3.3 Security Properties

In the following, we reiterate on the security properties we aim for and expect from a DKG protocol. Hereby, we follow the definitions given by Gennaro et al. [63] and Neji et al. [87] for *correctness* and *secrecy* and refer to the corresponding works for a more formal definition. The *uniformity* property highlights a shortcoming identified by Gennaro et al. [63] that was not covered by the original Joint-Feldman protocol. Because recent DKG implementations appear to not consider this property, e.g., the Ethereum-based DKG implementation in [88], we use a distinct category to further emphasize this characteristic. *Robustness* ensures that a subset of parties, which want to recover the master secret key, is able to do so under adversarial influence. The definitions of secrecy, uniformity and robustness follow the correctness definitions C3 and C1' from Gennaro et al. We also add a definition for *liveness*, which was not explicitly stated in Gennaro et al.'s work.

Secrecy No information about the master secret key msk can be learned by the adversary except for what is implied by the value of the master public key $mpk = h^{msk}$.

Correctness All sets of $t+1$ correct key shares define the same unique master secret key msk and all honest parties agree on the common value of the master public key $mpk = h^{msk}$.

Uniformity The master secret key msk is uniformly distributed in $\mathbb{Z}_q$, and hence the master public key mpk is uniformly distributed in $\mathbb{G}_q$.

Robustness There is an efficient procedure that, on input of the public information of the DKG protocol and n submitted shares, outputs msk, even if up to t invalid shares have been submitted by malicious or faulty participants.

Liveness As long $t+1$ nodes are controlled by correct parties, an adversary cannot prevent the protocol from completing successfully.

3.4 The EthDKG Protocol

In this section, we present our generalized DKG protocol design for discrete logarithm based cryptosystems. We start by giving a brief overview of our three consecutive protocol phases, and then describe each phase in detail in sections 3.4.1, 3.4.2 and 3.4.3. For implementation specific details we refer to Section 3.6.

Sharing Phase During the first phase, each participant in $P_i \in \mathcal{P}$ selects a randomly chosen secret $s_i \in_R \mathbb{Z}_q$ and subsequently uses Feldman's VSS to share this secret among all parties, such that $t+1$ collaborating parties can recover s_i, in case a malicious party withholds the required information during the key derivation phase. The verification procedure of Feldman's protocol enables the parties to check that received shares are indeed valid.

Dispute Phase During the dispute phase, each party that received one or more invalid shares in the previous phase uses a non-interactive proof technique to convince other parties about the fact that the issuer violated the protocol.

Key Derivation Phase At the beginning of the last phase, a set of qualified parties $\mathcal{Q} \subseteq \mathcal{P}$ is formed. A party P_i is part of $\mathcal{Q}$ if and only if it (i) broadcasted the required information during the sharing phase and (ii) no party broadcasted a valid dispute against P_i during the dispute phase. In other words, the set $\mathcal{Q}$ contains all parties which correctly shared their secret and should thus contribute to form the master key pair $\langle msk, mpk \rangle$. Finally, for all parties $P_i \in \mathcal{Q}$ the values h^{s_i}, related to the randomly chosen secrets s_i, are either revealed or recovered and used to derive the master public key mpk. Using Lagrange interpolation, msk can be computed after pooling the shares from $t+1$ parties. However, depending on the use case scenario, it may not be desirable or necessary to ever obtain msk.

3.4.1 Sharing Phase

Share Generation At the beginning of the sharing phase, each party $P_i \in \mathcal{P}$ executes the first step of the Joint-Feldman DKG protocol [63]. In order to share a randomly chosen secret $s_i \in_R \mathbb{Z}_q$ among all[5] registered parties, P_i acts as the dealer in an (n, t)

[5] For ease of exposition, we assume that P_i also provides one share for itself.

Feldman VSS protocol [56]. For this purpose it picks a secret polynomial $f_i : \mathbb{Z}_q \to \mathbb{Z}_q$ with coefficients $c_{i0} = s_i$ and $c_{i1}, c_{i2}, ..., c_{it}$ drawn uniformly at random from $\mathbb{Z}_q$:

$$f_i(x) = c_{i0} + c_{i1}x + c_{i2}x^2 + ... + c_{it}x^t \pmod q \tag{3.1}$$

Then P_i computes the shares $s_{i\to j} = f_i(j)$ for all $P_j \in \mathcal{P}$, and the commitments $C_{i0} = g^{c_{i0}}, C_{i1} = g^{c_{i1}}, ..., C_{it} = g^{c_{it}}$ to the coefficients of $f_i(\cdot)$. These commitments are used in the verification process for the shares and implicitly define P_i's public polynomial $F_i : \mathbb{Z}_q \to \mathbb{G}_q$:

$$F_i(x) = C_{i0} \cdot C_{i1}^{x} \cdot C_{i2}^{x^2} \cdot ... \cdot C_{it}^{x^t} \tag{3.2}$$

Share Transmission Next, each P_i has to securely send its shares $s_{i\to j}$ to all other parties $P_j \in \mathcal{P}$. Contrary to the original description of the Joint-Feldman DKG, we do not assume access to private communication channels between parties, but rather realize the secure sending of the shares using encryption over our public broadcast channel. We use a symmetric key encryption algorithm $\mathsf{Enc}_{k_{ij}}(\cdot)$ to ensure secrecy of a sent share from P_i to P_j. The corresponding encryption key k_{ij} can be derived non-interactively by both parties:

$$k_{ij} = {pk_j}^{sk_i} = {pk_i}^{sk_j} = g^{sk_i sk_j} \tag{3.3}$$

Notice that this approach is inspired by the techniques used in the Diffie Hellman key exchange protocol [101] and the ElGamal encryption scheme [61].

Finally, P_i broadcasts the encrypted shares $\overline{s_{i\to j}} = \mathsf{Enc}_{k_{ij}}(s_{i\to j})$ for all $i \neq j$ as well as the commitments $C_{i0}, C_{i1}, ..., C_{it}$ from Feldman's VSS. Each party P_j monitors the communication channel for messages broadcasted by other participants. Upon receiving encrypted shares and commitments from P_i, P_j decrypts its share to obtain $s_{i\to j} = \mathsf{Dec}_{k_{ij}}(\overline{s_{i\to j}})$.

Share Verification P_j employs the verification procedure of Feldman's VSS to check the validity of each share $s_{i\to j}$. A share is valid if and only if the following share verification condition holds:

$$g^{s_{i\to j}} = F_i(j) \tag{3.4}$$

In case $s_{i\to j}$ is found invalid, further actions are required in the dispute phase. As P_i only expects to receive a single message from each party, only the first message is processed, any additional messages from the same sender are ignored. In our smart contract based implementation (see Section 3.6), the smart contract itself ensures that parties can only broadcast a single message during the sharing phase.

3.4.2 Dispute Phase

In case a party P_j notices that it received an invalid share $s_{i\to j}$ from P_i in the previous phase, P_j must broadcast a dispute claim in order to ensure that P_i is excluded from further steps of the protocol execution. Intuitively, P_i must be excluded because its secret s_i may not be recoverable by a collaboration of $t+1$ correct parties.

In the original description of the Joint-Feldman DKG protocol, an adversarial P_j can always issue an (unsupported) claim stating that it received an invalid share from a correct P_i, requiring P_i to prove adherence to the protocol rules. We flip this notion in the sense that it is P_j's obligation to show that P_i indeed violated the protocol. To accomplish this we use a non-interactive proof technique described below, and can consequently reduce the required number of interactions between parties.

Issuing a Dispute Claim The key idea how P_j is able to prove that P_i provided an invalid share $s_{i\to j}$ is to publish the key k_{ij} used for encryption and decryption of the share. Using this key, other parties are able to decrypt the previously distributed share $\overline{s_{i\to j}}$ and can, in the same way as P_j did, verify that $s_{i\to j}$ is indeed invalid. To ensure that an adversarial P_j cannot just publish an invalid key k'_{ij}, which would again lead to a false accusation of P_i, it is required that P_j proves the correctness of k_{ij}. We use a common non-interactive zero-knowledge (NIZK) proof technique for showing the equality of the two discrete logarithms [41, 35] to show the correctness of k_{ij}. The corresponding proving and verification procedures are denoted by $\mathsf{DLEQ}(x_1, y_1, x_2, y_2, \alpha)$ and $\mathsf{DLEQ\text{-}verify}(x_1, y_1, x_2, y_2, \pi)$.

Procedure 1: $\mathsf{DLEQ}(x_1, y_1, x_2, y_2, \alpha)$.

To show that $dlog_{x_1}(y_1) = dlog_{x_2}(y_2)$ holds without revealing the discrete logarithm α, a prover proceeds as follows:

1. compute $t_1 = x_1^w$ adding $t_2 = x_2^w$ for $w \in_R \mathbb{Z}_q$
2. compute $c = \mathsf{H}(x_1, y_1, x_2, y_2, t_1, t_2)$
3. compute $r = w - \alpha c \pmod q$
4. output $\pi = \langle c, r \rangle$

Instantiating the above procedure, P_j can prove the correctness of the decryption key k_{ij} by providing $\pi(k_{ij}) = \mathsf{DLEQ}(g, pk_j, pk_i, k_{ij}, sk_j)$ in addition to k_{ij}.

Verifying a Dispute Claim Upon receiving a dispute claim $\langle k_{ij}, \pi(k_{ij}) \rangle$ against P_i, issued by P_j, one can use $\mathsf{DLEQ\text{-}verify}(g, pk_j, pk_i, k_{ij}, \pi(k_{ij}))$ to check the validity of the received key k_{ij}.

Procedure 2: DLEQ-verify$(x_1, y_1, x_2, y_2, \pi)$.

To check the correctness of a proof $\pi = \langle c, r \rangle$, showing that $dlog_{x_1}(y_1) = dlog_{x_2}(y_2)$ holds, a verifier proceeds as follows:

1. compute $t'_1 = x_1^r y_1^c$ and $t'_2 = x_2^r y_2^c$
2. output VALID if $c = \mathsf{H}(x_1, y_1, x_2, y_2, t'_1, t'_2)$ holds
   output INVALID otherwise

If the key is found invalid, the dispute claim is invalid. Otherwise, the verification procedure continues by decrypting the corresponding share $s_{i\to j} = \mathsf{Dec}_{k_{ij}}(\overline{s_{i\to j}})$ and checking its correctness according to the share verification condition specified in Equation 3.4. The dispute is valid if and only if k_{ij} is found valid but the verification condition does not hold.

The protocol ensures that: (i) In case a correct participant received an invalid share from another party, the share issuer is considered disqualified by all (correct) parties at the end of the dispute phase. (ii) An adversary cannot wrongly accuse any correct party of providing it with an invalid share. (iii) The adversary does not gain any additional information when a party P_j reveals the values k_{ij} and $\pi(k_{ij})$, because the adversary can always compute (and therefore publish) $k_{ij} = {pk_j}^{sk_i}$ using P_i's secret key, and the NIZK proof $\pi(k_{ij})$ does not reveal additional information apart from the correctness of the statement.

3.4.3 Key Derivation

Deriving the Set of Qualified Nodes The first step in the key derivation phase is determining the set of qualified parties $\mathcal{Q} \subseteq \mathcal{P}$, describing which parties should contribute to the resulting key pair $\langle msk, mpk \rangle$. If we recall the current protocol state at the beginning of the key derivation phase, we observe that each $P_i \in \mathcal{P}$ has either:

1. *correctly* shared its secret s_i with all other parties.
2. *incorrectly* shared its secret s_i.
3. did not share its secret s_i at all.

We say a secret was correctly shared by P_i, if and only if no valid dispute claim against P_i was filed during the dispute phase. Parties which incorrectly shared their secrets, or did not share their secrets at all, are disqualified and excluded from the upcoming protocol steps. The remaining parties form the set $\mathcal{Q}$. In other words, a node $P_i \in P$ is only part of $\mathcal{Q}$ if (i) it published the values $C_{i0}, C_{i1}, ..., C_{it}$ and $\overline{s_{i\to j}}$ for all $i \neq j$ during the sharing phase and (ii) no node P_j filed a valid dispute against P_i during the dispute phase.

Bias when Computing the Keys Directly Using this definition of the set $\mathcal{Q}$, the resulting group public key mpk could be derived by following the description of the Joint-Feldman protocol:

$$mpk = \prod_{P_i \in \mathcal{Q}} C_{i0} = \prod_{P_i \in \mathcal{Q}} g^{s_i} \tag{3.5}$$

However, as described in great detail by the works of Gennaro et al. [63, 64], the above approach does not ensure that the resulting key pair is uniformly distributed. An adversary can bias bits of the resulting key by selectively denouncing one or more of its nodes, which influences the set $\mathcal{Q}$ and thus the resulting key. The critical[6] issue here is, that all information required to compute the resulting public key is known to the adversary *before* the set $\mathcal{Q}$ is fixed.

Protection against Biasing of the Generated Keys We adopt a recent countermeasure described by Neji et al. [87] to ensure the resulting key is uniformly distributed. The key idea to ensure uniformity is to instead compute mpk as follows:

$$mpk = \prod_{P_i \in \mathcal{Q}} h^{s_i} \tag{3.6}$$

Here, h is used to denote an additional generator of the group $\mathbb{G}_q$, such that $dlog_g(h)$ is unknown. The required values h^{s_i} used to compute mpk are published by the parties in $\mathcal{Q}$ after this set is fixed. Each value P_i shows the correspondence between the values h^{s_i} and $C_{i0} = g^{s_i}$ using the NIZK proof $\pi(h^{s_i}) = \mathsf{DLEQ}(g, g^{s_i}, h, h^{s_i}, s_i)$ as introduced in Section 3.4.2. In case any (adversarial) party $P_i \in \mathcal{Q}$ does not reveal its value h^{s_i} and a valid proof $\pi(h^{s_i})$ by the end of the key derivation phase, a set of $t+1$ correct parties is always able to use the recovery procedure of Feldman's VSS to obtain s_i and consequently h^{s_i} anyway. Without loss of generally, let $\mathcal{R} \subseteq \mathcal{Q}$ denote a set of $t+1$ correct parties. Then, s_i is obtained via Lagrange interpolation:

$$s_i = \sum_{P_j \in \mathcal{R}} s_{i \to j} \prod_{\substack{P_k \in \mathcal{R} \\ j \neq k}} \frac{k}{k-j} \tag{3.7}$$

Deriving the Keys Finally, the common master public key mpk can be derived as specified in Equation 3.6 using the published or recovered values $h^{s_i} \mid P_i \in \mathcal{Q}$. Additionally, each $P_j \in \mathcal{Q}$ can compute its individual group key pair $\langle gsk_j, gpk_j \rangle$:

$$gsk_j = \sum_{P_i \in \mathcal{Q}} s_{i \to j} \qquad gpk_j = h^{gsk_j} \tag{3.8}$$

In order to enable a third party to verify gpk_j, P_j provides the values g^{gsk_j} as well as a correctness proof $\mathsf{DLEQ}(g, g^{gsk_j}, h, gpk_j, gsk_j)$. The verifier accepts gpk_j as valid if

[6] See [64, 65] for an in-depth discussion on the implications of non-uniform distribution.

checking of the proof via DLEQ-verify$(\cdot)$ succeeds, and the verification of g^{gsk_j} using the previously committed public polynomials is successful:

$$g^{gsk_j} = \prod_{P_i \in \mathcal{Q}} F_i(j) \tag{3.9}$$

The corresponding master secret key msk is shared among all nodes in $\mathcal{Q}$ and can be obtained as follows:

$$msk = \sum_{P_i \in \mathcal{Q}} s_i \tag{3.10}$$

In case P_i does not reveal its secret s_i, it can always be computed by $t+1$ collaborating parties, because each $P_i \in \mathcal{Q}$ has correctly shared s_i among the parties during the first protocol phase. Alternatively, a set of $t+1$ collaborating parties, denoted by $\mathcal{R}$, can also derive the master secret key msk via Lagrange interpolation from their group secret keys:

$$msk = \sum_{P_j \in \mathcal{R}} gsk_j \prod_{\substack{P_k \in \mathcal{R} \\ j \neq k}} \frac{k}{k-j} \tag{3.11}$$

However, for many threshold cryptographic applications msk might never be computed at a single location. Considering, e.g. BLS threshold signatures, $t+1$ collaborating parties might produce a signature σ on message m which verifies under the public key mpk. For this purpose, each of these parties P_j uses its individual group signing key gsk_j to issue a partial signature for m, which upon aggregation form σ. There is no need to compute the master secret key msk in order to issue the signature in this scenario.

3.5 Security Analysis of the EthDKG Protocol

For brevity, we omit a detailed analysis of the guarantees in regard to *correctness* and *uniformity* in this chapter, as the corresponding security proofs provided by Gennaro et al. [63] and Neji et al. [87] directly apply to our protocol. We hence refer the reader to the aforementioned publications for further details.

Secrecy In order to show that the original security proof regarding secrecy still applies, we show that the dispute process we introduce as alternative to the steps described by Neji et al. [87] does not provide the adversary with any additional information, and hence preserves secrecy. Specifically, any information a correct node P_i secretly transfers to another correct node P_j must remain hidden from the adversary to ensure it cannot reconstruct the master secret key msk from those messages. The only point in time when information is exchanged secretly, is the share transmission step (see Section 3.4.1). Here a correct party P_i always encrypts the share $s_{i \to j}$ it sends to P_j using a symmetric key encryption algorithm $\mathsf{Enc}_{k_{ij}}(\cdot)$. Under the Computational Diffie-Hellman assumption, the shared key k_{ij} used for en-/decryption can only be derived using secret information

sk_i or sk_j from node P_i or P_j. However, neither P_i nor P_j reveal this information or k_{ij} itself during the protocol execution if they are both honest.

If we instead consider the case where P_i is honest but P_j is controlled by the adversary, the adversary also does not gain any additional information. In this case, the only point in time an honest node P_i would publish additional information, namely k_{ij} and the corresponding correctness proof $\pi(k_{ij}) = \mathsf{DLEQ}(g, pk_i, pk_j, k_{ij}, sk_i)$, is during the process of issuing a dispute claim (Section 3.4.2). However, being the intended communication partner, the adversary was already able to derive $k_{ij} = {pk_i}^{sk_j}$ (and thus obtain $s_{i\to j}$) as part of the protocol. Hence, no additional information is revealed when P_i publishes k_{ij}, Furthermore, e.g. as outlined by Camenisch and Stadler [35], the NIZK proof $\pi(k_{ij})$ does not reveal any information in addition to correctness of k_{ij}, in particular does not reveal any information about sk_i.

Robustness Robustness requires an efficient procedure, that recovers the master secret key msk from a set of at least $t+1$ correct shares. However, this set may additionally contain up to t invalid shares provided by the adversary. We obtain such a procedure, by first checking the validity of a provided share gsk_i using the verification condition specified in Equation 3.9. Lagrange interpolation is then used to compute msk from any set of $t+1$ valid shares (see Equation 3.11).

Liveness In our synchronous system model, the protocol always reaches the beginning of the key derivation phase, as the sharing and dispute phases always end after a fixed amount of steps (the respective number of blocks per phase). Consequently, the completion of the key derivation phase (and thus the completion of the protocol), depends on the nodes' ability to gather all the information required to compute mpk from the values h^{s_i} provided by all $P_i \in \mathcal{Q}$. Each correct node in the set of qualified nodes $\mathcal{Q}$, publishes this value at the beginning of the phase. However, up to t adversarial nodes, which completed the sharing and dispute phase successfully, and are thus part of $\mathcal{Q}$, might not reveal the respective values. In this case, the correct parties obtain all missing values h^{s_i} by recovering s_i using Lagrange interpolation from their shares for s_i (see Section 3.4.3 for additional details). This process requires the collaboration of at least $t+1$ correct nodes, and thus completes successfully for configurations where the adversary controls at most $n-t-1$ nodes.

3.6 Implementation of the EthDKG Protocol

To highlight the feasibility and practicality of our approach, we present a prototype implementation. It consists of two parts: (i) an Ethereum smart contract serving as the communication and verification platform, and (ii) a client application written in Python and executed locally by each participant. Both implementations are open source and publicly available on Github https://github.com/PhilippSchindler/EthDKG/.

In the following, we describe the steps required to apply our generalized protocol description for the concrete use case of deriving key pairs to be used with the BLS signature scheme. Thereby, we outline (i) how our communication model can be realized, (ii) which techniques are necessary to efficiently implement the required cryptographic primitives, and (iii) how the protocol execution can be verified at the smart contract level, despite the limitations of the Ethereum platform. The BLS signature scheme was chosen not only because Ethereum has built-in support for a pairing friendly elliptic curve which can be used with BLS, but also due to the wide range of desirable properties this signature scheme provides for different application scenarios. These properties include short signature size, non-interactive aggregation capabilities as well as signature uniqueness. For additional details on BLS signatures, their properties and use cases we refer the reader to the original descriptions [24, 23, 20].

When using our protocol for BLS signatures, a set of parties first executes our DKG protocol to compute a master BLS key pair $\langle msk, mpk \rangle$. The public key mpk is published and verified within the smart contract, whereas the (virtual) secret key msk is shared among the parties. Each party P_i is then capable of using its individual signing key gsk_i to sign messages with BLS. Any set of $t + 1$ valid[7] signatures on a common message can be combined to form a threshold signature, which verifies under mpk, for that message. This aggregation process can be performed without necessitating on-chain transactions within Ethereum. Furthermore, the cost of verifying the resulting threshold signature within the smart contract does not depend on the number of participants or signers.

3.6.1 Realizing our Communication Model

Revisiting the assumptions from our protocol description (see Section 3.3.1), we require a shared agreed-upon authenticated broadcast channel and adherence to certain synchrony assumptions to separate the different protocol phases. These assumptions are realized as follows:

Ethereum as a Broadcast Channel In our implementation, each participant of the DKG protocol actively monitors the Ethereum blockchain. In particular, clients monitor all transactions to the address of the pre-deployed DKG contract. A message is broadcast by issuing a transaction that calls a function within the DKG smart contract when the transaction is mined within a block in the Ethereum network. Upon being called successfully, the contract triggers Ethereum events, which are processed by the client implementation.

[7] The process is robust in the sense that the validity of an individual signature can also be checked using the issuer's public key.

Agreement After detecting the emission of a new event, the client software of each participant waits for a sufficient number[8] of confirming blocks. This ensures that all nodes agree on a common history of blocks, and consequently on the triggered events and their order w.h.p, before they react to the events. This requirement is a direct consequence of the fact that the Ethereum blockchain may fork and thus does not provide immediate agreement on newly mined blocks.

Message Authentication The requirements in regard to message authenticity are directly supported by Ethereum. In fact, Ethereum enforces that all transactions are cryptographically signed by the issuer in order to be processed.

Synchrony Assumptions Our synchrony assumptions can be realized by specifying the start and end of each protocol phase based on appropriate relative Ethereum block heights. Liveness, i.e. ensuring the protocol completes successfully even under adversarial conditions, critically depends on the ability of correct nodes to timely disseminate information. Consequently, it has to be ensured that any transaction a node issues at the beginning of a protocol phase is confirmed, and consequently received by all other correct nodes, by the beginning of the next phase. The required phase durations depend on a range of factors including: the number of participants, the state of the Ethereum network, and the amount of transaction fees the participants are willing to pay. Thus they need to be analyzed on a case by case basis or selected conservatively. We provide an evaluation in regards to the required durations, considering the network conditions at the time of writing as well as a general description, in Section 3.7.

3.6.2 Cryptographic Primitives

When leveraging a smart contract-based DKG implementation that is capable of performing the verification steps on-chain, an efficient implementation of the underlying cryptographic primitives can be crucial for a low cost protocol design. Within the Ethereum platform, only a limited range of so called pre-compiled contracts for elliptic curve cryptography are available currently. The supported operations target the groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order q, defined on the elliptic curve BN254 [15, 3] and include point/point addition ($\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_1$), point/scalar multiplication ($\mathbb{G}_1 \times \mathbb{Z}_q \to \mathbb{G}_1$) and a verification procedure for the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. We rely upon these operations to efficiently implement the verification procedures for our DKG, targeting the generation of keys for the BLS signature scheme.

As BLS public keys reside in $\mathbb{G}_2$, most of the operations required for our protocol would use group $\mathbb{G}_2$, if we directly apply our general protocol description. However, as of the current Ethereum release, computations in $\mathbb{G}_2$ are not natively supported,

[8] For an in depth discussion on the required number of confirmations we refer to the works of Gervais et al. [67] and Sompolinsky and Zohar [112]. We furthermore provide concrete values for this manner in our evaluation (see Section 3.7.3).

and implementing the required operations using available Ethereum Virtual Machine (EVM) opcodes would lead to very high gas consumption and thus render the approach inefficient.[9] Fortunately, the corresponding operations in group $\mathbb{G}_1$ and a verification procedure for the pairing e exist as pre-compiled contracts in Ethereum [100, 31]. This allows us to efficiently perform operations in $\mathbb{G}_1$ and verify the corresponding element in $\mathbb{G}_2$ using the pairing check within the smart contract. In the following sections 3.6.3, 3.6.4 and 3.6.5, we outline the details for incorporating this approach into our protocol design.

3.6.3 Sharing Phase

During the sharing phase, each participant $P_i \in \mathcal{P}$ proceeds as specified in our general protocol description (see Section 3.4.1). In particular, P_i shares a secret $s_i \in_R \mathbb{Z}_q$ among all parties in $\mathcal{P}$ using Feldman's VSS protocol. The commitments $C_{i0}, C_{i1}, ..., C_{it}$ are group elements from $\mathbb{G}_1$: $C_{ik} = g_1^{c_{ik}} \mid 0 \leq k \leq t$, where g_1 denotes a generator of $\mathbb{G}_1$. Because there are no primitives for symmetric encryption available within Ethereum, we realize the encryption and decryption algorithms $\mathsf{Enc}_{k_{ij}}(\cdot)$ and $\mathsf{Dec}_{k_{ij}}(\cdot)$ using a one time pad, where we derive a unique key from k_{ij} and j by using a cryptographic hash function[10] $\mathsf{H}(\cdot)$:

$$\mathsf{Enc}_{k_{ij}}(s_{i\to j}) = s_{i\to j} \oplus \mathsf{H}(k_{ij} \mid\mid j)$$
$$\mathsf{Dec}_{k_{ij}}(\overline{s_{i\to j}}) = \overline{s_{i\to j}} \oplus \mathsf{H}(k_{ij} \mid\mid j)$$

To ensure that such a simple approach is secure in practice, it is crucial that (i) the pads used for encryption of messages between honest parties are indeed used only once, and (ii) the encrypted data is additionally protected against malleability. For two distinct honest parties P_i and P_j, the value of k_{ij} is defined by the values of the randomly generated private keys sk_i and sk_j, and is thus unique. Combining this unique value with the index of the share receiver j further ensures that the one time pads used to encrypt the single message from P_i to P_j and the single message from P_j to P_i are encrypted with different pads. Consequently, criterion (i) is met. Also criterion (ii) is fulfilled, as the encrypted values are transmitted as part of Ethereum transactions, which are signed and published on the broadcast channel and thus protected against malleability. To publish the required information, namely the encrypted shares $\overline{s_{i\to j}}$ for all $i \neq j$ and the commitments $C_{i0}, C_{i1}, ..., C_{it}$, the client constructs and broadcasts the corresponding Ethereum transaction, invoking the pre-deployed smart contract.

The smart contract ensures that only *eligible* parties, i.e. $P_i \in \mathcal{P}$ may provide a *single*, *well-formed* message. The set of eligible parties is either specified statically at the time of creation of the smart contract, or via a dynamic registration process as described in

[9] A Solidity implementation of a single multiplication of a group element from $\mathbb{G}_2$ with a 256 bit scalar requires approximately 2 000 000 gas [2],

[10] In our implementation, the value $s_{i\to j}$ and the output of the used cryptographic hash function are 256 bits each.

Section 3.6.6. A message is considered well-formed, if it contains exactly $n-1$ encrypted shares, and $t+1$ commitments to the coefficients of the secret sharing polynomial. Upon receiving a well-formed transaction from an eligible party, the smart contract notifies all other participants about the published information using an Ethereum event. The contents of the encrypted shares and the validity of the commitments are not verified at this point in time. Instead, the verification is only performed on demand, i.e. in case a dispute is submitted in the next protocol phase. In order to verify a potential dispute in the next phase, the smart contract stores a cryptographic hash of the message content. As we see in Section 3.6.4, the hash is sufficient to fully verify a potential dispute. It would also be possible to store the entire message instead of the digest. However, storing only the hash significantly reduces the amount of on-chain storage required, and thus lowers transactions fees, in particular for large n.

3.6.4 Dispute Phase

In case a party P_j finds that P_i provided an invalid share for s_i, P_j follows our general protocol description to publish a dispute. For this purpose, it constructs a transaction which, in addition to k_{ij} and $\pi(k_{ij})$, includes the message content sent by P_i in the previous protocol phase. This enables the smart contract to recompute and compare the hash of P_i's message with the stored value. If the hashes do not match, the dispute is found invalid and the smart contract aborts. Otherwise the smart contract has all information required to perform a full verification. In particular, it can verify that the encrypted share $\overline{s_{i\to j}}$ present in the dispute transaction is indeed the share P_i previously distributed. The verification continues as stated in Section 3.4.1. The corresponding computations can efficiently be performed using the Ethereum pre-compiled contracts [100] for arithmetic in $\mathbb{G}_1$. If the dispute is considered valid, the share issuer is flagged as adversarial and thus excluded from the set $\mathcal{Q}$ in the key derivation phase. Additionally, the smart contract triggers a corresponding event to notify all parties about the successful dispute. Optionally the issuer may be economically punished, and a security deposit could be used to refund the disputer for its transaction fees. Similarly, an adversarial disputer could be penalized for submitting an invalid dispute. In either case, the contract may not process a dispute transaction against an already disqualified participant. In fact, in this scenario, our implementation of the smart contract aborts immediately in order to save transaction fees.

3.6.5 Key Derivation Phase

Again, we closely follow our protocol specification from Section 3.4.3 to implement the key derivation phase. Similar to the definition of h, we use $h_1 \in \mathbb{G}_1$ and $h_2 \in \mathbb{G}_2$ to denote independently selected generators for the groups $\mathbb{G}_1$ and $\mathbb{G}_2$.

As a first step, each $P_i \in \mathcal{Q}$ computes the values ${h_1}^{s_i}$ and the corresponding NIZK proof $\pi({h_1}^{s_i})$ showing its correctness. The corresponding computations are performed in group $\mathbb{G}_1$. However, as the master public key $mpk = {h_2}^{msk}$ is an element of $\mathbb{G}_2$, P_i

is also required to map its key share ${h_1}^{s_i}$ to $\mathbb{G}_2$, i.e. compute $h_2^{s_i}$. Then, P_i crafts and publishes a transaction, containing ${h_1}^{s_i}$, $\pi({h_1}^{s_i})$ and $h_2^{s_i}$. As described, a collaboration of $t+1$ parties recovers s_i (and thus ${h_1}^{s_i}$, $\pi({h_1}^{s_i})$ and $h_2^{s_i}$) in case P_i does no publish the required information by the end of the key derivation phase. This recovery process can be performed either with or without interaction with the Ethereum platform. We opted for using Ethereum for this purpose, instead of adding complexity to the design by implementing an additional off-chain communication channel. After completing the recovery, any one of the involved parties can issue the corresponding transaction on behalf of P_i. Either way, it is ensured that ${h_1}^{s_i}$, $\pi({h_1}^{s_i})$ and $h_2^{s_i}$ become public and available for the smart contract for all $P_i \in \mathcal{Q}$. The smart contract can verify the provided information with the DLEQ-verify$(\cdot)$ procedure and use the precompiled pairing contract [31] to check the validity of ${h_2}^{s_i}$. The value ${h_2}^{s_i}$ is considered correct if $e({h_1}^{s_i}, h_2) = e(h_1, {h_2}^{s_i})$ holds.

Finally, any party can compute and publish the master public key $mpk = \prod_{P_i \in \mathcal{Q}} h_2^{s_i}$ and $mpk^* = \prod_{P_i \in \mathcal{Q}} h_1^{s_i}$. The smart contract can recompute mpk^* and use the pairing $e(\cdot)$ to verify the correctness of mpk.

3.6.6 Dynamic Participation

The utilization of an open smart contract platform such as Ethereum also enables us to readily implement dynamic participation strategies. If the choice is made to employ this protocol feature, the set of participants $\mathcal{P}$ which run the DKG protocol is not defined a priori, but rather obtained in an additional *registration phase*, executed at the beginning of the protocol. For this purpose, the creator of the corresponding smart contract specifies a set of participation rules at the time of contract creation. A participation rule specifies under which condition a particular Ethereum account is allowed to "join" the set $\mathcal{P}$. Within the limitations of the Ethereum platform, arbitrary smart contract code can be used to define participation rules. In the following, we provide basic examples for participation rules while more elaborate and robust schemes against adversarial behavior are left to future work.

1. *First come, first serve*: Only the first N parties to register are allowed to join the protocol.
2. *Security deposit*: Only parties, which provide a security deposit of at least X Ether are allowed to join the protocol.
3. *Highest bidding*: The N parties, which provided the highest amount of security deposit are allowed to join the protocol.

For conditions 1 and 2 the participation rules are checked as soon as a registration transaction is included in an Ethereum block. Only upon success is the issuer of the transaction added to the set $\mathcal{P}$, tracked within the smart contract. The implementation of condition 3 is rendered slightly more complex. In this case, the smart contract keeps track of the set $\mathcal{P}$ consisting of up to N participants and their provided security deposits.

Upon registration of party P_{N+1}, the registration is accepted if the deposit provided is bigger than the smallest deposit received so far. If this is the case, the registration is accepted by adding P_{N+1} to the set $\mathcal{P}$ and removing the participant with the smallest deposit from $\mathcal{P}$. Otherwise the registration is rejected and $\mathcal{P}$ remains unchanged.

3.7 Evaluation of the EthDKG Protocol

The following paragraph provides a brief overview of our evaluation results, while our detailed findings are provided in the corresponding subsections: Section 3.7.1 (computational costs for all interaction between the parties and the smart contract), Section 3.7.2 (communication complexity), and Section 3.7.3 (execution time).

Even in demanding scenarios (tested with up to 256 nodes), each step in our implementation the DKG protocol can be verified at the smart contract level well within the Ethereum block gas limit. However, the overall costs of running the DKG protocol depends on various highly fluctuating factors such as the exchange rate of ETH to, e.g., USD, or the gas price depending on the current network load. Therefore, it is difficult to provide accurate execution cost estimates. For example, at the time we initially performed our evaluation, Ethereum gas prices of 2 GWei[11] were recommended, whereas at the time of writing 50 GWei are common. Combining the increase in gas prices with the increase in the ETH to USD exchange rate, our approach, while technically feasible, is currently rendered costly for scenarios with a high number of nodes, whereas our original estimate of $1.68 per participant in a 256 node scenario, was of little practical concern. We note that our solution can also be deployed on other EVM compatible ledgers that currently offer markedly lower transaction fees compared to Ethereum.

3.7.1 Computational Costs

Figure 3.1 provides the measured gas consumption per executed transaction for different numbers of parties participating in the DKG protocol. We observe that (i) gas costs for contract deployment (2 551 221), for registration (106 385), and (recovered) key share submission (222 510) do not depend on the number of participants, (ii) costs for recovery linearly depend on the number of recovered parties, whereas (iii) the costs for the other operations increase linearly with increasing numbers of participants. Figure 3.1 reports the measured costs for (ii) and (iii) in the worst case for different numbers of participants (n). We use a setup with $n = 2t+1$ participants, where t parties are executing adversarial actions, i.e. they either provide invalid shares, handled by issuing dispute transactions, or they withhold the required values during the key generation phase, leading to a recovery of the missing information.

The most critical operations in terms of gas consumption are the execution of a dispute transaction (potentially executed once per adversarial node), and the submission and

[11] 1 GWei $= 10^{-9}$ ETH

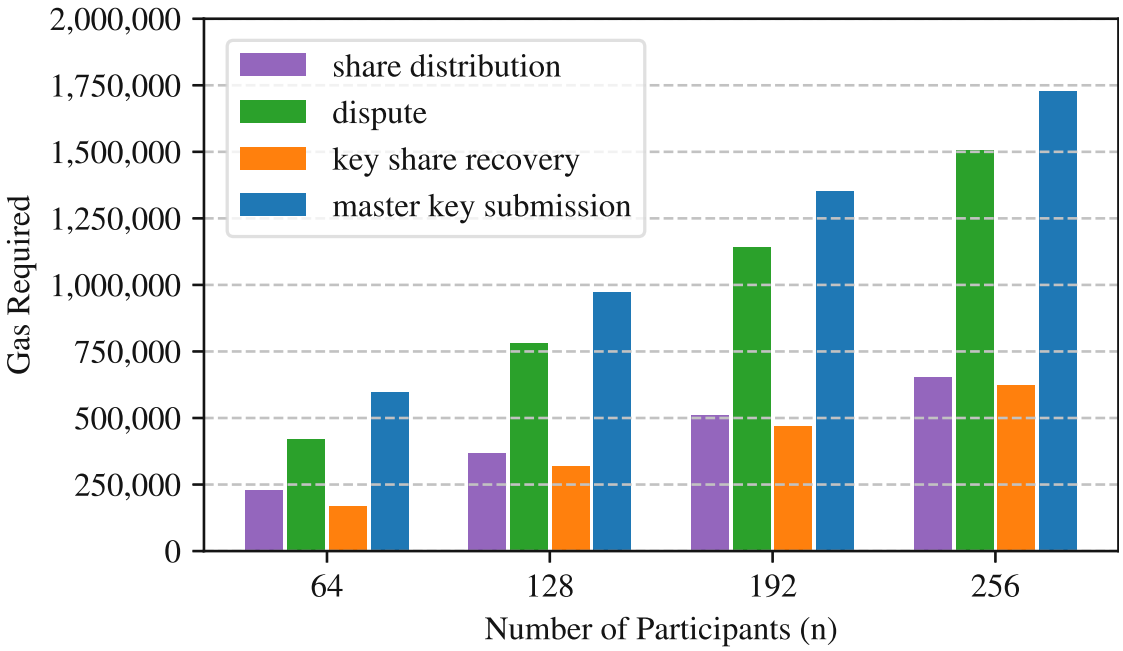

Figure 3.1: Computational costs, measured in gas per transaction, for the different types of interactions with the EthDKG smart contract

verification of the master public key (once at protocol end). In the most demanding scenario with $n = 256$ participants we evaluated, a dispute consumes approximately 1.5 million gas, whereas the master key submission requires around 1.7 million gas. In both cases, the costs are largely dominated by the internal verification procedures, relying on elliptic curve multiplications. Compared to the evaluation performed for an earlier version of this work, the gas costs for these operations have been significantly reduced due to implementation of the EIP-1108 proposal [37] as part of Ethereum's Istanbul hardfork. With these reductions, we tested our implementation with up to 256 participants and find that our protocol is able to perform all required operations well within the current Ethereum block gas limit of 15 000 000 gas [55].

In case all participants behave according to the protocol, no transactions for dispute, and key share recovery are executed. In the worst case, a dispute transaction has to be executed for each adversarial party in order to prove that the respective party violated the protocol rules. To avoid that correct parties need to cover the costs for the dispute transactions, a recommended mitigation strategy is to require security deposits during registration. The deposit from an adversarial party is then seized when a valid dispute is submitted, and used to refund the disputing party for the expenses incurred by publishing the dispute transaction. In case dispute transactions against the same party are issued concurrently, the fees for all but the first processed transaction are much cheaper, as the contract aborts prematurely. However, also in this case the additional costs may be covered by the adversary's security deposit. A different mitigation strategy is to reduce

number of nodes	8	16	32	64	128	192	256
base	0.14 $	0.15 $	0.16 $	0.18 $	0.23 $	0.28 $	0.32 $
dispute	0.03 $	0.05 $	0.08 $	0.14 $	0.26 $	0.38 $	0.50 $
key share recovery	0.01 $	0.01 $	0.01 $	0.01 $	0.01 $	0.01 $	0.01 $
master key verification	0.09 $	0.10 $	0.13 $	0.20 $	0.32 $	0.45 $	0.57 $
deployment	0.84 $	0.84 $	0.84 $	0.84 $	0.84 $	0.84 $	0.84 $

Table 3.1: Estimated transaction fees for EthDKG at the time of initial evaluation (2020-04-12; gas price: 2 GWei, exchange rate: 165 $ / ETH)

number of nodes	8	16	32	64	128	192	256
base	69 $	72 $	77 $	88 $	111 $	133 $	156 $
dispute	17 $	24 $	38 $	67 $	124 $	181 $	239 $
key share recovery	5 $	5 $	5 $	5 $	5 $	5 $	5 $
master key verification	42 $	50 $	65 $	94 $	154 $	214 $	274 $
deployment	405 $	405 $	405 $	405 $	405 $	405 $	405 $

Table 3.2: Estimated transaction fees for EthDKG at the time of writing (2021-08-22; gas price: 50 GWei; exchange rate: 3 175 $ / ETH)

the likelihood of concurrent submission of transactions by continuously monitoring for dispute transactions. In this case, dispute transactions are only issued on demand, i.e. in case there was no dispute against the specific party submitted yet, at randomized points in time within the bounds of the dispute phase. For many real world scenarios, in particular when the DKG is run between known entities, we expect the number of disputes to be very low if not zero. In this case, a high number of disputes would likely be addressed at an organization level and not within the protocol itself.

In order to keep costs for the share distribution low, we minimize the amount of data stored within the smart contract. In particular, we do not store the transaction data, i.e. $n-1$ encrypted shares and $t+1$ commitments to the secret sharing polynomial, in the smart contract. Instead, a cryptographic hash of the above information is stored, whereas triggering a corresponding Ethereum event renders the full data easily accessible to all clients. During the verification of a dispute, this cryptographic hash is recomputed and compared to the stored value to ensure that the disputer's information is correct.

To further illustrate the costs in practice, tables 3.1 and 3.2 provide a costs overview for running our protocol on the Ethereum platform by converting the gas consumption into USD. Hereby we group the estimated base costs (covering the registration, share distribution and key share submission steps) each joining party has to cover, and list

	transaction size	number of invocations	communication complexity (smart contract)	communication complexity (broadcast)
register	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
sharing	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
dispute	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
key share submission	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
key share recovery	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
recovered key share submission	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$
master key submission	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n^2)$

Table 3.3: Communication complexity for the different interactions types with the EthDKG smart contract

the costs for a dispute (per adversarial participant), a key share recovery (per failed participant after successful key sharing), as well as one-time costs (per DKG execution) for contract deployment and master key verification separately. Note that the costs are highly depended on the current gas price and Ethereum to USD exchange rate.

To reduce transaction fees, aside from choosing an alternative EVM compatible ledger with lower transaction fees, our protocol may also be adapted for layer 2 scaling solutions such as Plasma [95] and Arbitrum [74]. As a concrete example, the Matic/Polygon network [82] (an already deployed Plasma variant), reports current gas prices of 1 GWei at the time of writing [83]. This leads to greatly reduced transaction fees compared to a native execution on Ethereum.

3.7.2 Communication Complexity

Table 3.3 describes the size, the number of invocations, the total amount of data processed within the Ethereum blockchain, as well as the total amount of data transferred trough the network, for all the different transactions executed throughout a protocol execution. The reported values consider the worst case scenario, where the adversary sends invalid shares and fails to provide the required information during the key derivation phase. Overall, the communication complexity of our protocol is $\mathcal{O}(n^3)$, that is considering the network traffic generated by broadcasted all transactions. For the smart contract this is equivalent to a communication complexity of $\mathcal{O}(n^2)$, as the Ethereum client transparently handles the network communication. This distinction is crucial, as gas costs costs are only paid for the smart contract execution and are not dependent on the actual network traffic.

number of participants	64	128	192	256
estimated lower bound	61 min	73 min	91 min	115 min
estimated upper bound	85 min	100 min	123 min	153 min

Table 3.4: EthDKG protocol execution times for different numbers of participants

3.7.3 Execution Time

In the following, we estimate the total (worst case) execution time required to run our protocol. In practice, this execution time depends on a range of factors, including:

1. the number of confirmation required Δc, before a transaction is considered confirmed: ≈ 12 blocks [28]
2. Ethereum's (average) time between two subsequent blocks Δb: $13 - 17$ seconds [55, 27]
3. Ethereum's block gas limit c_{block}: $\approx 15\,000\,000$ gas [55]
4. the current load on the Ethereum network
5. the gas price participants are willing to pay
6. the number of parties executing the DKG protocol

Since all three protocol phases are executed subsequently, the total time required to execute the protocol T is the sum of the times required to execute each protocol phase. We use b_r, b_s, b_d and b_k to denote the number of blocks required to execute the protocol phases, *registration*, *sharing*, *dispute* and *key derivation* respectively. Consequently, we obtain T as follows:

$$T = \Delta b \cdot (b_r + b_s + b_d + b_k) \tag{3.12}$$

To compute the number of blocks required for each protocol phase, in particular b_r, b_s and b_d, we consider a consensus stabilization period at the end of each phase (Δc blocks), a safe upper bound for the number of blocks to wait until a transaction is included in the Ethereum blockchain (Δi), the capacity required to fit all transactions of the specific phase (c_r, c_s, c_d), as well as the maximum capacity c_{max} the DKG protocol should use on the Ethereum platform during execution (e.g. 10% of the block gas limit).

$$b_x = \Delta c + \Delta i + \left\lceil \frac{c_x}{c_{max}} \right\rceil \quad x \in \{r, s, d\} \tag{3.13}$$

Here, the capacities c_r, c_s and c_d are derived from the required gas for the specific transaction type, as given in Section 3.7.1 and the number of transactions executed.

Similarly, the values c_{k1}, c_{k2} and c_{k3} used below represent the capacities for the key share submission, key share recovery, and recovered key share submission transactions, respectively. As the key derivation phase requires multiple steps, b_k is computed by considering:

- the number of blocks required for the submission of key shares: $\Delta c + \Delta i + \left\lceil \frac{c_{k1}}{c_{max}} \right\rceil$
- the number of blocks required for a potential key share recovery: $\Delta c \Delta i \left\lceil \frac{c_{k2}}{c_{max}} \right\rceil$
- the number of blocks required for submission of the recovered key shares: $\Delta c \Delta i \left\lceil \frac{c_{k2}}{c_{max}} \right\rceil$
- as well as the number of blocks required for publishing the resulting master public key: $\Delta c \Delta i$.

In the following, we distinguish between the optimal case (no recovery) and the worst case (49% of all nodes need to be recovered) to get and lower and upper bound for b_k:

$$b_{k,min} = 2\Delta c + 2\Delta i + \left\lceil \frac{c_{k1}}{c_{max}} \right\rceil \tag{3.14}$$

$$b_{k,max} = 4\Delta c + 4\Delta i + \left\lceil \frac{c_{k1}}{c_{max}} \right\rceil + \left\lceil \frac{c_{k2}}{c_{max}} \right\rceil + \left\lceil \frac{c_{k3}}{c_{max}} \right\rceil \tag{3.15}$$

If we consider a worst case scenario with $n = 2t + 1$ participants, and select conservative values for the parameters above, i.e., we wait for $\Delta c = 20$ confirmations (≈ 4.4 minutes) before considered a transaction confirmed, assume the latency for transaction inclusion in a block is $\Delta i = 30$ blocks (≈ 6.6 minutes) and target a network load of 10% of Ethereum capacity ($c_{max} = 15\,000\,000$), and use Ethereum current block interval of $\Delta b = 13$ seconds [55] we obtain Table 3.4, summarizing the estimated execution times for different numbers of nodes.

3.8 Discussion and Comparison of EthDKG and Existing Distributed Key Generation Protocols

Model In our DKG protocol, we follow the model described in the theoretical works of Gennaro et al. [63]. Consequently, we inherit three important characteristics for our protocol: (i) the synchronous communication model, (ii) the separation of the underlying consensus platform and the DKG protocol itself, (iii) the optimal threshold t, i.e. secrecy and liveness for all $t < n/2$. These are in contrast to the properties of the more recent works by Kate et al. [76, 77], which consider an asynchronous communication model. While these works still require a weak synchrony assumption [39] to ensure liveness, the protocol's safety guarantees do not depend on timing assumptions of the underlying message delivery network. To mitigate this risk in a synchronous protocol design, the

corresponding timings, i.e. the number blocks in each protocol phase for our protocol, have to be selected appropriately.

A drawback of moving to the asynchronous model, is a reduced resilience against Byzantine adversaries. In the hybrid failure model ($n = 3t + 2f + 1$), described by Kate et al., the protocol can only tolerate less than 1/3 Byzantine parties (t), and less than 1/2 crashed participants (f). Here, our protocol design can prove advantageous as it ensures the desired security properties, in particular secrecy and liveness, with up to $n = 2t + 1$ participants.

Secrecy / Liveness Trade-off Our protocol design enables the use of different values for the parameter t, specifying the threshold for the underlying secret sharing protocol, depending on the specific application scenario. The choice of t directly incurs a trade-off between liveness and secrecy. If an adversary controls at most t nodes, secrecy is ensured, whereas at least $t+1$ honest nodes are required to guarantee liveness. For example, setting $t = n$, ensures that as long as there is at least one honest participant, the master secret key msk cannot be learned by the adversary. On the contrary, even a single adversarial node can prevent successful completion of the protocol. In practice the choice of t is directly related to the application scenario. If we consider, for example, a synchronous BFT protocol in a setting with $n = 2f + 1$ participants, t is set to equal f, whereas a typical requirement in asynchronous or particularly synchronous BFT protocols, i.e. that more than 2/3 of the parties have to sign a particular state or message, is supported by setting $t = \lceil 2/3n \rceil - 1$.

Uniform Key Distribution During the key derivation phase, we follow Neji et al. [87] to implement a protection mechanism, which prevents the adversary from biasing bits of the generated key pair. While the implemented countermeasure does not require a full additional secret sharing round, it requires up to two[12] additional transactions issued by all participants. To save these costs and reduce the protocol's complexity, one might decide to omit the additional steps required to ensure uniform distribution of the key pair. Instead, each party P_i publishes a commitment $H(C_{i0})$ to the value C_{i0} prior to the sharing phase. The values C_{i0}, published during the sharing phase, are only accepted if they match the corresponding commitment. During the key derivation phase, the master public key mpk is directly computed as described for the Joint-Feldman protocol (see Equation 3.5). Such a design decision may be useful e.g. in a deployment scenario, where we expect the DKG protocol to complete without any errors, i.e. in a scenario where we assume that it is very likely that all participants follow the protocol accordingly. However, as described in Section 3.7.1, the additional costs required to achieve uniformity do not add much overhead to the overall protocol execution. Consequently, we recommend to use our protocol design without this modification for most practical scenarios.

[12] one transaction for publishing the key share h^{s_i} and proof $\pi(h^{s_i})$, and potentially an additional message for recovering any missing key shares

Ethereum as Communication Infrastructure As described in Section 3.3.1, a key component necessary for the implementation of our DKG protocol is a suitable communication layer. Using an existing distributed ledger that provides Byzantine fault tolerance and agreed upon total ordering of exchanged messages. Although our approach may also be used on top of traditional BFT protocols or other available blockchain platforms, we decided to use an existing blockchain platform, namely Ethereum, instead of deploying our own communication infrastructure. If we compare our solution to the protocol described by Kate et al. we observe a key difference in the design approach: whereas in our protocol, the core functions of the DKG protocol are separated from the the underlying consensus mechanism, Kate et al. describe their protocol in a stand-alone setting, intertwining a custom BFT protocol with the DKG logic. We see advantages in both approaches, depending on the application scenario. While the technique we present can benefit from an easier deployment and a simplified protocol design due to the separation of concerns, the security of Kate et al.'s approach does not depend on an external consensus mechanism and can hence operate in a stand-alone setting.

On-Chain Verification While on-chain verification is not required for the core functionality of the protocol, it immediately provides a range of benefits: e.g. other applications on the Ethereum platform can be assured that the master public key was correctly computed, and can thus safely use this key to verify threshold signatures issued under the corresponding (shared) secret key. Furthermore, including monetary incentive mechanisms allows us to define a wide range of interesting dynamic and possibly open participation models. It is no longer required to define the set of parties $\mathcal{P}$, executing the protocol, prior to the protocol start. Instead, the smart contract logic can be used to specify under which conditions a party is allowed to join the protocol. When on-chain verification is not used, clients can still fully verify the protocol execution. However, the lack of on-chain verification also comes with the disadvantage, that seizing a security deposit becomes more difficult and potentially places honest clients at risk. It is no longer possible to seize the deposit automatically during the submission process of a dispute, as the smart contract does not perform the corresponding verification steps. A partial mitigation strategy is that a majority of the participants of the DKG verify a dispute off-chain and confirm its validity. However, this leads to the issue that an honest party's security deposit may be seized if the DKG protocol is run by an adversarial majority. This is in contrast to the approach with on-chain verification, which always ensures that the deposit of correct party remains safe.

Implementation and Scalability To the best of our knowledge, there exist no implementations of a DKG protocol following Gennaro et al.'s design, despite the extensive theoretical research in this direction. Our protocol can be seen as a first realization of this theoretical line of research. It is implemented and evaluated using the Ethereum platform as a communication layer. Consequently, the scalability of our approach is limited by the computational capacities available and transaction fees required to execute transactions on Ethereum. Our measurements (see Section 3.7.1) show that even in a

demanding scenario with 256 participants, all transactions can be executed well within Ethereum's current block gas limit. However, at the time of writing, the recent steep increase in gas cost in Ethereum due to its rise in popularity and price speculation has increased transaction the overall recommended network fees, introducing economic limitations especially for scenarios with a large number of nodes. Nevertheless, we expect fees to eventually return to lower levels as protocol improvements increase scalability and furthermore outline that our solution can be deployed on other EVM compatible ledgers with lower transaction fees.

The protocol design by Kate and Goldberg [76] was implemented and evaluated in subsequent work [77], performing tests of their implementation with up to 70 nodes on the PlanetLab platform. While in terms of execution time for small numbers of nodes, our solution is one order of magnitude slower than the completion times reported by Kate et al. [77], the parameters we use in our evaluation (see Section 3.7.3) are selected conservatively, and only use 10% of Ethereum's block capacity. Kate et al.'s protocol execution time increases sharply with an increasing number of nodes as the communication complexity of their protocol is $\mathcal{O}(n^4)$. Our evaluation shows that the communication complexity of our protocol is within $\mathcal{O}(n^3)$, while the amount of data processed on Ethereum is $\mathcal{O}(n^2)$. This leads to an approximate doubling in execution time when increasing the number of participants from 128 to 256.

3.9 Summary of our Findings on the EthDKG Protocol

In this chapter we present EthDKG, a new state of the art protocol for distributed key generation, that demonstrates how to efficiently implement an improved variant of Gennaro et al.'s [63] theoretical work. Our enhancements include a new mechanism to resolve disputes, which arise if certain parties violate the protocol rules, as well as a range of techniques improving the performance of our implementation in practice. We outline that our tailored protocol design can readily be executed on existing blockchain infrastructures. In particular, we show that all verification steps required during the protocol execution can be performed efficiently within the constrained EVM environment of the Ethereum platform. By leveraging the Ethereum blockchain, or an alternative platforms with similar guarantees, we are able to decouple the implementation of the underlying consensus protocol and the cryptographic components at the core of the DKG protocol itself. This approach simplifies the protocol design and security analysis, while at the same time enabling novel features, such as dynamic participation and support for economic incentives, by utilizing the capabilities of the Ethereum smart contract platform. As such, our protocol provides a versatile building block for a range of designs within and beyond the Ethereum ecosystem.

3.A Appendix: EthDKG Notation Reference

Table 3.5: EthDKG notation reference

Symbol	Description
n	total number of participants
t	secret sharing threshold, $t+1$ parties may recover the master secret key
$\mathcal{P}$	set of all parties $P_1, P_2, ..., P_n$
$\mathcal{Q}$	set of qualified parties, i.e., those how contribute to form the master key pair
$\mathcal{R}$	set of $t+1$ parties collaborating in a recovery process
$\langle sk_i, pk_i \rangle$	P_i's keypair
$\langle gsk_i, gpk_i \rangle$	P_i's group keypair, i.e., a share of the master key
$\langle msk, mpk \rangle$	master keypair, output of the DKG protocol (typically msk is not computed)
k_{ij}	shared key for symmetric encryption between parties P_i and P_j
$\mathbb{G}_q$	group of prime order q, discrete log is hard to compute in $\mathbb{G}_q$
g, h	two generators of the group $\mathbb{G}_q$, $dlog_g(h)$ is unknown
$\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$	groups of prime order q, for the use with BLS signatures
g_1, g_2, h_1, h_2	generators of the groups $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively
$e(\cdot)$	bilinear pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
$s_i = c_{i0}$	P_i's secret value shared with all parties in $\mathcal{P}$
$f_i(\cdot)$	P_i's secret polynomial
c_{ik}	coefficient of the polynomial $f_i(\cdot)$, $0 \leq k \leq t$
C_{ik}	commitment to the coefficient c_{ik}, $0 \leq k \leq t$
$s_{i \to j}$	P_i's share of s_i for P_j
$\overline{s_{i \to j}}$	P_i's encrypted share of s_i for P_j
$\mathsf{H}(\cdot)$	cryptographic hashfunction
$\mathsf{Enc}_{k_{ij}}(\cdot)$	symmetric key encryption algorithm
$\mathsf{Dec}_{k_{ij}}(\cdot)$	symmetric key decryption algorithm
$\pi(k_{ij}), \pi(h^{s_i})$	non-interactive zero-knowledge proof for the correctness of k_{ij} or h^{s_i} respectively
T	estimated protocol execution time in seconds
Δb	block interval (average number of seconds between two blocks)
Δc	number of confirmations (blocks) before a transaction is considered confirmed
Δi	number of blocks to wait until a transaction is included in a block
b_r, b_s, b_d, b_k	number of blocks required for the registration (b_r), sharing (b_s), dispute (b_d) or key derivation phase (b_k) respectively
c_{max}	maximum per block capacity (in units of gas) used by the protocol
c_r, c_s, c_d, c_k	capacity (in units of gas) required to execute all the transactions in the registration (c_r), sharing (c_s), dispute (c_d) or key derivation phase (c_k) respectively

CHAPTER 4

RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness*

As we have discussed in the prior chapters, generating randomness collectively has been a long standing problem in distributed computing. It plays a critical role not only in the design of state-of-the-art Byzantine fault-tolerant (BFT) and blockchain protocols, but also for a range of applications far beyond this field. With our protocols HydRand and EthDKG, we provide a standalone protocol to the provisioning of randomness in a distributed setting, as well as a new practical distributed key generation protocol, which among its many applications is essential for the setup of threshold signature based randomness beacons.

In this chapter, we present *RandRunner*, our second novel random beacon protocol with a unique set of guarantees. RandRunner shares the advantage of low communication complexity provided by threshold signatures based randomness beacons compared to HydRand, but does not introduce the trade-off they require. In particular, RandRunner operates in the well studied RSA setting and does not require an interactive protocol during setup. Our design also avoids the necessity of a (BFT) consensus protocol and its accompanying complexity and communication overhead. This makes RandRunner an excellent choice for deployment scenarios where low communication complexity, a high number of nodes, or a highly dynamic set of nodes are key. To achieve these properties, we introduce a novel extension to verifiable delay functions (VDFs) and base our protocol design on this newly obtained cryptographic primitive. The extension

*This chapter is an updated version of the equally-named research paper [103] Network and Distributed System Security Symposium (NDSS) 2021. Large text passages from the original work are used in verbatim form in this work.

ensures the uniqueness of the VDFs outputs against adversaries which are in possession of a trapdoor for the used VDF. This novel design allows RandRunner to tolerate adversarial or failed leaders while guaranteeing safety and liveness of the protocol despite possible periods of asynchrony.

4.1 Revisiting the State-of-the-Art in Distributed Randomness

Generating cryptographically secure randomness *locally* is essential for secure communication. While being a challenging topic in itself, there exists a range of well established approaches to solve this problem. These range from direct support within modern operating systems, using a variety of different entropy sources, to dedicated CPU instructions or external hardware devices. However, as soon as randomness is not required on an individual basis but rather used *collectively*, local solutions fail to provide convincing evidence that some claimed random value was indeed derived randomly. Still, as outlined by the extensive body of prior works [106, 114, 38, 25], a broad range of applications relies on collectively used randomness. This includes the design of BFT and blockchain protocols, cryptographic parameter generation, e-voting, auditable selections, online gaming and gambling, privacy enhancing technologies, as well as Smart Contracts and other forms of multi-party computation. To address these scenarios, randomness from trusted third parties, for example, the NIST random beacon or random.org, may be used. However, the additional trust assumptions and reliance on a central randomness provider, which may know the beacon values well in advance before publishing, or could even manipulate the produced values without being detected, is undesirable. Fortunately, there exists a range of distributed protocols which can be used instead to avoid trusting centralized services.

The techniques used by modern protocols for distributed randomness generation have advanced significantly since *coin tossing protocols* and the notion of a *random beacon*, introduced by Blum [17] and Rabin [96] in 1983. As we compared in Section 2.7, modern techniques include threshold cryptography, in particular publicly-verifiable secret sharing (PVSS) [78, 38, 114, 106] and threshold signature schemes [33, 72], as well as verifiable random functions (VRFs) as seen in Algorand [42] and Ouroboros Praos [78]. Additionally, methods in which randomness is extracted from existing data sources such as the Bitcoin blockchain [25, 12, 93] or published financial data [44] have been considered. Methods based on delay functions (also known as slow-time functions) have been described [80] and were later realized via the Ethereum Smart Contract platform [32]. Recently, methods based on delay functions have received increased interest with the rise of verifiable delay functions (VDFs) [18, 19, 94, 118]. Although the characteristics of VDFs make them a promising candidate for their use in random beacon protocols, the number of protocols utilizing VDFs to construct random beacons is rather limited. To the best of our knowledge, there exists only RANDAO [30] which collects entropy from different parties to be used as input for a VDF, as proposed by J. Drake and discussed in the

online ethresear.ch forum [52]. Apart from this discussion, there has not been any formal security analysis of the scheme.

4.2 Introduction to RandRunner

With RandRunner we take on the challenge of exploring the use of VDFs of the construction of a modern distributed randomness beacon. Our results are positive. With our protocol we can successfully demonstrate that VDFs, specifically *strongly unique trapdoor VDFs* which we introduce in Section 4.3, can indeed be leveraged to construct a random beacon protocol with a unique set of security guarantees that also offers excellent scalability, performance and responsiveness. Our new protocol aims to fulfill all desirable properties previously considered for randomness beacons. These include the key properties of *unpredictability*, *bias-resistance*, *availability/liveness* as well as *public-verifiability*. In other words, an adversary must neither be able to predict future random beacons before they become publicly available, nor bias the distribution of the produced randomness, nor prevent the protocol from making progress. Furthermore, each produced protocol output must be efficiently verifiable even by third parties. As an extension to liveness and bias-resistance, we also set out to achieve the property of *guaranteed output delivery* [38, 106], ensuring that an adversary cannot even prevent the protocol from producing an output in any protocol round. In addition, RandRunner's construction and protocol description remains both simple to understand, as well as efficient (in terms of communication and verification complexity). To derive and prove the correctness of a fresh protocol output, only a single message, around 10 KB in size[1], has to be disseminated throughout the network of nodes running the protocol. The underlying message distribution mechanism is decoupled from the core protocol, providing the flexibility to adapt to a particular deployment scenario. For example, in large networks gossip protocols with communication complexity of $\mathcal{O}(n \log n)$ and higher latency may be used, while reliable broadcast with lower latency and complexity $\mathcal{O}(n^2)$ may suit smaller networks. By construction, our protocol ensures predetermined agreement on the sequence of random numbers produced without the necessity of continuous Byzantine agreement (BA). This also guarantees bias-resistance and public-verifiability and even allows for progress/liveness under periods of full asynchrony.

4.2.1 Contribution

Summarizing, the contributions of this chapter are as follows:

- We extend the concept of trapdoor verifiable delay functions (T-VDFs), as initially defined by Wesolowski [118], by formally defining the *strong uniqueness* property.
- We show how to instantiate T-VDFs that achieve this property and prove the security of our construction.

[1] This is essentially the proof size of the used VDF [94].

- Using a T-VDF with strong uniqueness as the main building block, we specify a new randomness beacon protocol called RandRunner and prove that it provides the desired security properties.
- We simulate the execution of our newly proposed protocol to demonstrate its practical feasibility under various scenarios and protocol configurations.
- We discuss and compare our solution to other state-of-the-art protocol designs.

4.2.2 Structure of the Remainder of this Chapter

In Section 4.3, we introduce the required background information on the topic of verifiable delay functions (VDFs), define trapdoor VDFs with the property of strong uniqueness, and show how to construct this type of VDF in practice. We provide an example and a first overview of the design of our randomness beacon, using the constructed VDF as the main cryptographic component, in Section 4.4, describe our system and threat model in Section 4.5, and give the details of our construction in Section 4.6. Section 4.7 presents our security proofs and simulation results for the protocol. Finally, we compare our design with existing state-of-the-art protocols in Section 4.8 and summarize our findings in Section 4.9. In Appendix 4.A at the end of this chapter, we provide additional evaluation results for a wide range of possible protocol parameterizations and scenarios to further highlight the feasibility of our approach in practice. A reference for the used notation is found in Appendix 4.B.

4.3 Trapdoor VDFs with Strong Uniqueness

In this section, we summarize the original concept of verifiable delay functions (VDFs) and define the exact requirements for a VDF serving as the main cryptographic component in our random beacon protocol: a trapdoor VDF with the strong uniqueness property[2]. We finally show how such a VDF can be constructed using standard cryptographic assumptions and provide the corresponding security proofs.

4.3.1 Background

VDFs were first introduced by Boneh et al. [18] in 2018, and have since received increased attention from other researchers (see, e.g., [18, 118, 94, 19, 57, 79, 111, 54, 51, 81]). As introduced by Boneh et al. [18, 19], a VDF is a function $f : \mathcal{X} \to \mathcal{Y}$ which maps every input $x \in \mathcal{X}$ to a *unique* output $y \in \mathcal{Y}$. Computing the VDF is *sequential* in the sense that it takes a prescribed amount of time, whether or not it is executed on multiple processors. Verification, on the other hand, should be as quick as possible. Closely following Boneh et al.'s definition [18], a VDF is described via a set of three algorithms:

[2] An interesting observation, noted by one anonymous reviewer, is that verifiable random functions (VRFs) [85] and these T-VDFs share the strong uniqueness property. The main difference is that (T-)VDFs are (slowly) publicly computable whereas VRFs are not.

$Setup(\lambda) \rightarrow pp$ is a randomized algorithm that takes a security parameter λ as input and outputs public parameters pp sampled from some parameter space $\mathcal{PP}$.

$Eval(pp, x, T) \rightarrow (y, \pi)$ takes public parameters $pp \in \mathcal{PP}$, an input $x \in \mathcal{X}$ and a time parameter $T \in \mathbb{N}$ and outputs a $y \in \mathcal{Y}$ together with a proof π.

$Verify(pp, x, T, y, \pi) \rightarrow \{accept, reject\}$ takes public parameters $pp \in \mathcal{PP}$, an input $x \in \mathcal{X}$, a time parameter $T \in \mathbb{N}$, the output $y \in \mathcal{Y}$ and the corresponding proof π and outputs *accept* if y is the correct evaluation of the VDF on input (pp, x, T) and *reject* otherwise.

Setup may require secret randomness as input. This secret input must not be recoverable by any party after *Setup* is completed, as knowledge of the secret randomness (depending on the construction) can be used to break the uniqueness and/or sequentiality properties of the VDF. In practice, e.g., in an RSA-based setting, this complicates the setup as the generation of the parameters requires either the use of a trusted dealer who has to delete the secret randomness after the process, or a rather complex secure multi-party computation.[3] As we will show, the trapdoor VDFs we use in RandRunner's protocol design avoid these disadvantages.

VDFs as introduced above have to satisfy certain properties, namely:

1. ϵ-evaluation time: a runtime constraint for *Eval*,
2. *sequentiality*: *Eval* must not be parallelizable, and
3. *uniqueness*: *Verify* must accept a single output per input (except with negligible probability).

We give the definition of these properties for the specific kind of VDF we require in the following and refer the reader to the excellent works of Boneh et al. [18, 19] and Pietrzak [94] for a formal definition of these properties in the general setting.

4.3.2 On Trapdoors and Strong Uniqueness

For our random beacon protocol, we require a special kind of VDF, namely a *trapdoor* VDF which ensures *strong uniqueness*. Regarding the corresponding definitions we give below, we closely follow Boneh et al.'s work [19] for the traditional setting. Whereas their work only considered public parameters generated by *Setup*, our definition covers all parameters from the parameter space $\mathcal{PP}$. All valid parameters, in particular all parameters generated by *Setup*, are part of $\mathcal{PP}$. However, $\mathcal{PP}$ is defined in terms of

[3] In a recent result, Frederiksen et al. [60] provide an implementation for the malicious two-party setting. Using server grade hardware connected via a 40.0 Gbps network link, they we able to achieve average runtimes of 35 seconds. However, we are not aware of any practical solutions for the malicious multi-party setting which would be desirable for the setup of VDFs.

the specific properties the parameters have to fulfill instead of being implicitly defined via *Setup*. This also allows us to reason about VDFs for which the parameters are generated by an adversary, and it is crucial for the definition of ϵ-evaluation time and *strong* uniqueness.

Trapdoor VDFs, as initially described by Wesolowski [118], are a modification and extension to traditional VDFs such that the *Setup* algorithm, in addition to the public parameters pp, outputs a secret key or trapdoor sk to the party invoking the setup algorithm. This parameter sk is kept secret by the invoker, whereas pp is published. Furthermore, the algorithm $TrapdoorEval(pp, x, T, sk) \rightarrow (y, \pi)$ provides an alternative way to evaluate the VDF efficiently, i.e., within time $\mathcal{O}(poly(\lambda))$, for parties which know the trapdoor sk. Parties without this knowledge, as in the traditional VDF case, can still compute the output by executing *Eval*. However, they require $(1 + \epsilon)T$ sequential computational steps to do so.

Definition 1. *(ϵ-evaluation time) For all inputs $x \in \mathcal{X}$ and all public parameters $pp \in \mathcal{PP}$, the algorithm $Eval(pp, x, T)$ runs in time at most $(1 + \epsilon)T$.*

Due to the introduction of the trapdoor, and in contrast to traditional VDFs, the sequentiality property only holds for parties which do not know the trapdoor, a property we make use of in the construction of our random beacon.

Definition 2. *(Sequentiality without trapdoor) A parallel algorithm $\mathcal{A}$, using at most $poly(\lambda)$ processors, that runs in time less than T cannot compute the function without the knowledge of a secret trapdoor sk. Specifically, for a random $x \in \mathcal{X}$ and all public parameters pp output by $Setup(\lambda)$, if (y, π) is the output of $Eval(pp, T, x)$, or $TrapdoorEval(pp, x, T, sk)$, then the probability that $\mathcal{A}$ can compute y in less than T steps is negligible.*

Strong uniqueness extends the requirement for uniqueness to a setting in which the public parameters of the VDF may be generated by an adversary. This setting was not considered in Wesolowski's paper [118], and unfortunately, Wesolowski's VDF also does not achieve this property. In their case, both uniqueness and sequentiality can be broken by an adversary knowing the trapdoor. We however envision a range of applications for trapdoor VDFs where this property is crucial. This includes, e.g., scenarios in which parties set up their VDF individually, as is the case with our randomness beacon.

Definition 3. *(Strong Uniqueness) For each input $x \in \mathcal{X}$, and all public parameters $pp \in \mathcal{PP}$, exactly one output $y \in \mathcal{Y}$ is accepted by Verify, with negligible error probability (even if the public parameters pp have been adversarially generated). Specifically, let $\mathcal{A}$ be an efficient algorithm that outputs (pp, x, T, y, π) such that $Verify(pp, x, T, y, \pi) = accept$. Then $Pr[Eval(pp, x, T) \neq y]$ is negligible.*

Notice that we follow Boneh et al.'s most recent definition of uniqueness [19], whereas uniqueness was previously implicitly defined by the properties of correctness and soundness [18].

4.3.3 Design Rationale

Efficient VDF designs, for example the protocols by Wesolowski [118] or Pietrzak [94], operate in groups of unknown order, such as the well known RSA groups or class groups of an imaginary quadratic field [26]. While the security of RSA groups has been studied for decades, the parameter setup for the VDF (i.e., computing the modulus N as the product of two safe primes) is considered difficult without requiring a trusted dealer. Class groups of an imaginary quadratic field do not require this trust assumption, but their security properties are less studied compared to the RSA case. With our protocol design, however, we show how we can leverage RSA-based VDFs without the trusted dealer requirements. This allows us to rely on well tested primitives, while avoiding additional trust assumptions.

The key motivation for the VDF design we use is that the party that sets up the VDF can always quickly compute it using the trapdoor generated during the setup. If this party fails to do so when required, any other party can step in and eventually obtain the same result by evaluating the VDF without the trapdoor. To construct a *trapdoor* VDF with *strong uniqueness* as outlined in Section 4.3.2, we rely on two components:

1. the VDF design by Pietrzak [94] in the RSA setting and
2. the zero-knowledge proof techniques for safe primes by Camenisch and Michaels [34], ensuring that an adversary cannot cheat during the VDF setup and consequently cannot break the uniqueness of the scheme.

On a high level, Pietrzak's VDF is based on the conjecture that for some random input $x \in \mathbb{Z}_N^*$ and RSA modulus $N = p \cdot q$, the computation of $y = x^{2^T} \pmod N$ requires T sequential squarings without knowledge of the factorization of N:

$$x \to x^2 \to x^{2^2} \to x^{2^3} \to ... \to x^{2^T} \pmod N, \tag{4.1}$$

an idea originally described in the context of time-lock puzzles by Rivest et al. [102]. The tuple (p, q) can be used as a trapdoor, because the knowledge of the group order $\phi(N) = (p-1)(q-1)$ enables one to efficiently compute y:

$$e = 2^t \pmod{\phi(N)}, \quad y = x^e \pmod N. \tag{4.2}$$

The construction of a trapdoor VDF from Pietrzak's VDF follows naturally, as the trapdoor is simply given by the primes p and q. In fact, the setup we use is actually simpler than in the non-trapdoor case, in which one has to assume a trusted dealer that generates N and later deletes p and q, or, alternatively, that N is generated without anyone knowing the factors using a multi-party computation. In our approach, the zero-knowledge proof techniques by Camenisch and Michaels [34] are used instead. They ensure that the assumptions for the original security proof of the uniqueness property of Pietrzak's VDF ([94], Theorem 1) are fulfilled, even if N is generated adversarially. Furthermore, these techniques only rely on common cryptographic assumptions, are quite efficient [34], and can be made non-interactive using the Fiat-Shamir heuristic [34, 58].

4.3.4 Construction

In the following, we describe the complete construction of a trapdoor VDF with strong uniqueness. We closely follow the definitions by Boneh et al. [18, 19] and Pietrzak [94] to define our VDF, mapping inputs $x \in \mathcal{X}$ to outputs $y \in \mathcal{Y}$, whereby $\mathcal{X} := QR_N^+$ and $\mathcal{Y} := QR_N^+$. Hereby, we use QR_N^+ to denote the group of signed quadratic residues modulo N (see [94], Section 2.2), and λ_{RSA} to denote a security parameter, specifying the length of the RSA modulus in bits, which offers at least λ bits of security[4]. The symbol π is used to represent a correctness proof of the evaluation of the VDF. It contains a list of intermediate values, which can be used to later check the result of the computation efficiently. Furthermore, let $\mathcal{PP} := \{pp \mid \mathit{VerifySetup}(\lambda, pp) = \mathit{accept}\}$ denote the space of all public parameters. Notice that *Eval*, *TrapdoorEval* and *Verify* are only defined for parameters $pp \in \mathcal{PP}$. In our random beacon protocol, we ensure that we only ever use VDFs with parameters $pp \in \mathcal{PP}$ by checking all public parameters once at the start of the protocol. The complete construction of our trapdoor VDF with strong uniqueness is as follows:

$\mathit{Setup}(\lambda) \rightarrow (pp, sk)$

1. Sample two random safe primes $p = 2p' + 1$ and $q = 2q' + 1$ of size $\lambda_{RSA}/2$, where p' and q' are prime and fulfill the following side-conditions required for the used proof techniques [34, 66]: $p, q, p', q' \not\equiv 1 \pmod 8$, $p \not\equiv q \pmod 8$, $p' \not\equiv q' \pmod 8$.

2. Run the zero-knowledge protocol for proving that a known N is the product of two safe primes ([34], Section 5.2) and the protocol "proving the knowledge of a discrete logarithm that lies in a given range" ([34], Section 2.2) to show that the prime factors p and q are $\lambda_{RSA}/2$ bits each. Let π_N denote the resulting proof obtained by running both protocols non-interactively using the Fiat-Shamir heuristic.

3. Return $pp := (N, \pi_N)$ as the public parameters and $sk := (p, q)$ as the secret key (trapdoor).

$\mathit{VerifySetup}(\lambda, pp) \rightarrow \{\mathit{accept}, \mathit{reject}\}$
Return *accept* if the validity of pp can be successfully checked by using the verification procedures corresponding to the proof techniques used in step 2) of *Setup* as specified by Camenisch and Michaels [34]. Return *reject* otherwise.

$\mathit{Eval}(pp, x, T) \rightarrow (y, \pi)$
Run the evaluation algorithm $\mathit{VDF.Sol}(N, (x, T)) \rightarrow (y, \pi)$ as originally defined by Pietrzak ([94], Section 6) and return its result.

[4]Typical choices for λ_{RSA} are between 2048 and 4096 bits. See e.g., https://www.keylength.com/ for a comparison of different recommendations.

TrapdoorEval$(pp, x, T, sk) \rightarrow (y, \pi)$
Derive the group order $\phi(N) = (p-1)(q-1)$ from the secret trapdoor $sk \coloneqq (p, q)$ and execute the evaluation algorithm $VDF.Sol(N, (x, T)) \rightarrow (y, \pi)$ efficiently. As illustrated in Equation 4.2, the result $y = x^{2^T}$ as well as the values required for the proof π can be computed efficiently by reducing large exponents in the computations modulo $\phi(N)$.

Verify$(pp, x, T, y, \pi) \rightarrow \{accept, reject\}$
Return the result of the verification algorithm $VDF.Ver(N, (x, T), (y, \pi))$ as originally defined by Pietrzak ([94], Section 6).

4.3.5 Security Assumptions

We inherit the security assumptions from (i) Pietrzak's VDF [94] in the RSA setting as well as (ii) for the proof techniques from Camenisch et al. [34]. Consequently, we assume:

- Factoring N is hard.
- Computing x^{2^T} is sequential in (QR_N^+, x), where x is a generator[5].
- The existence of groups $G = \langle g \rangle$ of large known order Q and a generator h, where computing discrete logarithms is hard and the value of $dlog_g(h)$ is unknown.
- Hash functions are modeled as Random Oracles [10].

4.3.6 Security Proof

In this section we show that our construction of a trapdoor VDF with strong uniqueness achieves the required security properties, i.e., ϵ-evaluation time, sequentiality without trapdoor, and strong uniqueness. Therefore, the security proof of our trapdoor VDF construction extends the security proof provided by Pietrzak [94] for the underlying VDF. As the properties of ϵ-evaluation time and sequentiality (without trapdoor) are not affected by our extension to the trapdoor setting, we focus on showing that our construction indeed achieves strong uniqueness. We prove that this property is achieved by first revisiting Pietrzak's original security statement for uniqueness, and then show how our construction ensures all preconditions required to apply the original proof in our setting.

Theorem 6. *As given in [94]. If the input (N, x, T) to the protocol satisfies*

1. *$N = p \cdot q$ is the product of two safe primes, i.e., $p = 2p' + 1$, $q = 2q' + 1$ for primes p', q'.*

[5] As Pietrzak ([94], Section 2.2) shows, this assumption is essentially equal to the sequentiality assumption of the RSA time-lock puzzle [102] in $(\mathbb{Z}_N^*, \cdot)$.

2. $\langle x \rangle = QR_N^+$.[6]

3. $2^\lambda \leq min\{p', q'\}$.

Then for any malicious prover $\tilde{\mathcal{P}}$ who sends as first message y anything else than the solution to the RSW time-lock puzzle, i.e., $y \neq x^{2^T}$ [a verifier] $\mathcal{V}$ will finally output accept *with probability at most $\frac{3log(T)}{2^\lambda}$.*

The security proof of the above statement ([94], Section 4) shows that Pietrzak's VDF achieves uniqueness. For uniqueness to hold in the original model, only the case in which the public parameters pp (i.e., N in this setting) are generated by *Setup* have to be considered. In this case, N and p', q' satisfy conditions 1) by construction and 3) for all reasonable choices of λ_{RSA} as $\lambda_{RSA} \gg \lambda$. Condition 2) is met because almost every $x \in QR_N^+$ generates QR_N^+. A trivial exception is 1, which can easily be checked for, and some hard to find[7] elements of order p' or q'.

For *strong* uniqueness, however, the uniqueness property needs to hold for all public parameters $pp \in \mathcal{PP}$. Consequently, we need to show that conditions 1), 2) and 3) still hold, in particular without restricting pp to be generated by *Setup*.

Lemma 8. *For all public parameters $pp \in \mathcal{PP}$ and random inputs $x \in QR_N^+$ the protocol described in Section 4.3.4 ensures that conditions 1), 2) and 3) as required by Theorem 6 are satisfied.*

Proof. Recall that $\mathcal{PP} := \{pp \mid VerifySetup(\lambda, pp) = accept\}$. Since *VerifySetup* only accepts pp after running the verification technique from Camenisch et al. [34], which shows that (i) N is the product of two safe primes and (ii) p and q are of size $\lambda_{RSA}/2$ conditions 1) and 3) are satisfied. Since $N = p \cdot q$ is the product of two safe primes $p = 2p' + 1$, $q = 2q' + 1$, the group QR_N^+ of size $p'q'$ contains only $1 + (p' - 1) + (q' - 1)$ elements which do not generate QR_N^+. Consequently, the probability of picking such a small order element at random, i.e., $\frac{1+(p'-1)+(q'-1)}{p'q'}$, is negligible and thus satisfies condition 2) for random inputs. □

Regarding condition 2), we note that for the application within our randomness beacon we only use random inputs, therefore the probability of randomly generating a problematic value is negligible in this case. However, for applications in which the adversary can freely select a particular value x, it can be a problem to ensure that condition 2) indeed holds in all cases. An efficient procedure to check this property in this setting was stated

[6] That is, x generates QR_N^+, the [signed] quadratic residues modulo N. For our choice of N we have $|QR_N^+| = |QR_N| = p'q'$, so $\langle x \rangle := \{x, x^2, ..., x^{p'q'}\} = QR_N^+$. [94]

[7] The probability of finding such elements, without knowing the factors of N, is negligible since there are only $p' - 1$ or $q' - 1$ elements of order p' or q' respectively, whereas QR_N^+ contains $p'q'$ elements.

as an open problem in Pietrzak's work [94]. With the following formula, we provide an efficient way to verify if x is indeed a generator of QR_N^+, thereby describing a method to check if condition 2) holds for *all* inputs instead of requiring random inputs:

$$\langle x \rangle = QR_N^+ \text{ if } x \in QR_N^+ \wedge gcd(x^2 - 1, N) = 1\,. \tag{4.3}$$

A short proof of the above statement is presented in the following.

Proof. We show the above statement by deriving a contradiction. Assume that x does not generate the group QR_N^+, i.e., $\langle x \rangle \neq QR_N^+$. This means that the order of x in QR_N^+ is not equal to $p'q'$. One easily verifies that we may write $x = a^{p'} \mod N$ or $x = a^{q'} \mod N$ for some a. This implies $x^2 = 1 \mod p$ or $x^2 = 1 \mod q$, hence the $gcd(x^2 - 1, N)$ in statement 4.3 cannot be 1. □

Note that membership in QR_N^+ is also efficiently decidable by computing the Jacobi symbol of x modulo N (see Section 2.1 in [94]).

4.4 Conceptual Design of the RandRunner Protocol

RandRunner is a distributed randomness beacon which relies on trapdoor VDFs with strong uniqueness, previously introduced in Section 4.3, as the key cryptographic building block. These VDFs are set up prior to the start of the protocol. In particular, each party running the protocol is responsible for the initialization of its individual VDF. It keeps the trapdoor generated during setup secret, while making the verification parameters and the cryptographic proof of the setup's correctness publicly available.

Following this initial protocol setup, the main protocol execution can start. The execution of the protocol proceeds in consecutive rounds. At the end of each round a fresh random beacon output is produced. In the common case, the protocol is driven one step/round forward, as a dedicated party – a leader which changes every round – uses its trapdoor to evaluate its VDF based on the previous random beacon output. The leader initiates a broadcast of the result together with a short correctness proof which enables all parties to verify and complete the current round. In case of an attack, a malicious or failed leader, or network issues, the protocol can still advance, as all parties are able to evaluate the VDF of the current round without the trapdoor. This is further illustrated in Figure 4.1, showing a protocol execution with three nodes. In this example, the sequence of leaders $(n_1, n_2, n_3, n_1, \dots)$ is derived in a round-robin fashion. In the rounds r_1 and r_2, the respective leaders evaluate the VDFs and send the results to all parties – the protocol progresses quickly. In the third round r_3, the leader n_3 fails to forward the result to the other parties. Therefore, nodes n_1 and n_2 are slowed down as they are required to evaluate this round's VDF without the trapdoor. In the meantime, node n_3 already starts computing the result of the following rounds, but the other nodes catch up, because in round r_4 and r_5 node n_3 has to compute the VDFs without the trapdoors.

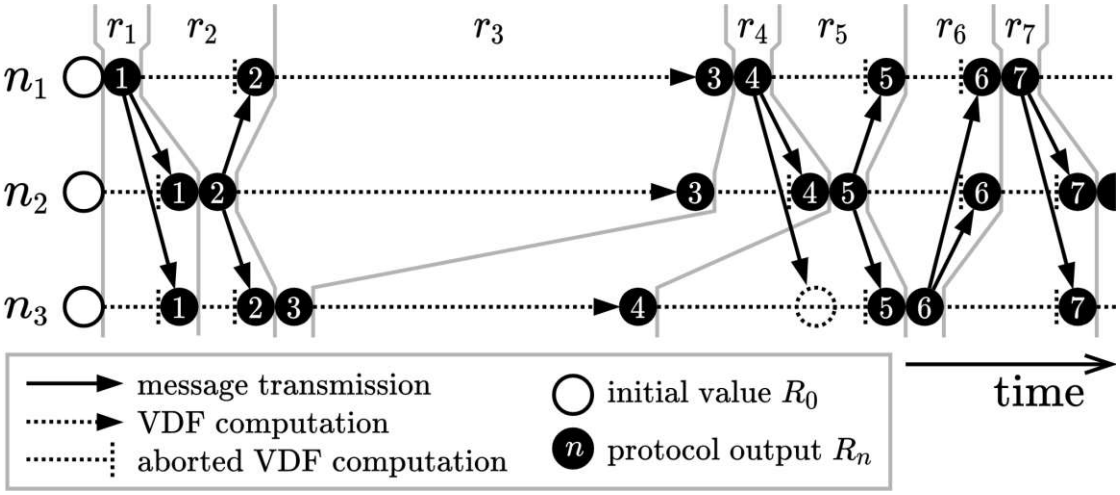

Figure 4.1: Schematic execution of the RandRunner protocol with three nodes n_1, n_2 and n_3, over a period of seven rounds $r_1, ..., r_7$

In any case, the strong uniqueness property of the used VDF ensures that the result obtained via the trapdoor and by evaluation are equal. As the unique output of one VDF serves as the input of the next VDF, the entire sequence of random beacon values generated through these chained VDFs is deterministic and predetermined after the initial protocol setup. By relying solely on the computation of the (chained) unique VDF outputs, either with or without the trapdoor, as random beacon values, agreement on the sequence of these values by all participants is trivially achieved. Therefore, our protocol design avoids the necessity for a Byzantine consensus protocol during execution to agree on random beacon values and the hereby associated requirements and overheads such as high communication complexity. Further, as RandRunner's beacon values are deterministic, the protocol does not suffer from inconsistencies due to network partitions. Hence, an adversary may only be able to influence the unpredictability guarantees of the presented design, for which we show in Section 4.7.4 that it can be sufficiently bounded within our protocol such that the desirable properties expected from a random beacon are nevertheless achieved.

4.5 System and Threat Model of the RandRunner Protocol

The adversary's goals are to violate the security guarantees expected for a random beacon protocol. In particular, the adversary might try to bias the produced randomness, induce a liveness- or consistency failure, or trick a (third) party into accepting an invalid random beacon. Another attack is to learn/predict future random beacon outputs before other nodes obtain those values. We consider the following system model in which we demonstrate the security of our protocol against all of these attacks:

We assume a fixed set of n participants $\mathcal{P} = \{1, 2, \ldots, n\}$ with corresponding public parameters $\mathscr{P} = \{pp_i \mid i \in \mathcal{P} \wedge \mathit{VerifySetup}(\lambda, pp_i) = \mathit{accept}\}$. The validity of these parameters can independently and non-interactively be verified by all parties, and only valid participants with valid parameters form the set $\mathcal{P}$. For our analysis, we consider a static adversarial model where at most f nodes may be corrupted and exhibit Byzantine behavior, i.e., deviate arbitrarily from the protocol. A node is termed *correct* or *honest* if it does not engage in any incorrect behavior over the duration of the protocol execution, otherwise it is considered *Byzantine*. Adaptive adversaries and their impact on security are further discussed in Section 4.7.4.

Messages sent by correct participants are reliably delivered within a bounded network delay of Δ_{NET} seconds. However, within this work we also show that the unique properties of our VDF-based construction provide an upper bound Δ_{VDF} on the time it takes any participant to learn of the next random beacon value independent of the actual network delay, guaranteeing a notion of liveness to the protocol that is not captured by more classical protocol designs. Specifically, we outline that only *unpredictability* is affected by network asynchrony while all other properties are upheld regardless. After a sufficient period of network stability where Δ_{NET} holds, i.e., some global stabilization time (GST) [53], unpredictability is again achieved quickly. Our additional simulation results in Appendix 4.A.2 show that in practice the original unpredictability guarantees are restored within a linear amount of time relative to the duration of network asynchrony.

To start the protocol, we assume an initial unpredictable value R_0 which becomes available or is computed by all parties after the setup is completed. This bootstrapping step is further described in Section 4.6.2. We furthermore inherit the security assumptions for the underlying trapdoor VDF with strong uniqueness, as described in Section 4.3, and model cryptographic hash functions as random oracles. All VDFs are configured such that correct nodes are able to evaluate them within Δ_{VDF} time without knowledge of the trapdoor. We grant the adversaries a *computational advantage* allowing them to perform this computation α times faster, i.e., within Δ_{VDF}/α seconds. The number T of iterations used for evaluating the VDFs is empirically derived as it highly depends on the speed of the actual implementation. It is set such that executing T iterations of the VDF takes approximately Δ_{VDF} seconds on the best hardware available.

In Section 4.7.4, we carefully analyze the interplay between the protocol parameters Δ_{NET}, Δ_{VDF}, α and the assumption regarding the adversarial strength (f vs. n). For example, if the adversary can compute a VDF as quickly as correct nodes, i.e., $\alpha = 1$, and the parameter Δ_{VDF} and Δ_{NET} are chosen such that $\Delta_{VDF} \gg \Delta_{NET}$, the protocol achieves unpredictability (against all attacks) as long as the adversary controls less than half of all nodes, i.e., $f < n/2$. If we consider a (weaker) covert adversary [5], which *secretly* wants to predict future values, instead we show that our protocol can even tolerate a majority of nodes under the adversary's control (Section 4.7.4).

4.6 The RandRunner Protocol

In this section, we provide details on how to setup and execute the RandRunner protocol. Throughout our description, we will reuse the *Setup*, *VerifySetup*, *Eval*, *TrapdoorEval* and *Verify* algorithms introduced in Section 4.3.4.

4.6.1 Setup

Before the random beacon protocol can be started, each participant has to execute the parameter generation, exchange and verification steps:

Parameter Generation: Regarding initialization, each participant i has to generate the public parameters pp_i used with its individual trapdoor VDF with strong uniqueness. Each party i computes the public parameters pp_i and the corresponding secret trapdoor sk_i by executing $Setup(\lambda)$. Note that λ (and λ_{RSA} in the specific case) are globally agreed upon security parameters, i.e., they cannot be selected by the participant individually as the produced parameters would be considered invalid by other participants.

Parameter Exchange: After all parties have completed the initialization, they have to exchange their public parameters pp_i, but keep their individual trapdoor sk_i secret. At the end of this step, each participant should have the *same* set $\mathscr{P}^* = \{pp_1, pp_2, ..., pp_{n^*}\}$ containing the public parameters of all participants. There are several options how to realize this in practice, ranging from the use of a consensus protocol or public blockchain used as a bulletin board, to an offline exchange where all parties come together in person.

Parameter Verification: Finally, each party verifies the set of exchanged parameters. For the particular VDF we use, this is accomplished by running $VerifySetup(\lambda, pp_i)$ for all $pp_i \in \mathscr{P}^*$. Since *VerifySetup* is a deterministic function, all honest participants implicitly agree on the result for each pp_i. All invalid parameters can be removed from the set $\mathscr{P}^*$ to form $\mathscr{P}$, the set of verified public parameters. The remaining parties which provided the valid parameters form the set $\mathcal{P}$ of parties executing the protocol.

4.6.2 Bootstrapping

After all public parameters are set up, exchanged and verified, the protocol is ready to be executed. Starting the protocol requires an initial random beacon value R_0 which becomes available to all parties running the protocol after the setup is completed at approximately the same time. R_0 is used to select the leader for the first protocol round and serves as the input to the first (leader's) VDF being evaluated.

One can of course use an output of another randomness beacon protocol as initial value R_0. Fortunately, there are a range of possible solutions which avoid this circular dependency, because the properties required from R_0 are less strict compared to the properties expected from a random beacon. In particular, we require that R_0 is unpredictable at the time the public parameter are set up and that it is of high min-entropy. This independence of the generated parameters and R_0 then ensures that adversaries cannot

tweak their public parameters in a way which would give them an unfair advantage at protocol start. A rather simple, yet secure method to obtain R_0 is to use the block hash of some future block from an existing blockchain such as Bitcoin or Ethereum.[8] Notice that a miner-introduced bias is not a problem for bootstrapping our protocol because bias-resistance is not required for R_0, yet using an existing blockchain in this way does not provide an efficient randomness beacon with strong guarantees, as among many properties the missing bias-resistance is crucial for the latter purpose.

Algorithm 1: The RandRunner protocol as executed by each node $i \in \mathcal{P}$

Input: sk_i, $\{pp_1, pp_2, ..., pp_n\}$, T, R_0
Output: $R_1, R_2, R_3, ...R_\infty$
begin
  set $r \longleftarrow 1$
  repeat forever
    derive the round's leader l_r
     // details provided in Section 4.6.4
    compute $x_r \longleftarrow H_{in}(R_{r-1})$
     // maps R_{r-1} to in input space of the VDF
    if $i = \ell_r$ **then**
      // in this case, this node (i) is the leader of round r, so the trapdoor sk_i
      // is used to quickly compute the VDF
      compute $(y_r, \pi_r) \longleftarrow VDF.TrapdoorEval(pp_i, x_r, T, sk_i)$
      broadcast (y_r, π_r)
    else
      // otherwise we obtain the VDF output via the network or by evaluation
      // without the trapdoor
      start computing $(y_r, \pi_r) \longleftarrow VDF.Eval(pp_{\ell_r}, x_r, T)$
      while (y_r, π_r) *is not yet computed/received* **do**
        listen for incoming messages (y, π)
        if *message* (y, π) *received* **and**
         $VDF.Verify(pp_{\ell_r}, x_r, T, y, \pi) = accept$ **then**
          set $(y_r, \pi_r) \longleftarrow (y, \pi)$
    compute and output $R_r = H_{out}(y_r)$
     // maps the VDF output y_r to a 256 bit string
    set $r \longleftarrow r + 1$ // move to the next round

[8] In the unlikely case that there is indeed a fork for the exact block used,the randomness beacon can be executed in parallel until the fork is eventually resolvedand the initial value R_0 becomes agreed upon.

4.6.3 Execution

After successful completion of the protocol setup and bootstrapping, the participants are ready to start the protocol execution. The aim of this execution is to provide a continuous sequence of publicly-verifiable, unpredictable and bias-resistant random beacon values $R_1, R_2, ..., R_\infty$. We give the full protocol from the viewpoint of a node $i \in \mathcal{P}$ in Algorithm 1 and describe the details for protocol execution as follows: Our protocol proceeds in consecutive rounds. At the beginning of each round $r \geq 1$, a unique leader ℓ_r is selected. For this purpose we consider two different approaches: round-robin selection (RandRunner-RR) and randomized sampling (RandRunner-RS) of a leader with uniform probability from all nodes $\mathcal{P}$, using the previous protocol output R_{r-1} as seed for the selection. We provide the details for both approaches in Section 4.6.4. Independent of the method chosen, the protocol produces a new random beacon value R_r, i.e., a fresh 256 bit value as output of a cryptographic hash function at the end of each round.

Execution (common case): In each round r, it is the leader's duty to advance the protocol into the next round. It does so by first mapping the previous random beacon value R_{r-1} to the input space of its VDF using a cryptographic hash function $H_{in} : \{0,1\}^{256} \to \mathcal{X}_{\ell_r}$:

$$x_r \longleftarrow H_{in}(R_{r-1})\,. \tag{4.4}$$

Here, the leaders public parameters pp_{ℓ_r} define the input and output space $\mathcal{X}_{\ell_r}$ and $\mathcal{Y}_{\ell_r}$ of ℓ_r's VDF, whereas x_r is used to denote the input to ℓ_r's VDF in round r. Then, the leader computes the output y_r and corresponding proof π_r of its VDF as follows:

$$(y_r, \pi_r) \longleftarrow \mathit{TrapdoorEval}(pp_{\ell_r}, x_r, T, sk_{\ell_r})\,. \tag{4.5}$$

Finally, the values (y_r, π_r) are broadcast to all nodes. As soon as such a message is received, a node checks the correctness of the received values using $\mathit{Verify}(pp_{\ell_r}, x_r, T, y_r, \pi_r)$. If the values are valid, the node can compute the round's random beacon output R_r by applying a cryptographic hash function $H_{out} : \mathcal{Y}_{\ell_r} \to \{0,1\}^{256}$ to the output:

$$R_r \longleftarrow H_{out}(y_r)\,. \tag{4.6}$$

Execution (failure / adversarial case): In case the leader does not fulfill its duties as described, independent of whether it failed or actively tried to attack the protocol, we still want to ensure that each round r is completed and produces the same result. To achieve this, at the beginning of round r each non-leader node immediately starts to compute the round's VDF output $(y_r, \pi_r) \longleftarrow \mathit{Eval}(pp_{\ell_r}, x_r, T)$ in the background. Due to the sequentiality property of the VDF, this computation takes at least T sequential steps. However, after completing those steps (or receiving the valid values from the round's leader) the values y_r and π_r are available and R_r can be derived as before (see Formula 4.6). Here, the strong uniqueness property of the VDF ensures that the resulting values are always equal to the ones computed by the leader.

4.6.4 Leader Selection

In this section, we describe two possible leader selection strategies which can be used in our protocol design, namely *randomized round-robin* and *sampling uniformly at random*. Depending on the used strategy, the achievable unpredictability guarantees differ to some extent. Random sampling bounds the predictability of the sequence of future leaders and ensures a probabilistic guarantee for the unpredictability of the random beacon, whereas the round-robin approach can provide an absolute bound for unpredictability but the entire sequence of leaders is known after R_0 has been published. For a detailed analysis we refer to Section 4.7.4.

Randomized Round-Robin (RandRunner-RR): When employing randomized round-robin as the leader selection method in our protocol, we rely on R_0 to deterministically derive a randomized sequence $\widetilde{\mathcal{P}}$ of the protocols participants $\mathcal{P}$. In other words, R_0 is used as a seed to shuffle (a canonical representation of) the set of participants $\mathcal{P}$ to obtain the list of participants in randomized order. Let $\widetilde{\mathcal{P}}[j]$ denote the j^{th} element of this list using 0-based indexing. Then, the leaders for all rounds $r \geq 1$ are defined as follows:

$$\ell_r \coloneqq \widetilde{\mathcal{P}}[r \bmod n]\,. \tag{4.7}$$

Randomized sampling (RandRunner-RS): In this case, the output from the previous round, i.e., R_{r-1}, is used to sample the leader ℓ_r for round r uniformly at random from the set of all parties $\mathcal{P}$. Interpreting the 256-bit beacon outputs as numbers, a simple approach which guarantees that each participant i, denoted by its index from 1 to n, is selected with probability (very close to) $1/n$, is to define ℓ_r as:

$$\ell_r \coloneqq (R_{r-1} \bmod n) + 1\,. \tag{4.8}$$

4.6.5 Dissemination

As described in Section 4.6.3 and given in Algorithm 1 (line 8), the leader of each round r is responsible for broadcasting the VDF's unique output y_r and the corresponding proof π_r. If all nodes follow the described protocol and the network is reliable, then this broadcasting step is as simple as the leader sending the values (y_r, π_r) to the other $n-1$ participants directly. This would result in a communication complexity of $\mathcal{O}(n)$. However, an adversarial leader might *selectively* send out this information to a subset of all nodes. While any node can always derive (y_r, π_r) by computing the round's VDF eventually, a slowdown for the subset of nodes which did not receive the message from the adversarial leader is introduced. A potential consequence is a violation of RandRunner's unpredictability guarantees (see Section 4.7.4): Some correct nodes, in inadvertent collaboration with the adversary, may progress faster than the other correct nodes. The root cause for this phenomenon is a combination of two events: (i) an adversary only selectively sent information to some correct nodes and (ii) some correct nodes are not yet able to verify the information received from other correct nodes, as they are missing values from prior rounds (not sent to them by the adversary). Since there is no way to

influence the adversary's actions, we focus on (ii) for our countermeasures. In particular, we set out to ensure that after a correct leader broadcasts (y_r, π_r) all (correct) nodes already have, or timely receive, the information required from prior rounds to verify these values. Two possible strategies to accomplish this are given in the following:

Reliable Broadcast

A straightforward solution is to employ a reliable broadcast where every (correct) node forwards any valid message (y_r, π_r) it received to all other nodes once. This results in a communication complexity of $\mathcal{O}(n^2)$ as each of n nodes sends $\mathcal{O}(n)$ bits per round, minimizes latency (Δ_{NET} is small) as message are not relayed over multiple hops, and is practical as long as the number of nodes n is reasonably small.

Gossip Protocol

If n is large, one can use gossip/rumor spreading protocols instead. Here, one node, in our case the leader of the current round, initiates the spreading of the information (y_r, π_r) by sending it to a random subset of nodes. All nodes which have received a valid message continue to forward the message to another subset of nodes until all nodes are eventually informed with high probability.[9] As messages are forwarded over multiple hops, typically logarithmically many, latency increases compared to the prior approach (Δ_{NET} is higher). However, the communication complexity is significantly reduced to (at least) $\mathcal{O}(n \log n)$ in total or $\mathcal{O}(\log n)$ per node respectively. We refer to the works of Demers et al. [47], Karp et al. [75], and the large body of subsequent work for further details on gossip protocols.

These approaches are provided exemplary as an optimization of the dissemination layer is not the main focus of this work. Our security proofs presented in Section 4.7 are agnostic to the selected information dissemination approach. Any optimization, which can reliably disseminate our small and inherently verifiable message (y_r, π_r) in every round, is suitable. The choice of the approach largely depends on the intended application scenario. As a general guideline, we consider that reliable broadcast is best suited if the number of participants n is small, as it minimizes latency and is straightforward to implement. The larger the number of participants n, the more appropriate gossip-based approaches become. Additionally, we note that one may actually use all available network bandwidth in favor of a lower latency instead of minimizing the communication costs to achieve best possible performance in practice. Either way, an expected higher latency Δ_{NET} can be compensated by increasing the Δ_{VDF} parameter, which defines the number of iterations T for the used VDFs.

[9] For the case of RandRunner, the unlikely delivery failures a probabilistic gossip protocol may produce are not a problem, as the transmitted values are eventually obtained via evaluation of the VDFs after at most Δ_{VDF} time.

4.7 Analysis of RandRunner's Security Guarantees

4.7.1 Liveness

Intuitively, a distributed protocol achieves the liveness property if an adversary cannot prevent the protocol from making progress. A stronger form of liveness, specifically in the context of random beacon protocols, is the property of guaranteed output delivery [38, 106]. A protocol which achieves this property additionally ensures that the adversary can not even prevent the protocol from producing a fresh output in each round. As this stronger form of liveness is also closely related to the bias-resistance property (see Section 4.7.2), it is crucial for a randomness beacon protocol such as RandRunner which targets the continuous provision of random numbers. As we outline in the following, our protocol achieves liveness and its stronger form of guaranteed output delivery, independent of the adversary's actions and network conditions.

Theorem 7. *(Liveness & Guaranteed Output Delivery) Each correct node which has completed some round $r \geq 0$, completes round $r+1$ and outputs a new random beacon R_{r+1} within at most Δ_{VDF} seconds.*

Proof. Round $r = 0$ is completed by all nodes as soon as the protocol setup is finished and the initial random beacon R_0 becomes available. For all other rounds $r \geq 1$, each node can non-interactively derive the unique round leader ℓ_r using the specified leader selection algorithm and use the hash function H_{in} to derive the input x_r for ℓ_r's VDF. With the *Eval* function, each node can further compute the result (y_r, π_r) of the VDF within Δ_{VDF} seconds. Finally, H_{out} is used to map y_r to R_r. Since both the time required to compute the leader selection algorithm and the hash functions are negligible, each node can output R_r within Δ_{VDF} seconds after it completed the previous round. □

4.7.2 Bias-Resistance

Bias-resistance ensures that an adversary cannot manipulate the produced random beacon values to its advantage. Ideally, a protocol fully prevents that an adversary can influence the distribution of the produced outputs. As adversaries can even benefit from just withholding produced results after they become available to them, the strongest form of bias-resistance can only be achieved by protocols which also guarantee that an output is produced in every round.

Theorem 8. *(Bias-Resistance) For any round $r \geq 1$, the output R_r can not be influenced in any way after the protocol setup is completed.*

Proof. As discussed in the section on liveness, the result of round R_r is derived from R_{r-1} by mapping R_{r-1} to a value x_r from the input space of the leader's VDF, computing the leader's VDF to obtain (y_r, π_r), and finally mapping y_r to R_r. The mapping steps just use (deterministic) hash functions and are thus not prone to any manipulation by the adversary. The VDF is computed using either the *Eval* or *TrapdoorEval* algorithm. Due

to the strong uniqueness property of the VDF the obtained result y_r is equal, no matter which of the two algorithms is used. Also, in case an adversarial leader sends out some invalid message (y'_r, π'_r), all correct nodes check the values using the *Verify* algorithm and only accept a single unique output per input. Consequently, also the VDF step is deterministic and fully verifiable, and the full derivation step from R_{r-1} to R_r cannot be influenced by the adversary in any way. As the setup of the protocol is executed and verified before the first input R_0 becomes available, and each step is shown to be deterministic, bias-resistance is ensured during the entire execution of the protocol. □

4.7.3 Public-Verifiability

In order to verify the correctness of a random beacon output R_r, a (third-party) verifier needs a transcript of the protocol's execution. A valid transcript can be provided by any correct party and consists of

1. the public parameters $\mathscr{P}$ of all protocol participants,
2. the initial random beacon value R_0, and
3. the round's VDF output (y_s, π_s) for all $s \in \{1, 2, ..., r\}$.

The setup of the protocol can be publicly verified, as specified in Section 4.6.1. The same is true for each step in the protocol execution: As seen in the proofs for liveness and bias-resistance, the random beacon output R_r of every round $r \geq 1$ is derived cryptographically from the previous output R_{r-1}. The only primitives used are cryptographic hash functions for mapping in- and outputs, and trapdoor VDFs with strong uniqueness. In order (for a third party) to verify the correctness of a protocol output R_r, given R_{r-1}, the involved hash functions are recomputed and the correctness of the VDF computation is checked by using the *Verify* algorithm. Essentially, a third party just follows the protocol as described for a participant i in Algorithm 1, leaving out the evaluation and communication steps.

Regarding computation complexity, the verification of each round r requires the execution of two hash functions and one *Verify* algorithm. The costs for the hash functions are negligible, and also the *Verify* algorithm is efficient as it requires only around three exponentiations for typical parameters of the VDF [94]. Furthermore, the verification complexity does not depend on the number of parties executing the protocol.

4.7.4 Unpredictability

Unpredictability describes a security guarantee which ensures that the adversary's ability to predict future protocol outputs is bounded. Depending on the particular protocol, this bound can be absolute or probabilistic. An absolute bound ensures that, for some fixed $d \geq 1$, the adversary cannot obtain the protocol output of round $r + d$, when correct nodes only know the outputs up to round r. A probabilistic bound guarantees that the likelihood that the adversary can successfully predict d future protocol outputs drops

exponentially as d increases linearly. For our protocol the achieved bound depends on the chosen leader selection method. In the following, we prove that the round-robin variant (RandRunner-RR) ensures an absolute unpredictability bound of $d = f \cdot \alpha$ (see Theorem 9), whereas our stochastic simulations show that random sampling of leaders (RandRunner-RS) guarantees that predicting future values becomes exponentially less likely when d increases.

The Adversary's Strategy

In a leader-based protocol like RandRunner, the adversary can always predict future random beacon outputs to some extent. This is possible because in every round the corresponding leader knows the output before sending it to the other parties. In our case, an adversarial leader can compute the round's output by evaluating its VDF using the trapdoor. Clearly, this is faster compared to correct nodes, which only obtain such outputs after the adversary chooses to broadcast them, or if they compute the VDF without the trapdoor, which takes Δ_{VDF} seconds. In order to extend this advantage to multiple rounds, the adversary must withhold the output of the VDF on purpose. In case the adversary is lucky, and continues this strategy of withholding its outputs, the adversary increases its advantage (i.e., the number of outputs it knows before the correct nodes do) as long as a continuous sequence of adversarial nodes are selected as leaders. However, due to the randomized leader selection, long sequences of this kind quickly become unlikely. As soon as an honest node is selected as leader, the adversary's advantage decreases as the adversary is not in possession of the trapdoor for an honest node's VDF and consequently has to spend Δ_{VDF}/α time to predict one additional step. We recall that $\alpha \geq 1$ denotes the adversary's VDF computation speed relative to correct nodes. An α value of 1.5, for example, means that we assume that the adversary can compute VDFs up to 50% faster. In the meantime, the honest nodes work on reducing the adversary advantage. For each round in which the adversary was selected as leader, honest nodes have to spend Δ_{VDF} time to catch up one step. As soon as all adversarial leaders' outputs have been computed (and a correct leader is selected again) it takes them only Δ_{NET} seconds to compute and distribute a new random beacon output, thus quickly diminishing the adversary's advantage.

A first Glance at RandRunner's Unpredictability Bounds

Rounds with an adversarial leader benefit the adversary in terms of its ability to predict future protocol outputs, whereas rounds with a correct leader benefit the honest nodes. This rather natural phenomenon can be observed in our stochastic simulations and constitutes the basis for the security proof of Theorem 9. However, as it is so fundamental, we want to provide further insights into why this is indeed the case: In each round r, we either have an adversarial or correct leader. In case the leader is adversarial, the adversary can immediately predict the outcome R_r of round r using the leader's trapdoor for the evaluation of the VDF. The correct node may be delayed by up to Δ_{VDF} seconds before they learn R_r if the adversary does not broadcast the round's VDF output and

proof as specified by the protocol. Clearly, following the strategy of withholding this information the adversary gains a (temporary) advantage in its ability to predict future protocol outputs. In the other case, i.e., in rounds with an honest leader, all honest nodes advance by one round within Δ_{NET} time, whereas the adversary can only advance to the next round after it received the round's output from the leader or obtained the result by computing the leader's VDF without the trapdoor. If the adversary cannot finish this computation before the message from the leader is received, all honest nodes catch up and all the adversary's advantage in diminished. Otherwise, the adversary loses some of its advantage as it takes the adversary Δ_{VDF}/α time to proceed to the next round, whereas the honest nodes require at most $\Delta_{NET} < \Delta_{VDF}/\alpha$ seconds.

With this intuition at hand, we now have an informal look on the unpredictability guarantees RandRunner-RR provides in a simplified scenario in which not only the adversary, but also the honest nodes can communicate without a network delay ($\Delta_{NET} = 0$). In this setting, the protocol achieves absolute unpredictability for $d = f \cdot \alpha$ as long as the following inequality is fulfilled:

$$n > f + f \cdot \alpha \,. \tag{4.9}$$

For the case that the adversary and the honest nodes can compute VDFs at the same speed, i.e., $\alpha = 1$, this is reduced to a standard majority assumption $n > 2f$. In case the adversary can compute VDFs faster ($\alpha > 1$), the fraction of honest nodes compared to adversarial mode must increase accordingly. In cases where $\Delta_{NET} > 0$ and $\Delta_{VDF} \gg \Delta_{NET}$, the simplified bound provided by the above inequality for the $\Delta_{NET} = 0$ case closely resembles the general bound we prove in Theorem 9. This more precise bound carefully considers the interplay between the network delay Δ_{NET} and the VDF computation time Δ_{VDF}.

Unpredictability for RandRunner-RR

If we use (randomized) round-robin as the leader selection method, our protocol achieves an absolute unpredictability bound of $d = f \cdot \alpha$ rounds for all configurations which satisfy the following inequality:

$$f \cdot \alpha \leq (n - f) \cdot \left(1 - \frac{\Delta_{NET} \cdot \alpha}{\Delta_{VDF}}\right) \tag{4.10}$$

or, equivalently:

$$n \geq f + \frac{f \cdot \alpha}{1 - \frac{\Delta_{NET} \cdot \alpha}{\Delta_{VDF}}} \,. \tag{4.11}$$

To simplify the formulation of the following statements showing this claim, we formally define two intuitive terms: the k^{th} *period of rounds* and the adversary's *advantage*:

Definition 4. *For every natural number k, the k^{th}* period of rounds *of the protocol is defined by the n consecutive rounds $(k-1)n+1, (k-1)n+2, ..., kn$.*

For example if $n = 5$, rounds 1 to 5 form the 1^{st} period of rounds ($k = 1$), rounds 6 to 10 the 2^{nd} period ($k = 2$) and so on.

Definition 5. *The adversary has* advantage $v \geq 0$ with respect to round r *if and only if the following two conditions hold:*

1. *Some correct node knows the protocol output of round r, but no correct node knows the output of round $r + 1$.*
2. *The adversary knows the protocol output of round $r + v$, but not of round $r + v + 1$.*

In our proof of Theorem 9, we will *show by induction on k* that there is no k^{th} period of rounds of the protocol in which the advantage of the adversary with respect to any round exceeds $f \cdot \alpha$. We start by showing the following Lemma 9, which will help us to establish the induction base.

Lemma 9. *For all protocol configurations which fulfill Inequality 4.10, the following holds: If the adversary has advantage 0 with respect to some round r, its advantage with respect to the rounds $r + 1, r + 2, \ldots, r + n$ is at most $f \cdot \alpha$.*

Proof. We start by first considering rounds with a correct leader. In this case, the time required for the adversary to predict a protocol output is bounded by the VDF computation time of Δ_{VDF}/α, whereas the correct nodes advance to the next round within Δ_{NET} seconds. Since $\Delta_{NET} \leq \Delta_{VDF}/\alpha$ holds for all protocol configurations fulfilling Inequality 4.10, the number of rounds the adversary can predict never increases during periods with honest leaders. Consequently, to obtain an upper bound for the number of predictable rounds, we only have to consider rounds with adversarial leaders. Within a period of n consecutive rounds, there are exactly f such rounds, which are consecutive in the worst case. Let us consider this worst case for our upper bound: At the beginning of these f rounds, the adversary immediately obtains the results of all of those rounds, as it can use the adversarial leaders' VDF trapdoors to compute the results. The honest nodes, on the other hand, have to compute f VDFs without the trapdoor (assuming that the adversary withholds the results), requiring $f \cdot \Delta_{VDF}$ seconds to complete. In the same time, the adversary may already try to compute the VDFs of the honest leaders in the next few rounds. As it takes the adversary Δ_{VDF}/α time to compute one such VDF (without the trapdoor), it can compute at most

$$\frac{f \cdot \Delta_{VDF}}{\Delta_{VDF}/\alpha} = f \cdot \alpha \tag{4.12}$$

outputs during this time period. Consequently, as soon as the honest nodes finish the computations for the f rounds of adversarial leaders and hence catch up by f rounds, the adversary's ability to predict future protocol outputs increases from f to $f - f + f \cdot \alpha = f \cdot \alpha$ rounds. From that point on, there are only rounds with correct leaders remaining, and as the number of rounds the adversary can predict cannot increase in rounds with correct leaders, the correctness of the lemma follows. □

Next, we prove a claim that will be important for the induction step of the proof of Theorem 9.

Lemma 10. *For all protocol configurations which fulfill Inequality 4.10, the following holds: If the adversary has advantage $v \leq f \cdot \alpha$ with respect to some round r, its advantage with respect to round $r + n$ is at most $v' \leq f \cdot \alpha$.*

Proof. In the worst case, all correct nodes can complete n consecutive rounds within

$$\Delta_w = f \cdot \Delta_{VDF} + (n - f) \cdot \Delta_{NET} \tag{4.13}$$

seconds, because there are f rounds in which the adversarial leader may not broadcast the result, requiring Δ_{VDF} time each, and $(n - f)$ rounds with correct leaders which make progress immediately and broadcast the results within Δ_{NET} seconds. If during this time period the adversary obtains a round's output by relying on a correct node's message, instead of obtaining it via computation by itself, the adversary could not predict this value – its advantage with respect to this round is zero. Consequently, this lemma immediately holds by Lemma 9. If, on the other hand, the adversary does not rely on messages from correct nodes for its progress, it can compute the outputs of at most

$$w = f + \frac{\Delta_w}{\Delta_{VDF}/\alpha} = f + \frac{f \cdot \Delta_{VDF} + (n - f) \cdot \Delta_{NET}}{\Delta_{VDF}/\alpha} \tag{4.14}$$

rounds during the period of Δ_w seconds, because there are f steps in which the adversary immediately obtains the result as an adversarial node is leader, whereas all other steps rely on computing a VDF without a trapdoor, taking Δ_{VDF}/α time each. In other words, the adversary advances by w rounds, while, during the same period of time, the correct nodes advance by n rounds. As $w \leq n$ follows directly from rearranging Inequality 4.10, the adversary cannot increase its advantage ($v' \leq v$) and the lemma holds. □

Theorem 9. *(Unpredictability): All protocol configurations satisfying Inequality 4.10 guarantee absolute unpredictability for $d = f \cdot \alpha$.*

Proof. The statement is equivalent to the claim that the adversary's advantage with respect to any round never exceeds $f \cdot \alpha$. We give a proof by induction on the k^{th} periods of rounds and start with the induction base $k = 1$. Since the adversary cannot predict the initial random beacon value R_0, Lemma 9 implies that the adversary's advantage with respect to the rounds $1, 2, ..., n$ is bounded by $f \cdot \alpha$. This already proves that the statement is true for the first period.

For the induction step, we have to show that if the adversary's advantage with respect to the rounds in the k^{th} period is bounded by $f \cdot \alpha$, the same is true for the rounds in the $(k + 1)^{th}$ period. Consider the rounds $(k - 1)n + 1, (k - 1)n + 2, \ldots, kn$ of the k^{th} period. We apply the induction assumption together with Lemma 10 and obtain that the advantage of the adversary in the rounds $kn + 1, kn + 2, ..., (k + 1)n$ is at most $f \cdot \alpha$, which we wanted to prove.

We have shown that there is no k^{th} period of the protocol containing a round in which the adversary's advantage exceeds $f \cdot \alpha$. This covers all rounds and hence concludes the proof. □

In order to simplify the exposition of the proof, Theorem 9 and definition 5 consider the adversary's ability to predict future protocol outputs relative to *some* honest node. However, if one wants to consider the unpredictability guarantee relative to all / the slowest honest node, a very similar result applies as all correct nodes synchronize within Δ_{NET} time in the broadcasting step of the protocol. Therefore, the same security bound of $f \cdot \alpha$ rounds holds for all nodes if we add an additive term of Δ_{NET}, i.e., the adversary does not have advantage $f \cdot \alpha + 1$ for longer than Δ_{NET} time. The additional term of Δ_{NET} time is required as we assume that the adversary can send and receive messages (also from the correct nodes) without any network delay, whereas messages between correct nodes experience a network delay of up to Δ_{NET} seconds.

Unpredictability for RandRunner-RS

As described in Section 4.6.4, the leaders in RandRunner can also be selected uniformly at random (RandRunner-RS) as an alternative to the round-robin style leader selection (RandRunner-RR) analyzed in the previous section. Due to the probabilistic nature of selecting leaders, RandRunner-RS provides a probabilistic guarantee of unpredictability, whereas RandRunner-RR guarantees an absolute bound for the adversary's ability to predict future outputs. The reason for this difference is that the round-robin leader selection ensures that there cannot be more than f adversarial nodes within any period of n rounds at any point in the protocol execution. When leaders are picked at random, however, there can be up to $u \leq v$ adversarial leaders in any period of v rounds (for an arbitrary number of rounds v), although the likelihood of having a high fraction of adversarial leaders during such a period decreases exponentially for longer periods.

Similar to the round-robin case, the probabilistic guarantee for unpredictability can be provided as long as the honest nodes make progress faster than the adversarial nodes. Let $p_A \coloneqq f/n$ and $p_H := 1 - p_A$ denote the probability of selecting an adversarial or honest leader respectively. Then the rates of progress, i.e., the average time required per protocol round, λ_H for the correct nodes and λ_A for the adversary, are given as follows:

$$\lambda_H \coloneqq \frac{1}{\Delta_{NET} \cdot p_H + \Delta_{VDF} \cdot p_A} \tag{4.15}$$

$$\lambda_A \coloneqq \frac{1}{\Delta_{VDF}/\alpha \cdot p_H}\,. \tag{4.16}$$

Intuitively, RandRunner-RS works as long as correct nodes progress faster than adversarial nodes, i.e., if $\lambda_H > \lambda_A$, because any advantage an adversary has in some round will disappear after a sufficient number of rounds. The more both rates differ, the quicker any advantage disappears and the more unlikely a big advantage becomes. Obtaining a closed form expression for the corresponding probability appears difficult, as the advantage of

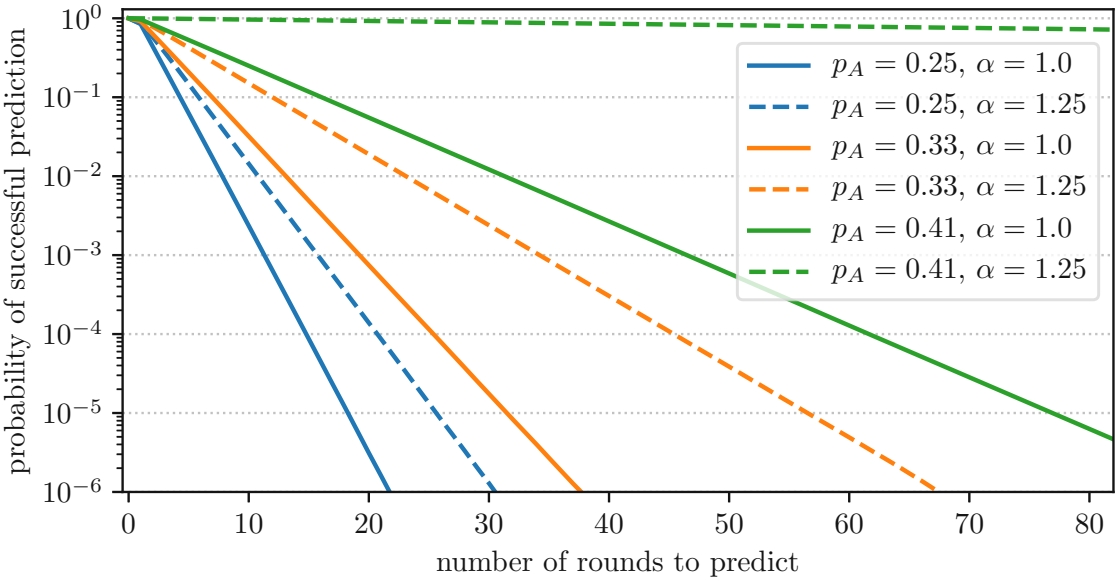

Figure 4.2: Simulation of RandRunner-RS' unpredictability guarantees, showing the likelihood of adversaries with different strengths (p_A, α) being able to predict future protocol outputs at any particular point of the protocol's execution, simulated over a duration of 10^{10} rounds. $(\Delta_{NET}/\Delta_{VDF} = 1/10)$

the adversary in a particular round depends on the previous protocol state as well as on the sequence of future leaders. However, by simulating protocol executions we can derive these probabilities empirically. This is illustrated in Figure 4.2 which presents our simulation results, considering different assumptions in regard to the fraction of adversarial nodes (p_A) and the adversary's advantage in terms of sequential computation speed for the VDF, denoted by α. For the parameters Δ_{NET} and Δ_{VDF}, we select a fixed ratio of $\Delta_{NET}/\Delta_{VDF} = 1/10$, as we observe that the simulation is typically more sensitive to a change of p_A and α. For each of the exemplary parameters picked, we simulate the protocol execution for 10^{10} rounds[10]. At any point in time where a state change happens, i.e., when a new value is received or computed, we measure any potential advantage the adversary has in comparison to the other nodes, and use this measurement to derive the probabilities that it can predict a certain number of rounds at any particular point in time. An extended evaluation is provided in Appendix 4.A. The source code used to obtain the simulation results is publicly available on Github [104].

Unpredictability against a Covert Adversary

In the previous sections, we analyzed the unpredictability guarantees RandRunner provides in regards to an adversary which actively attacks the protocol during execution. However, in practice an adversary can hardly profit from any attack on unpredictability if

[10]For $\Delta_{NET} = 5$ seconds, this corresponds to more than 1500 years of protocol execution in real time.

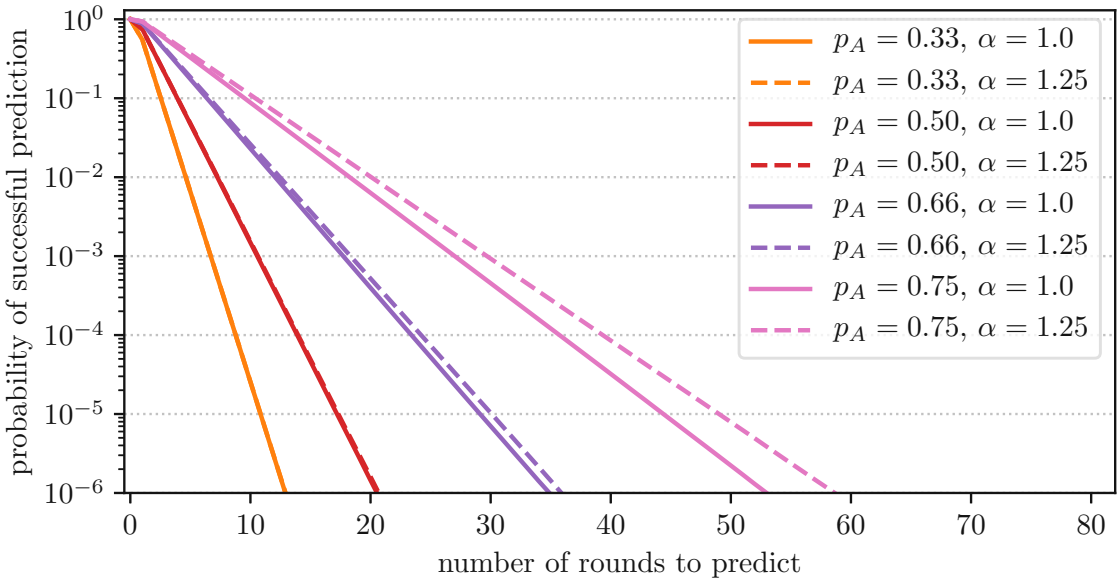

Figure 4.3: Simulation of RandRunner-RS' unpredictability guarantees against covert adversaries. As in Figure 4.2, the simulation shows the likelihood of successful prediction by adversaries of different strengths (p_A, α) considering a simulation duration of 10^{10} rounds. $(\Delta_{NET}/\Delta_{VDF} = 1/10)$

the correct network participants are aware of the fact that the protocol is being attacked. The base for the detection of ongoing attacks is that correct nodes expect new protocol outputs in intervals of at most Δ_{NET} seconds. There are only two reasons for protocol outputs being delayed any further:

1. an adversarial leader withholds the next protocol output or
2. the network behaves asynchronously.

As the second case is unlikely if Δ_{NET} is properly configured, any delay is a strong indicator for an attack. This leads us to the notion of a *covert* adversary [5] which aims to hide all traces that can be used for detecting the attack.

RandRunner is resilient against covert adversaries, because a covert adversary has to broadcast new protocol outputs after at most Δ_{NET} seconds to make sure the attack stays invisible. Also, the computation time available to compute honest leaders' VDFs is reduced to Δ_{NET}. Therefore, the bound of $\lambda_H > \lambda_A$ for achieving unpredictability in the general case is reduced to the following inequality considering the covert case:

$$\Delta_{NET} < \Delta_{VDF}/\alpha \cdot p_H \,. \tag{4.17}$$

The bigger the (relative) difference between both sides, the more the fraction of adversarial nodes p_A and their computational advantage can be increased. A particular distinguishing advantage compared to other protocol designs is that RandRunner even works against

an attacker which controls a majority of nodes in the covert adversary model. This is illustrated in Figure 4.3, where we, among others, consider an adversary which controls 75% of all nodes in the system. As for the non-covert case, we provide additional simulation results for a range of different parameters in Appendix 4.A.

Unpredictability against an Adaptive Adversary

In the static adversary model, it is assumed that the adversary may control up to a threshold of f nodes. Those nodes may behave Byzantine, however the set of nodes controlled by the adversary is fixed and defined prior to the start of the protocol. Another commonly encountered adversarial model is concerned with *adaptive* corruptions [33, 89]. Adaptive adversaries may decide which nodes to corrupt (take control of) based on information collected during the execution of the protocol. As in the static case, the adversary's capabilities are bounded by the threshold f. A further distinction can be made between *fully adaptive* and *mildly adaptive* adversaries [89], where the former implies practically instantaneous corruptions, whereas the latter incurs some non-zero delay before a corruption takes effect. In practice, a fully adaptive adversary is likely an unrealistically strong assumption, in particular if we consider communication delay to be non-zero. Nevertheless, we discuss the resilience of the RandRunner protocols against both variants of adaptive adversaries.

We recall that the security proof provided for RandRunner-RR explicitly covers the worst case of f consecutive adversarial nodes. Consequently, the absolute unpredictability bound (see Theorem 9) remains unaffected even if an adaptive adversary of either flavor is assumed. As we elaborate in Appendix 4.A.3, RandRunner-RR additionally provides probabilistic unpredictability guarantees before this unpredictability bound is reached, very similar to the probabilistic guarantee shown for RandRunner-RS. However, because in essence this is achieved by randomizing the round-robin sequence of leaders after (static) corruptions have taken place, these additional probabilistic guarantees in RandRunner-RR do not hold for mildly or fully adaptive corruptions.

RandRunner-RS provides probabilistic guarantees regarding unpredictability. Intuitively, the further into the future an adversary wishes to predict beacon values, the less likely they are to succeed. Considering an adaptive adversary, RandRunner-RS's probabilistic guarantees degrade gracefully, depending on the number of corruptions and the time required to corrupt the nodes. In the worst case, considering a fully adaptive adversary which may instantaneously corrupt up to f nodes, our simulations show that the adversary's prediction capabilities are shifted by f rounds. In other words, in this case the adversary can pick a single point in time at which it is able to once instantaneously corrupt at most the sequence of the next f leaders, and thereby predict the outcome of the next f protocol rounds. However, after that point the probability of predicting any further rounds again start to drop exponentially. This is to be expected behavior and can be seen as granting the adversary a one-time lead of f rounds. Fortunately, in practice the worst case of a fully adaptive adversary is highly unrealistic. In a more realistic case, corruptions would require a considerable amount of time, i.e., much longer than Δ_{NET}

and Δ_{VDF}. In this case however, by the time the adversary is able to successfully corrupt the next leader it has already fulfilled its duty of broadcasting the next beacon value, rendering the attack ineffective for gaining an additional advantage over static corruptions. Thus, such mildly adaptive corruptions do not affect the guarantees provided.

The analysis of adaptive adversaries for both RandRunner-RR and RandRunner-RS serves to further highlight their different properties and potential use cases. In particular, if resilience against fully adaptive adversaries is deemed a necessity, utilizing RandRunner-RR and waiting for the absolute unpredictability bound presents a solution. We point out that for this scenario a smaller set of participants is advantageous in regard to the required waiting periods before unpredictability can be guaranteed. On the other hand, RandRunner-RS can offer unpredictability with probabilistic guarantees that incurs shorter waiting periods, if mildly adaptive adversaries are assumed.

4.8 Comparing RandRunner to Existing Distributed Randomness Beacons

In recent years, a wide range of possible approaches to obtain publicly-verifiable randomness have been presented. This includes solutions which extract randomness from existing systems. In this regard, Clark and Hengartner [44] show how to collect (small amounts) of entropy from closing prices of stocks. As noted by Pierrot and Wesolowski [93] this approach relies on the assumption that the published financial data cannot be manipulated. Similarly, the works of Bonneau et al. [25] and Bentov et al. [12] demonstrate how to extract near-uniformly distributed bits from one or a sequence of Bitcoin blocks. However, as stated by the authors and analyzed in later work [93], these approaches cannot provide truly unbiased randomness.

The line of research on blockchain protocol designs, in particular Algorand [42] and Ouroboros Praos [46], can also be used to obtain distributed randomness. Both protocols internally use verifiable random functions [85] (VRFs) to produce a sequence of random numbers. In this way, both designs can output randomness as a byproduct of their operation without any significant additional communication cost. Using hashchains instead of VRFs, Azouvi et al. [6] present a solution with similar characteristics as a Smart Contract for the Ethereum blockchain. However, all of these approaches, where the adversary might be responsible for computing and then revealing the next random output, are not strictly bias-resistant, as the adversary can always decide to withhold the next random output after gaining knowledge of it [106]. Strong bias-resistance, as also provided by RandRunner, ensures that there is a guaranteed protocol output in every round, regardless of the actions taken by the round's leader.

Protocols which can provide strong bias-resistance have also been constructed by using threshold cryptography, in particular using publicly-verifiable secret sharing ([78, 38, 114, 106]) or unique threshold signatures ([33, 72]). The proposal of running and combining the results of n secret sharing instances, as seen in the Ouroboros [78] and Scrape [38]

protocols, has since been improved by Syta et al. [114] (RandHerd) and by our HydRand protocol (see Chapter 2 of this dissertation or our corresponding publication [106]). HydRand achieves a communication complexity of $\mathcal{O}(n^2)$ in a synchronous system model with $n = 3f + 1$ participants, without requiring a distributed key generation (DKG) protocol or relying on pairing-based cryptography. As it is the case with RandRunner, unpredictability for HydRand is achieved after a few rounds, whereas the approaches of Cachin et al. [33] and Dfinity [72] ensure unpredictability after a single round. The two latter approaches also achieve a communication complexity of $\mathcal{O}(n^2)$. They, however, rely on a trusted dealer or DKG protocol and, e.g., BLS [24, 23], as a unique pairing-based threshold signature scheme. In comparison, RandRunner is built using an RSA-based VDF and does not require a trusted dealer or DKG protocol for its setup. Its communication complexity improves upon all the threshold cryptographic approaches, as a single leader drives the protocol forward, whereas the interaction between all, or at least a large subset of the participants, is required for the other protocols. Regarding the guaranteed output delivery property, HydRand can output fresh randomness at regular intervals as it operates in a fully synchronous system model, whereas RandRunner and other protocols which are safe under asynchrony can only guarantee that an output is produced every round. For RandRunner, the round duration may vary depending on network conditions or if the protocol is attacked, but is upper bounded by the Δ_{VDF} parameter. The delay RandRunner introduces in these circumstances can be seen as an advantage, as any delay serves as a strong indicator for an active attack (assuming network outages are rare) and thus strengthens the confidence in the protocol if it progresses as fast as expected. Similar to Cachin et al. [33], our protocol ensures consistency even under asynchronous network conditions and proceeds at the network speed when not attacked, whereas HydRand loses consistency if the synchrony assumption is violated and cannot progress faster than the initially specified network delay, i.e., does not offer optimistic responsiveness [90]. Dfinity's security proofs also rely on synchrony.

A different line of research focuses on the instantiation of a randomness beacon based on delay functions (also known as slow-time functions), which can be seen as predecessor to VDFs (as used in RandRunner) without an efficient verification procedure. Using this primitive, Lenstra and Wesolowski [80] designed the Unicorn protocol, in which in a first phase a set of distrusting parties collect a pool of inputs. In a second step those inputs are hashed and fed into a delay function, the output of which forms the randomness. As the delay parameter is picked such that no party can compute the output during the time when changes to the inputs are allowed, the result is bias-resistant and unpredictable as long as at least one party provides a random input with sufficient entropy. A similar approach is later implemented by leveraging a Smart Contract on the Ethereum platform for agreement on the inputs [32]. To circumvent the limitations of the platform, the authors of this approach describe an interactive, incentive-based game for verification. We believe that these systems and the underlying idea of first agreeing on a set of inputs and then executing a long-running (verifiable) delay function on these inputs are well suited for scenarios in which unpredictable and bias-resistant randomness is required infrequently. In comparison, RandRunner does not require an

agreement protocol for the VDF inputs and can provide a sequence of random numbers in short intervals and with much lower communication overhead. Moreover, RandRunner can also ensure unpredictability in scenarios where the adversary can compute the VDF faster than honest nodes.

4.9 Summary of our Findings on the RandRunner Protocol

By extending the VDF introduced by Pietrzak [94] to a trapdoor VDF with strong uniqueness, which may be of independent interest, we lay the foundation for our novel randomness beacon protocol RandRunner. Our design and the properties we achieve are unique in many ways. First, RandRunner is extremely simple: It is built on top of cryptographic hash functions, and the introduced VDF is based on the well studied RSA assumption. The setup of the protocol does not require a DKG protocol and can be verified non-interactively. Instead of relying on a Byzantine or blockchain-based agreement protocol to ensure consistency across all nodes, consistency is achieved by leveraging the strong uniqueness property of the underlying VDF. Thereby, the protocol essentially provides a predetermined, yet unpredictable sequence of random numbers. This novel design has tremendous advantages in terms of efficiency and scalability, as the removal of the agreement protocol reduces communication costs significantly. In our case, only a single message of approximately 10 KB in size has to be propagated through the network to produce a fresh random beacon output.

Additionally, our design is very resilient to temporary network delays or network outages. Although being designed for practical deployment scenarios with bounded network delay, RandRunner retains consistency and liveness even if the network connectivity between correct nodes breaks down completely. We have proven that RandRunner achieves unpredictability under a synchronous network model, and provided stochastic simulations to analyze the protocol in case of temporary network failures. Under these circumstances, we observed that the provided unpredictability guarantees degrade gradually, even when we consider an adversary which is not affected by the network delays. Furthermore, our results also show that the protocol can recover quickly, i.e., in a linear amount of time respective to the duration of the network outage.

Whenever the network is in good condition, and the protocol is not under attack, the protocol is *responsive* [90, 121] and proceeds at the speed of the network, i.e., it is not slowed down by introducing artificial delays. Attacks introduce a (parameterizable) slowdown of the protocol, serving as a strong indication for an ongoing attack. This leads us to the additional evaluation of RandRunner in a covert adversary model [5], in which the adversary wishes to hide its attack traces. Our results show that unpredictability is achieved even if a majority of nodes is under adversarial control or the adversary can evaluate VDFs significantly faster compared to the other nodes.

4.A Appendix: Additional Evaluation Results for the RandRunner Protocol

4.A.1 Simulation Results for Additional Protocol Parameterizations

As outlined in Sections 4.5 and 4.7.4 the selection of the parameter Δ_{VDF}, which determines the time parameter T for the VDF, is crucial for the unpredictability guarantees provided by RandRunner. In our simulation results presented in Section 4.7.4, we considered setting Δ_{VDF} such that $\Delta_{NET}/\Delta_{VDF} = 1/10$, a choice which works well across a wide range of scenarios. To further support the process of picking a suitable value for Δ_{VDF}, we provide additional simulation results in Figures 4.4–4.7. As before, we run our simulation over a period of 10^{10} rounds for each parameter set and consider both types of adversaries, i.e., an attacker which (i) does and (ii) does not want to hide its traces. We fix $\Delta_{NET} = 1$ and vary Δ_{VDF}, as the simulation results only depend on the relation $\Delta_{NET}/\Delta_{VDF}$ of the parameters Δ_{NET} and Δ_{VDF}. In general, we observe that increasing Δ_{VDF} compared to Δ_{NET} strengthens the protocol's unpredictability guarantee, while at the same time introducing longer delays whenever a leader fails or withholds an output on purpose. The bigger the adversarial strength, i.e., the fraction of adversarial nodes p_A and their advantage in computation speed compared to the honest nodes α, the more important is it to select higher values for Δ_{VDF}. Regarding the covert adversary model we analyzed in Section 4.7.4, Figure 4.8 further illustrates the correspondence between the protocol parameters regarding the security bound $\Delta_{NET} < \Delta_{VDF}/\alpha \cdot p_H$ (Inequality 4.17).

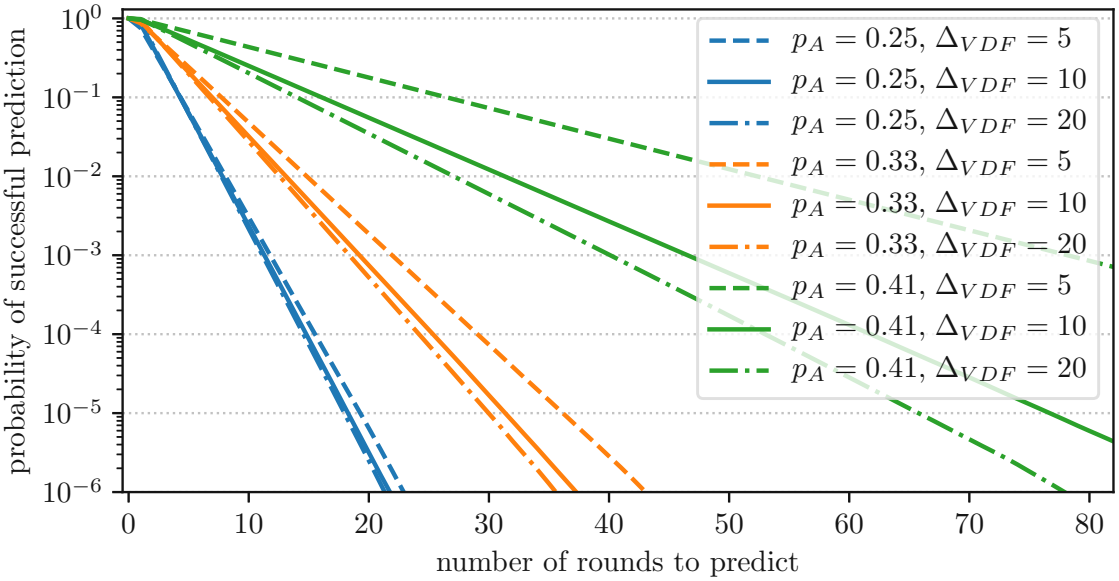

Figure 4.4: Simulation of RandRunner-RS' unpredictability guarantees for parameters $\alpha = 1, \Delta_{NET} = 1$ and different values of p_A and Δ_{VDF}

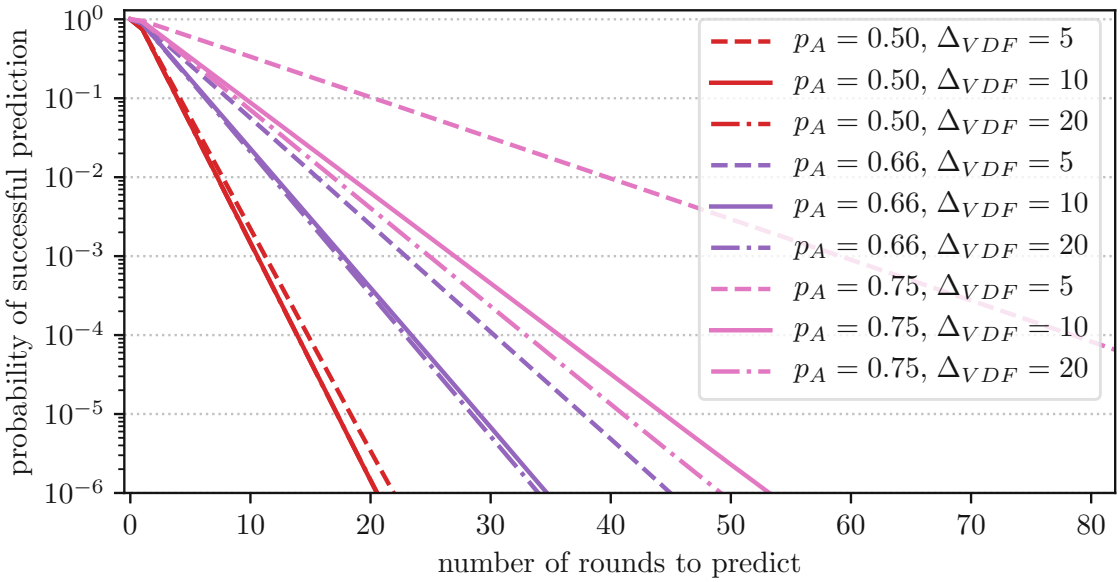

Figure 4.5: Simulation of RandRunner-RS' unpredictability guarantees against a covert adversary for parameters $\alpha = 1, \Delta_{NET} = 1$ and different values of p_A and Δ_{VDF}

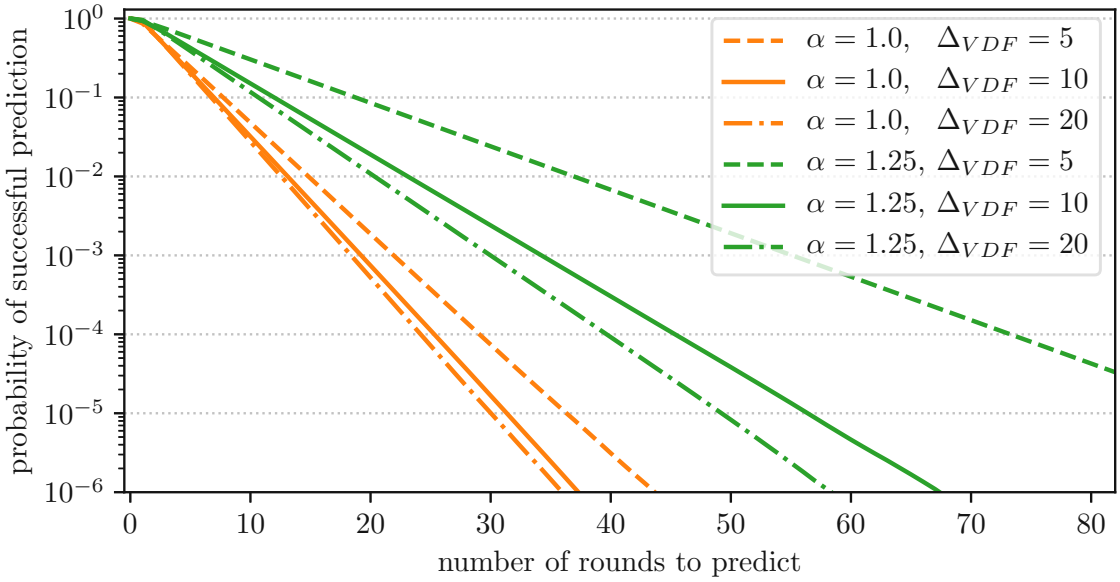

Figure 4.6: Simulation of RandRunner-RS' unpredictability guarantees for parameters $p_A = 0.33, \Delta_{NET} = 1$ and different values of α and Δ_{VDF}

4.A.2 Recovery from Asynchronous Network Conditions

We recall that RandRunner relies on network synchrony to ensure the unpredictability guarantees described in Section 4.7.4. Therefore, during periods of asynchrony, i.e., in

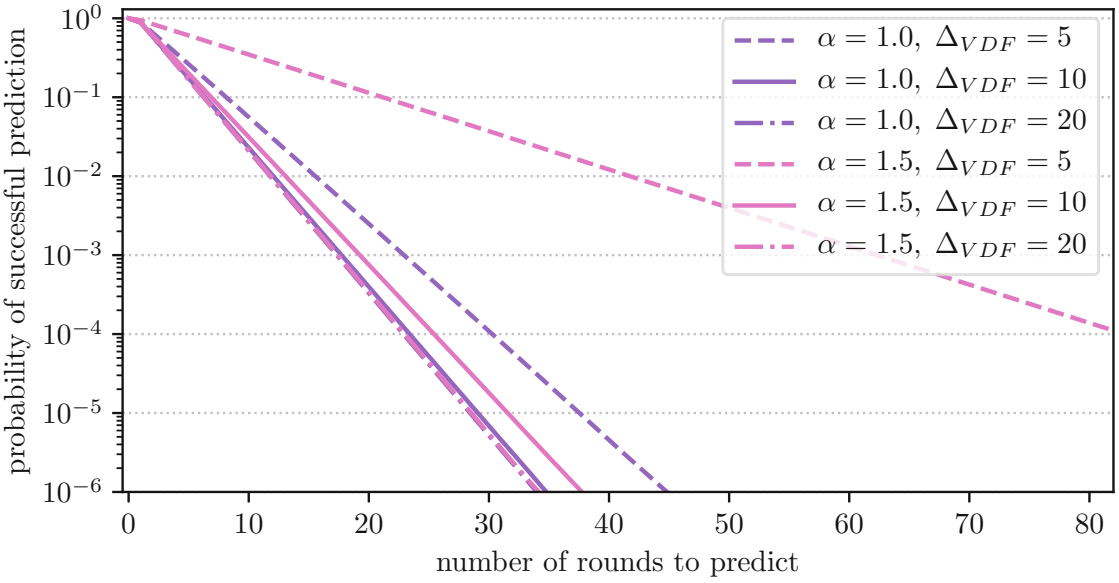

Figure 4.7: Simulation of RandRunner-RS' unpredictability guarantees against a covert adversary for parameters $p_A = 0.66, \Delta_{NET} = 1$ and different values of α and Δ_{VDF}

situations in which correct nodes cannot disseminate message within Δ_{NET} seconds, the protocol's unpredictability guarantees are gradually weakened. However, by design, RandRunner ensures liveness and consistency even during periods in which correct nodes cannot communicate with each other at all. During periods of asynchrony an adversary can increase its advantage (in terms the of number of random beacon output it can predict), whereas honest nodes catch up and RandRunner regains its unpredictability guarantees quickly once network connectivity is restored. In particular, this is the case when we consider a perfectly coordinated adversary which is not affected by the network delays or is itself responsible for the asynchronous network conditions. Considering this worst case, our simulation results in Figures 4.9 and 4.10 show how quickly the original unpredictability guarantees are restored after the network conditions normalize. We observe that the recovery time required increases linearly with the duration of the asynchronous period. Consequently, short periods of asynchrony have very little effect on the provided guarantees, whereas the protocol can still recover rather quickly even from long-lasting asynchronous network conditions. We note that in practice we only expect long-lasting asynchronous periods in extremely unlikely circumstances. In any case, a client using the produced random numbers is likely to notice the problem due to the temporary slowdown of the protocol and can consequently take appropriate countermeasures on the application layer, e.g., it may require a longer delay prior to the use of future outputs.

For our simulations we consider different parameterizations of RandRunner-RS, vary the duration of network outages (in multiples of the Δ_{NET} parameter), and plot the mean time until unpredictability guarantees are restored, with the standard deviation

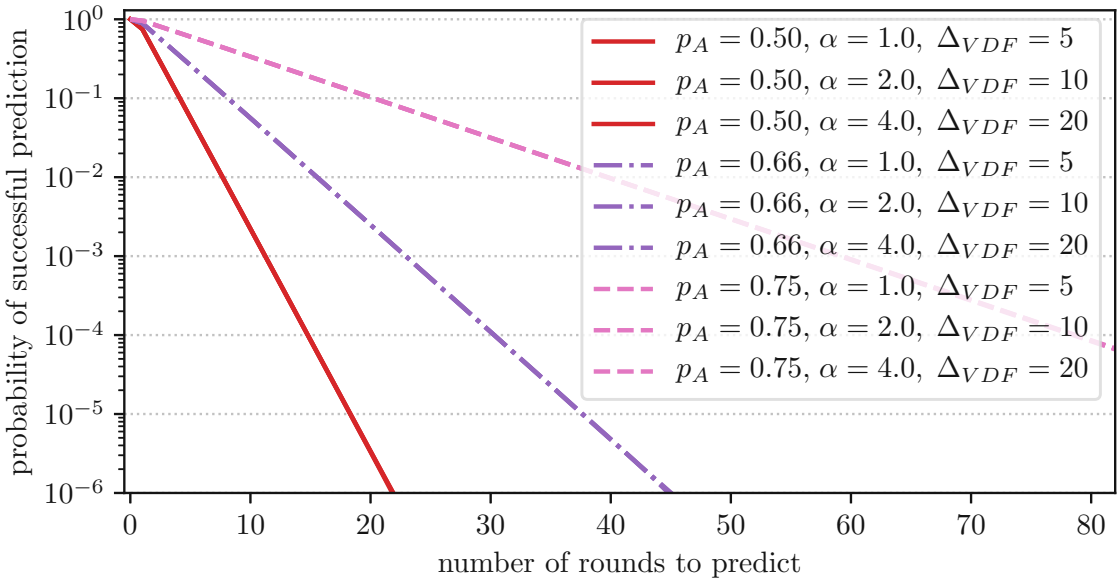

Figure 4.8: Simulation of RandRunner-RS' unpredictability guarantees against a covert adversary for parameter $\Delta_{NET} = 1$, highlighting the relations between p_A, α and Δ_{VDF}

highlighted. Concretely, we report the average recovery time (y-axis) of 100000 simulation runs for each outage duration (x-axis). In in each run, we simulate a network outage for the given duration at a random point in time. Considering the (theoretical) worst case, we assume that during the network outage/attack correct nodes cannot communicate with each other at all, yet the adversary can perfectly coordinate its actions and does not mind being detected during the attack.

4.A.3 Comparison of Probabilistic Unpredictability Guarantees

We omitted to present simulation results for RandRunner-RR in the main part of this chapter, as we have provided a formal proof for the provided unpredictability guarantees. However, in addition to the bounds proven in Section 4.7.4, RandRunner-RR also provides stochastic guarantees similar to RandRunner-RS. In general, we observe that the probabilistic guarantees of RandRunner-RR approach the guarantees RandRunner-RS provides with an increasing number of participants n considering equivalent scenarios. In other words, the probabilistic guarantees of RandRunner-RS give an upper bound for the (stronger) guarantees of RandRunner-RR. This is further illustrated in Figure 4.11, which also highlights RandRunner-RR's proven absolute bound of $d = 8$ rounds for the given example with $n = 24, f = 8$ nodes.

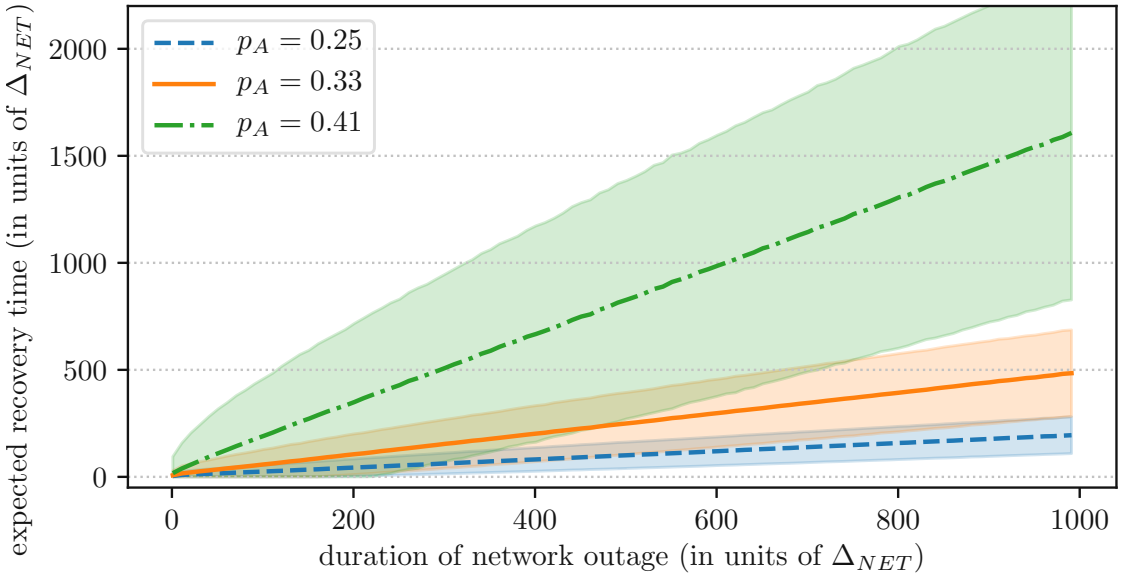

Figure 4.9: Mean time and standard deviation for recovery of RandRunner-RS' unpredictability after a network outage ($\Delta_{NET}/\Delta_{VDF} = 1/10$, $\alpha = 1.0$)

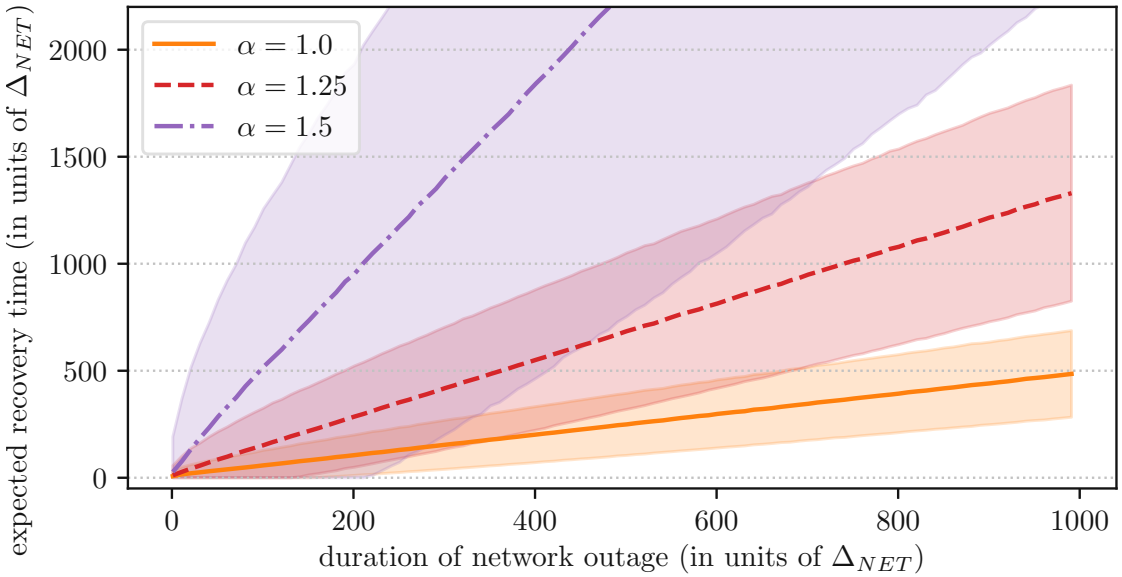

Figure 4.10: Mean time and standard deviation for recovery of RandRunner-RS' unpredictability after a network outage ($\Delta_{NET}/\Delta_{VDF} = 1/10$, $p_A = 0.33$)

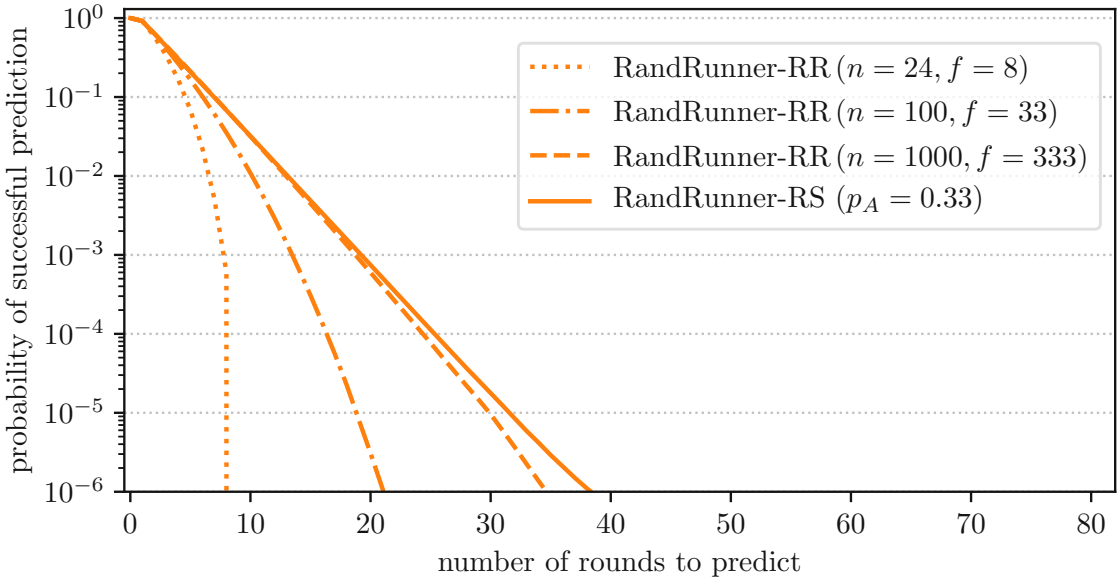

Figure 4.11: Comparison of RandRunner-RR/RS's probabilistic unpredictability guarantees ($\Delta_{NET}/\Delta_{VDF} = 1/10$, $\alpha = 1.0$)

4.B Appendix: RandRunner Notation Reference

Table 4.1: RandRunner notation reference – symbols

Symbol	Description
n	number of nodes running the protocol
f	number of adversarial / Byzantine nodes
$\alpha \geq 1$	adversaries VDF computation speed relative to the correct nodes
$\mathcal{P}$	set of participants running the protocol
$\mathscr{P}$	set of verified public parameters
$\mathscr{P}^*$	set of public parameter prior to verification
$r, s \geq 1$	some protocol round as specified by the context
d, v, w	number of rounds as specified by the context
R_0	initial random seed for the protocol
R_r	protocol output at round r
$i \in \mathcal{P}$	some node running the protocol as specified by the context
$\ell_r \in \mathcal{P}$	leader of round r
pp_i	public parameters for node i's VDF
sk_i	secret key / trapdoor for node i's VDF
Δ_{NET}	network propagation delay (between correct nodes)
Δ_{VDF}	correct nodes' upper bound for the computation time of $Eval$ (the VDF parameter T is set accordingly)
Δ_{VDF}/α	adversary's lower bound for the computation time of $Eval$
p_A	fraction of adversarial nodes (f/n)
p_H	fraction of honest/correct nodes $(1 - f/n)$
λ_A, λ_H	rate of progress for the adversarial and honest/correct nodes
Δ_w	RandRunner-RR's worst case completion time of n consecutive protocol rounds for correct nodes
$\tilde{P}$	randomized sequence of the set of participants $\mathcal{P}$
$\tilde{P}[j]$	j^{th} element of $\tilde{P}$ using 0-based indexing
k^{th} period	the sequence of rounds $(k-1)n+1, (k-1)n+2, ..., kn$

Table 4.2: RandRunner notation reference – algorithms

Algorithm	**Description**
$Setup(\lambda) \rightarrow pp$	setup function for a (general) VDF
$Setup(\lambda) \rightarrow (pp, sk)$	setup function for a trapdoor VDF
$VerifySetup(\lambda, pp) \rightarrow \{accept, reject\}$	verification algorithm for the parameters generated by $Setup(\cdot)$
$Eval(pp, x, T) \rightarrow (y, \pi)$	VDF evaluation algorithm (without knowledge of the trapdoor)
$TrapdoorEval(pp, x, T, sk) \rightarrow (y, \pi)$	VDF evaluation algorithm with knowledge of the trapdoor
$Verify(pp, x, T, y, \pi) \rightarrow \{accept, reject\}$	verification algorithm for the VDF evaluation
$H_{in} : \{0,1\}^{256} \rightarrow \mathcal{X}$	cryptographic hash function mapping a 256-bit string to the input space of the VDF
$H_{out} : \mathcal{Y} \rightarrow \{0,1\}^{256}$	cryptographic hash function mapping a VDF output to a 256-bit string

Table 4.3: RandRunner notation reference – VDFs

Symbol	**Description**
$\mathcal{X}$	input space of the VDF, $\mathcal{X} = QR_N^+$ in our case
$\mathcal{Y}$	output space of the VDF, $\mathcal{Y} = QR_N^+$ in our case
$\mathcal{PP}$	public parameter space of the VDF
$T \in \mathbb{N}$	time parameter of the VDF (number of iterations)
$x \in \mathcal{X}$	input to the VDF
$y \in \mathcal{Y}$	output of the VDF
π	correctness proof for the VDF output
$pp \in \mathcal{PP}$	public parameters of the VDF
p, q	large safe primes
N	RSA modulus
π_N	proof that N is a product of two safe primes of size $\lambda_{RSA}/2$
QR_N^+	group of signed quadratic residues modulo N
λ	security parameter
λ_{RSA}	security parameter for an RSA-based VDF

CHAPTER 5

Conclusion

In the last decade, we have observed a tremendous growth and interest in the field of public distributed ledger platforms. The early ideas surrounding the cryptocurrency Bitcoin led to a multi-trillion dollar field of platforms and projects that now reaches far beyond the scope of a digital payment system. Inspired by these developments, this thesis covers a wide range of existing, as well as newly developed, approaches for the generation of distributed randomness – a crucial component used in the design of many current and expected designs for distributed ledgers which do not rely on Proof-of-Work as the underlying mechanism to drive consensus decisions.

5.1 Highlights of our Research Contributions

During the work on this thesis we developed two unique protocols for the secure provisioning of randomness in a distributed setting as well as a protocol for distributed key generation crucial for the setup of threshold signatures based randomness beacons and other cryptographic protocols.

HydRand, our first protocol design presented at the 2020 IEEE Symposium on Security and Privacy, follows and advances upon prior work [6, 38, 78, 114] on randomness beacons based on publicly-verifiable secret sharing (PVSS). It operates in a stand-alone fashion and continuously outputs fresh randomness at regular intervals. Although it is a leader based protocol, a round's leader can neither bias the randomness of a particular round, nor prevent the randomness from becoming available to all parties by the end of that round. Using a pipelined approach and by interleaving the developed Byzantine fault tolerant (BFT) consensus algorithm with the distribution of the secret shares required for the PVSS protocols, we can improve the communication complexity of the overall protocol by an order of magnitude. This improvement allows us to obtain an efficient and stand-alone design. Together with the minimal requirements for protocol setup as well

as excellent latency and throughput characteristics shown by performance testing our prototype implementation, we see HydRand ready to support many interesting use cases.

Our second randomness beacon protocol, RandRunner, was accepted at the Network and Distributed System Security Symposium (NDSS 2021). While aiming for the similar security guarantees and properties (i.e., availability, unpredictability, bias-resistance, public-verifiability, guaranteed output delivery and the avoidance of an initial trusted setup) as HydRand, RandRunner's design is based on a different cryptographic primitive, namely trapdoor verifiable delay functions (T-VDFs) with strong uniqueness. Since VDFs have been introduced by Boneh et al. [18] in 2018, they have sparked a tremendous amount of interest within the various research communities, which has led to a range of practical designs for VDFs with different properties. These designs include VDFs with trapdoors, i.e., VDFs which can be efficiently evaluated by parties which are in possession of a certain secret (similar to a private key used for digital signatures). Unfortunately, we find that all such designs we are aware of cannot guarantee uniqueness of the VDF's output for a particular input against parties with knowledge of the trapdoor. In other words, the trapdoor can be used to forge a proof which falsely asserts that a certain output is the result of evaluating the VDF on a particular input. To prevent this kind of attack, we introduce the notion of T-VDFs with the property of strong uniqueness and give the first instantiation of such a T-VDF. Despite being of independent interest, for example, as a commitment scheme with enforceable revealment as we discuss further in Section 5.3, this advancement is of crucial importance for the construction of the RandRunner protocol, in particular to ensure that its strong bias-resistance guarantees hold. RandRanner also achieves the other main properties expected from a randomness beacon. In particular, it is very communication efficient as a single message broadcasted per round advances the protocol to the next round. This allows RandRunner to be scaled to a large number of nodes. Inherited by the use of VDFs, RandRunner's system model requires network synchrony. In a period where this synchrony assumption is violated, this leads to a degradation of the unpredictability guarantees (in particular considering the worst case of a fully adversarially controlled network). However, periods of asynchronous network communication are handled gracefully by the protocol, i.e., consistency and bias-resistance are always guaranteed, and the protocol automatically regains its unpredictability guarantees quickly after synchronous communication is reestablished. Assuming synchrony and limiting the adversary to covert/stealthy attacks only, another unique property RandRunner can provide is that unpredictability can even be ensured in scenarios where a majority of nodes are under adversarial control.

Considering EthDKG, our third protocol designed within the scope of this dissertation, we see our main contribution in bridging the gap between the theoretical research results in the field of distributed key generation (DKG) and practical systems which benefit from readily accessible implementations of DKG protocols. Building upon the popular DKG protocol design by Gennaro et al. [63, 64], we implement a range of optimizations which allows us to obtain an efficient, scalable and practical design. Our protocol design consists of two main parts: a smart contract deployed on a platform

like Ethereum and a client implementation interacting with this smart contract. One of the key characteristics of our design is that all steps of the key generation process can be verified within the smart contract itself. This removes another central point of trust and enables easy interoperability with the applications using the DKG protocol. Following the developments in the cryptocurrency ecosystems, we further find more and more platforms have already adopted or are in the process of adopting the Ethereum Virtual Machine (EVM) as the base layer for smart contract execution. Therefore, with our protocol being designed for execution on the EVM, a range of other platforms beyond Ethereum directly benefit from the availability of our DKG protocol.

With HydRand and RandRunner we added two novel protocols to the landscape of solutions for generating distributed randomness, while EthDKG supports the setup of existing threshold signature based approaches in this domain. To navigate the available solutions and aid the selection process for a particular use case, we also see the broad comparisons between existing solutions (among either other and compared to our protocol) as additional important research contributions. Summarizing our findings in this regard, we find that the underlying models, assumptions, and cryptographic techniques as well as the achieved security properties vary greatly, depending on the design at hand. While certain approaches clearly improve upon others, at the current state of the field, we cannot identify a single approach which outperforms all other approaches in all aspects. From the experience we gained during the work on this thesis, although of course difficult to prove, it appears unlikely that such a universally superior approach can exist at all. Hence, the decision on which approach is most suitable for a particular use case of interest has to be taken on a case-by-case basis. For example, the designs for distributed randomness based on verifiable random functions (VRFs), as employed in AlgoRand [42] or Ouroboros Praos [46], can be added to existing systems with low overhead, but fail to provide the property of bias-resistance, as there is always one (last) party which may choose to withhold a VRF computation result. The studied approaches using publicly-verifiable secret sharing (PVSS) generally achieve a wide range of desirable properties using well established cryptographic assumptions at the cost of a high, often cubic, communication complexity. Within this set of protocols, we improved theoretical communication complexity to be quadratic with our design HydRand, and demonstrate the practicability of the protocol with our prototypical implementation. Designs using unique threshold signatures, typically BLS signatures [24], as described by the early work of Cachin et al. [33] or later used in protocols such as Dfinity [72], are bias-resistant, quite communication efficient and achieve unpredictability quickly. The main drawbacks of these approaches are the reliance on less established cryptographic assumptions (pairing-based cryptography), as well as the requirement for a trusted dealer or distributed key generation (DKG) protocol, which complicates the setup and reconfiguration of the protocol. In this context, our DKG protocol named EthDKG can help the bootstrapping process, as it can readily be used on existing smart contract platforms such as Ethereum. Finally, we designed RandRunner, a new distributed randomness beacon based on the cryptographic primitive of a trapdoor verifiable delay function (T-VDF) with strong uniqueness which we introduce alongside the protocol

itself. Compared to prior VDF-based designs, RandRunner does not rely on Proof-of-Work or another separate approach for entropy collection and agreement, but rather leverages the property of strong uniqueness of the used T-VDFs to generate an essentially predetermined, yet unpredictable sequence of random numbers. Compared to threshold cryptographic approaches which combine values from a majority of participants in each protocol step, RandRunner produces a new output as a leader disseminates a single message in the network. As a drawback, unpredictability considering a colluding adversary is not achieved as quickly as in approaches using threshold signatures. On the other hand, however, the design has major advantages in terms of communication complexity and scalability, while remaining simple and based on well established cryptographic assumptions. Another unique property that sets RandRunner apart from other designs is the security considering a covert adversary [5]. Analyzing the protocol in this model, where a covert adversary wishes to hide traces of any ongoing attacks, we show that unpredictability can be guaranteed even under circumstances where a majority of the participants are controlled by the adversary.

5.2 Research Impact

Observing the domain of distributed ledger technologies, in particular the different cryptocurrency ecosystems, during the period of working on this dissertation, we already observe a shift away from Proof-of-Work towards alternative protocol designs. We expect this trend to continue with few (unfortunate) exceptions. The use of alternatives to Proof-of-Work as a consensus mechanism is not only mandated by the avoidance of the tremendous consumption of electrical energy used but also to improve upon the scalability, reliability and security of these systems. From the theoretical needs for randomness in the design of these protocols, towards a large body of deployed systems using distributed randomness protocols at the core of their designs, e.g., the PVSS-based approach in the early Ouroboros variant, VRF-based protocols such as Cardano or AlgoRand, threshold signature based beacons as used in Dfinity and many permissioned BFT designs, or VDFs as found in the cryptocurrency Chia, randomness beacons have already successfully demonstrated their utility in this domain. Also Ethereum, the large smart contract platform by market capitalization at the time of writing, is actively looking for suitable primitives for their next generation Proof-of-Stake system. The Ethereum foundation, as well as other large organizations such as Tezos or Protocol Labs, actively support research efforts, in particular towards the design and implementation of VDF-based systems. Considering these developments, while of course hard to predict, we believe RandRunner, the design of our trapdoor VDF with strong uniqueness, or one of the many ideas from the protocol's design could well find its way into the next generation of systems in this domain.

EthDKG, as well as other competing distributed key generation (DKG) protocols, are vital for the setup of many threshold cryptographic protocols, the secure generation of keys for the use with the BLS signature scheme being one of the prime use cases. For this purpose, EthDKG is a particularly flexible protocol design. The threshold, i.e.,

the number of parties required to recover the generated secret or generate a threshold signature, can be specified arbitrarily between one and all nodes. This allows the tradeoff between liveness and safety to be set according to application specific requirements rather than being limited to thresholds of 1/3, 1/2 or 2/3 of the nodes which are regularly used in the context of consensus protocols. Additionally, EthDKG supports a large number of participating nodes. Its smart contract capabilities can be used to, for example, specify dynamic participation models as well as enforce the use of security deposits in a secure way, i.e., our protocol can guarantee that an honest participant cannot wrongfully be accused of misbehavior even if the protocol is executed by a dishonest majority. With its smart contract based design, EthDKG is also a prime candidate for being used for the setup of smart contracts which want to avoid a central controlling entity of the respective contract. This is essential to circumvent the repeatedly found bad practice of using proxy contracts, controlled by a single private key, to update the main contract functionally. Similarly, decentralized autonomous organizations (DAOs) or other contracts which need to hold crypto assets in shared custody, including protocol for cross-chain asset transfer, are key examples for potential use case scenarios for EthDKG.

With HydRand our primary target application scenario is concerned with permissioned systems, considering a range of use cases beyond the scope of public distributed ledgers. In the following, we highlight a few selected use cases which we believe can benefit from our randomness beacon protocols, in particular HydRand, with its rather simple design, integrated Byzantine fault tolerant (BFT) protocol, and readily available prototypical implementation. For a broader discussion of other use case scenarios we refer to Section 2.1 as well as a set of related works [114, 38, 25]. The first group of use cases we consider are applications that require the fair allocation of a limited amount of resources. For example, when 50 students register for a seminar limited to 30 participants, randomly assigning the 30 spots among the students seems to be (at least somewhat) better than the typical first-come-first-serve rule which comes to its limits when a large number of students try to register at the same time as soon as the registration opens. In this particular case, one may argue that trusting the university to make this random assignment in a fair manner is reasonable, this argument is more difficult to make in other cases – opening the case for distributed protocols like HydRand. Consider, for example, the assignment of a limited set of preschool spots in a year with a high number of applicants or the allocation of grants. While the governing bodies may claim the corresponding procedures are executed fairly, a range of factors (e.g., political interest, personal involvement, or corruption) may influence the discussions, which at least to some extent could benefit from a well defined distributed process supported by a protocol like HydRand. Another potential use case are common statistical methods where results observed from a randomly selected data sample are extrapolated. Clearly, the results of this extrapolation can only be as good as the initial selection process. Integrating randomness beacons into this process can help to improve transparency and reduce trust assumptions. Closely related are use cases in e-voting, for example, random-sample voting as recently described in a paper by D. Chaum [40]. Here a publicly-verifiable source of randomness is used to verifiably select a random subset of voters for polling on a particular issue.

5.3 Directions for Future Research

Regarding directions for future research, we observe that the overwhelming majority of protocols designed for the provisioning of distributed randomness rely on cryptographic assumptions which are considered insecure against adversaries that are in possession of general purpose quantum computers. There exist rather simple post-quantum secure solutions, for example, constructed using hash-chains where preimages of cryptographic hash functions are revealed over time by the protocol participants. Unfortunately, these simple solutions fail to achieve the key properties of bias-resistance and guaranteed output delivery. A promising way to obtain a high performance design which is secure against quantum computers would be to replace's RandRunner's RSA based construction for its T-VDF with strong uniqueness with a new post-quantum secure alternative. Similarly, approaches using BLS threshold signatures now could directly benefit from emerging post-quantum secure alternatives for threshold signing.

The strong uniqueness property of our T-VDF design allows for interesting use cases where the T-VDF is used as a commitment scheme with enforceable revealment. In this scenario, a (potentially adversarial) party first sets up a T-VDF and uses it to create a commitment to some value. At a later point in time, the committing party may choose to open the commitment using the T-VDF's trapdoor. In case the committing party is obliged to open the commitment, but fails to do so, another party, which is not in possession of the trapdoor, is nevertheless able to evaluate the VDF and thus recover the unique previously committed value after a specified amount of time has passed. Given a commitment which has been found to be valid initially, the success of the recovery is guaranteed and independent of the original issuer, for example, the failure or the deliberate adversarial behavior of the committing node. We see the discovery of new application scenarios for such a commit-reveal protocol, for example, within the landscape of smart contracts, as well as other use cases or improvements of T-VDFs with the strong uniqueness property as exciting topics for further research.

Another particularly interesting use case for distributed randomness in the context of distributed ledgers is leader selection. At the current state of the available practical designs, there is a trade-off between the leader's secrecy and the ability to guarantee leader uniqueness. In this regard, secrecy refers to the ability that at first only the leader itself is aware that it is indeed in the leader role in a particular protocol round, whereas it may choose to prove this fact to other participants. A common implementation supporting this notion of leader secrecy is based upon the use of verifiable random functions (VRFs), executed independently by the protocol's participants. A participant is then considered the leader in a particular round, if the leader's VRF, computed on some prescribed input, leads to an output of a particular form, e.g., a number below a certain threshold. Due to the independence of the VRF executions, using such an approach however frequently leads to multiple or no leaders selected for a particular round – leader uniqueness cannot be ensured. The opposite issue occurs for the many other approaches, first producing a globally agreed unique random beacon value which is then used to uniquely select a leader among the possible candidates. Here uniqueness is achieved, whereas the leader's

identity is immediately revealed and leader secrecy can consequently not be ensured. While one may also find arguments opposing leader secrecy, for example, considering the leader's accountability to be more important than its secrecy, achieving leader secrecy and uniqueness simultaneously is certainly an interesting future research challenge and one where we can already observe the first results [21]. Similar to the recovery procedure used in HydRand's protocol design, the use of threshold cryptographic techniques may allow for leader secrecy and accountability at the same time. In this case, a leader's identity remains secret until it fulfills its duties, or, if it fails to do so, it is revealed by a collaborative effort of the other parties.

Finally, the quest of how to implement and further improve upon current state-of-the-art designs for public distributed ledgers remains ongoing. With this thesis, we set out to tackle the crucial aspect of distributed randomness within this larger domain. However, with many open questions remaining, we are excited and look forward to discovering and further supporting the many interesting current and future developments in this field.

List of Figures

List of Tables

Bibliography

[1] Ben Adida. Helios: Web-based open-audit voting. In *17th USENIX Security Symposium (USENIX Security '08)*, pages 335–348. USENIX Association, 2008.

[2] Mustafa Al-Bassam. Implementation of elliptic curve operations on G2 for alt_bn128 in Solidity. Online at https://github.com/musalbas/solidity-BN256G2 (accessed: 2021-05-11), 2019.

[3] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, and Julio César López-Hernández. Faster explicit formulas for computing pairings over ordinary curves. In *30th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '11)*, pages 48–68. Springer, 2011.

[4] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. A survey of attacks on ethereum smart contracts (SoK). In *6th International Conference on Principles of Security and Trust (POST '17), Held as Part of the European Joint Conferences on Theory and Practice of Software (ETAPS '17)*, pages 164–186. Springer, 2017.

[5] Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In *4th Theory of Cryptography Conference (TCC '07)*, pages 137–156. Springer, 2007.

[6] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. Winning the caucus race: Continuous leader election via public randomness. arXiv preprint arXiv:1801.07965, 2018.

[7] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In *25th ACM Conference on Computer and Communications Security (CCS '18)*, pages 913–930. ACM, 2018.

[8] Thomas Baignères, Cécile Delerablée, Matthieu Finiasz, Louis Goubin, Tancrède Lepoint, and Matthieu Rivain. Trap me if you can – million dollar curve. Cryptology ePrint Archive, Report 2015/1249, 2015.

[9] Razvan Barbulescu and Sylvain Duquesne. Updating key size estimations for pairings. *Journal of Cryptology*, (4):1298–1336, 2019.

[10] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *1st ACM Conference on Computer and Communications Security (CCS '93)*, pages 62–73. ACM, 1993.

[11] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *2nd ACM Symposium on Principles of Distributed Computing (PODC '83)*, pages 27–30. ACM, 1983.

[12] Iddo Bentov, Ariel Gabizon, and David Zuckerman. Bitcoin beacon. arXiv preprint arXiv:1605.04559, 2016.

[13] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016.

[14] Daniel J. Bernstein, Tanja Lange, and Ruben Niederhagen. Dual EC: A standardized back door. In *The New Codebreakers*, pages 256–281. Springer, 2016.

[15] Jean-Luc Beuchat, Jorge Enrique González-Díaz, Shigeo Mitsunari, Eiji Okamoto, Francisco Rodríguez-Henríquez, and Tadanori Teruya. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. In *4th International Conference on Pairing-Based Cryptography (Pairing '10)*, pages 21–39. Springer, 2010.

[16] George Robert Blakley. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge*, pages 313–318. IEEE Computer Society, 1979.

[17] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *SIGACT News*, (1):23–27, 1983.

[18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *38th Annual International Cryptology Conference (CRYPTO '18)*, pages 757–788. Springer, 2018.

[19] Dan Boneh, Benedikt Bünz, and Ben Fisch. A survey of two verifiable delay functions. Cryptology ePrint Archive, Report 2018/712, 2018.

[20] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *24th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '18)*, pages 435–464. Springer, 2018.

[21] Dan Boneh, Saba Eskandarian, Lucjan Hanzlik, and Nicola Greco. Single secret leader election. In *2nd ACM Conference on Advances in Financial Technologies (AFT '20)*, pages 12–24. ACM, 2020.

[22] Dan Boneh and Matthew K. Franklin. Efficient generation of shared RSA keys (extended abstract). In *17th Annual International Cryptology Conference (CRYPTO '97)*, pages 425–439. Springer, 1997.

[23] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *22nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '03)*, pages 416–432. Springer, 2003.

[24] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *7th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '01)*, pages 514–532. Springer, 2001.

[25] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. On bitcoin as a public randomness source. Cryptology ePrint Archive, Report 2015/1015, 2015.

[26] Johannes Buchmann and Hugh C. Williams. A key-exchange system based on imaginary quadratic fields. *Journal of Cryptology*, (2):107–118, 1988.

[27] Vitalik Buterin. On slow and fast block times. Online at https://blog.ethereum.org/2015/09/14/on-slow-and-fast-block-times/ (accessed: 2021-05-11), 2015.

[28] Vitalik Buterin. How should I handle blockchain forks in my DApp? Online at https://ethereum.stackexchange.com/questions/183/how-should-i-handle-blockchain-forks-in-my-dapp/203 (accessed: 2021-05-11), 2016.

[29] Vitalik Buterin. Validator ordering and randomness in PoS. Online at https://vitalik.ca/files/randomness.html/ (accessed: 2018-04-24), 2016.

[30] Vitalik Buterin. Randao++. Online at https://redd.it/4mdkku (accessed: 2021-05-11), 2017.

[31] Vitalik Buterin and Christian Reitwiessner. EIP 197: Precompiled contracts for optimal ate pairing check on the elliptic curve alt_bn128. Online at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-197.md (accessed: 2021-05-11), 2017.

[32] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. Proofs-of-delay and randomness beacons in ethereum. In *1st IEEE Security and Privacy on the Blockchain Workshop (S&B '17)*, 2017.

[33] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *Journal of Cryptology*, (3):219–246, 2005.

[34] Jan Camenisch and Markus Michels. Proving in zero-knowledge that a number is the product of two safe primes. In *18th Annual International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, pages 107–122. Springer, 1999.

[35] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. *Technical report/Dept. of Computer Science, ETH Zürich*, 1997.

[36] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *19th Annual International Cryptology Conference (CRYPTO '99)*, pages 98–115. Springer, 1999.

[37] Antonio Salazar Cardozo and Zachary Williamson. EIP 1108: Reduce alt_bn128 precompile gas costs. Online at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1108.md (accessed: 2021-05-11), 2018.

[38] Ignacio Cascudo and Bernardo David. SCRAPE: scalable randomness attested by public entities. In *15th International Conference Applied Cryptography and Network Security (ACNS '17)*, pages 537–556. Springer, 2017.

[39] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, (4):398–461, 2002.

[40] David Chaum. Random-sample voting. Online at https://rsvoting.org/whitepaper/white_paper.pdf (accessed: 2021-11-15), 2016.

[41] David Chaum and Torben P. Pedersen. Wallet databases with observers. In *12th Annual International Cryptology Conference (CRYPTO '92)*, pages 89–105. Springer, 1992.

[42] Jing Chen and Silvio Micali. Algorand. arXiv preprint arXiv:1607.01341, 2016.

[43] Arka Rai Choudhuri, Matthew Green, Abhishek Jain, Gabriel Kaptchuk, and Ian Miers. Fairness in an unfair world: Fair multiparty computation from public bulletin boards. In *24th ACM Conference on Computer and Communications Security (CCS '17)*, pages 719–728. ACM, 2017.

[44] Jeremy Clark and Urs Hengartner. On the use of financial data as a random beacon. In *9th Electronic Voting Technology Workshop / Workshop on Trustworthy Elections (EVT/WOTE '10)*. USENIX Association, 2010.

[45] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains (A position paper). In *3rd Workshop on Bitcoin and Blockchain Research (BITCOIN '16), Held as part of the 20th International Conference on Financial Cryptography and Data Security (FC '16)*, pages 106–125. Springer, 2016.

[46] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *37th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '18)*, pages 66–98. Springer, 2018.

[47] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In *6th ACM Symposium on Principles of Distributed Computing (PODC '97)*, pages 1–12. ACM, 1987.

[48] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. Tor: The second-generation onion router. In *13th USENIX Security Symposium (USENIX Security '04)*, pages 303–320. USENIX, 2004.

[49] Danny Dolev, Cynthia Dwork, and Larry J. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM (JACM)*, (1):77–97, 1987.

[50] Jason A. Donenfeld, Matt Mackall, and Theodore Ts'o. Linux source code drivers/char/random.c – a strong random number generator. Online at https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/drivers/char/random.c (accessed: 2021-11-20), 2017.

[51] Nico Döttling, Sanjam Garg, Giulio Malavolta, and Prashant Nalini Vasudevan. Tight verifiable delay functions. In *12th International Conference on Security and Cryptography for Networks (SCN '20)*, pages 65–84. Springer, 2020.

[52] Justin Drake. Minimal VDF randomness beacon. Online at https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566 (accessed: 2020-07-08), 2018.

[53] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, (2):288–323, 1988.

[54] Naomi Ephraim, Cody Freitag, Ilan Komargodski, and Rafael Pass. Continuous verifiable delay functions. In *39th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '20)*, pages 125–154. Springer, 2020.

[55] Ethereum network status. Online at https://ethstats.net/ (accessed: 2021-08-22), 2021.

[56] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th IEEE Symposium on Foundations of Computer Science (FOCS '87)*, pages 427–437. IEEE Computer Society, 1987.

[57] Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In *25th International*

Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT '19), pages 248–277. Springer, 2019.

[58] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *6th Annual International Cryptology Conference (CRYPTO '86)*, pages 186–194. Springer, 1986.

[59] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[60] Tore Kasper Frederiksen, Yehuda Lindell, Valery Osheter, and Benny Pinkas. Fast distributed RSA key generation for semi-honest and malicious adversaries. In *38th Annual International Cryptology Conference (CRYPTO '18)*, pages 331–361. Springer, 2018.

[61] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, (4):469–472, 1985.

[62] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *34th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '15)*, pages 281–310. Springer, 2015.

[63] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *18th Annual International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT '99)*, pages 295–310. Springer, 1999.

[64] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Revisiting the distributed key generation for discrete-log based cryptosystems. *RSA Security'03*, pages 89–104, 2003.

[65] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of pedersen's distributed key generation protocol. In *The Cryptographers' Track at the RSA Conference*, pages 373–390. Springer, 2003.

[66] Rosario Gennaro, Daniele Micciancio, and Tal Rabin. An efficient non-interactive statistical zero-knowledge proof system for quasi-safe prime products. In *5th ACM Conference on Computer and Communications Security (CCS '98)*, pages 67–72. ACM, 1998.

[67] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. On the security and performance of proof of work blockchains. In *23rd ACM Conference on Computer and Communications Security (CCS '16)*, pages 3–16. ACM, 2016.

[68] Mainak Ghosh, Miles Richardson, Bryan Ford, and Rob Jansen. A torpath to torcoin: Proof-of-bandwidth altcoins for compensating relays. In *7th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs '14)*, 2014.

[69] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *26th ACM Symposium on Operating Systems Principles (SOSP '17)*, pages 51–68. ACM, 2017.

[70] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. Herbivore: A scalable and efficient protocol for anonymous communication. Technical report, Cornell University, 2003.

[71] David Goulet and George Kadianakis. Random number generation during tor voting. *Tor's protocol specifications–Proposal*, 2015.

[72] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. arXiv preprint arXiv:1805.04548, 2018.

[73] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. Design science in information systems research. *MIS quarterly*, pages 75–105, 2004.

[74] Harry A. Kalodner, Steven Goldfeder, Xiaoqi Chen, S. Matthew Weinberg, and Edward W. Felten. Arbitrum: Scalable, private smart contracts. In *27th USENIX Security Symposium (USENIX Security '18)*, pages 1353–1370. USENIX Association, 2018.

[75] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking. Randomized rumor spreading. In *41st IEEE Symposium on Foundations of Computer Science (FOCS '00)*, pages 565–574. IEEE Computer Society, 2000.

[76] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *29th IEEE International Conference on Distributed Computing Systems (ICDCS '09), 22-26 June 2009, Montreal, Québec, Canada*, pages 119–128. IEEE Computer Society, 2009.

[77] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. Cryptology ePrint Archive, Report 2012/377, 2012.

[78] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *37th Annual International Cryptology Conference (CRYPTO '17)*, pages 357–388. Springer, 2017.

[79] Esteban Landerreche, Marc Stevens, and Christian Schaffner. Non-interactive cryptographic timestamping based on verifiable delay functions. In *24th International Conference on Financial Cryptography and Data Security (FC '20)*, pages 541–558. Springer, 2020.

[80] Arjen K. Lenstra and Benjamin Wesolowski. A random zoo: sloth, unicorn, and trx. Cryptology ePrint Archive, Report 2015/366, 2015.

[81] Mohammad Mahmoody, Caleb Smith, and David J. Wu. A note on the (im)possibility of verifiable delay functions in the random oracle model. Cryptology ePrint Archive, Report 2019/663, 2019.

[82] Matic network – scalable and instant blockchain transactions. Online at https://matic.network/ (accessed: 2021-05-11), 2021.

[83] Matic network | documentation | matic gas station. Online at https://docs.matic.network/docs/develop/tools/matic-gas-station/ (accessed: 2021-05-11), 2021.

[84] Alfred Menezes, Palash Sarkar, and Shashank Singh. Challenges with assessing the impact of NFS advances on the security of pairing-based cryptography. In *2nd International Conference on Cryptology and Malicious Security (Mycrypt '16)*, pages 83–108. Springer, 2016.

[85] Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th IEEE Symposium on Foundations of Computer Science (FOCS '99)*, pages 120–130. IEEE Computer Society, 1999.

[86] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, Dec 2008.

[87] Wafa Neji, Kaouther Blibech Sinaoui, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and Communication Networks*, (17):4585–4595, 2016.

[88] Orbs Network. DKG for BLS threshold signature scheme on the EVM using solidity. Online at https://github.com/orbs-network/dkg-on-evm (accessed: 2021-05-11), 2018.

[89] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *31st International Symposium on Distributed Computing (DISC '17)*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[90] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *37th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '18)*, pages 3–33. Springer, 2018.

[91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *11th Annual International Cryptology Conference (CRYPTO '91)*, pages 129–140. Springer, 1991.

[92] Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract). In *10th Workshop on the Theory and Application of of Cryptographic Techniques (EUROCRYPT '91)*, pages 522–526. Springer, 1991.

[93] Cécile Pierrot and Benjamin Wesolowski. Malleability of the blockchain's entropy. *Cryptography and Communications*, (1):211–233, 2018.

[94] Krzysztof Pietrzak. Simple verifiable delay functions. In *10th Innovations in Theoretical Computer Science Conference (ITCS '19)*, pages 60:1–60:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[95] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. Online at https://plasma.io/plasma.pdf (accessed: 2018-04-24), 2017.

[96] Michael O. Rabin. Randomized byzantine generals. In *24th IEEE Symposium on Foundations of Computer Science (FOCS '83)*, pages 403–409. IEEE Computer Society, 1983.

[97] Michael O. Rabin. Transaction protection by beacons. *Journal of Computer and System Sciences*, (2):256–267, 1983.

[98] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st Annual ACM Symposium on Theory of Computing (STOC '89)*, pages 73–85. ACM, 1989.

[99] randao.org. Randao: Verifiable random number generation. Online at https://randao.org/whitepaper/Randao_v0.85_en.pdf (accessed: 2021-11-25), 2017.

[100] Christian Reitwiessner. EIP 196: Precompiled contracts for addition and scalar multiplication on the elliptic curve alt_bn128. Online at https://github.com/ethereum/EIPs/blob/master/EIPS/eip-196.md (accessed: 2021-11-25), 2017.

[101] Eric Rescorla. Diffie-hellman key agreement method. *RFC 2631*, pages 1–13, 1999.

[102] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[103] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner: distributed randomness from trapdoor VDFs with strong uniqueness. In *28th Annual Network and Distributed System Security Symposium (NDSS '21)*. The Internet Society, 2021.

[104] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. RandRunner research artifacts. Online at https://github.com/PhilippSchindler/RandRunner (accessed: 2021-11-10), 2021.

[105] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. EthDKG: distributed key generation with ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985, 2019.

[106] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. HydRand: efficient continuous distributed randomness. In *41st IEEE Symposium on Security and Privacy (SP '20)*, pages 73–89. IEEE, 2020.

[107] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Python implementation of the HydRand protocol. Online at https://github.com/PhilippSchindler/HydRand (accessed: 2021-11-10), 2020.

[108] Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In *19th Annual International Cryptology Conference (CRYPTO '99)*, pages 148–164. Springer, 1999.

[109] David Schwartz, Noah Youngs, Arthur Britto, et al. The ripple protocol consensus algorithm. *Ripple Labs Inc White Paper*, (8):151, 2014.

[110] Adi Shamir. How to share a secret. *Communications of the ACM*, (11):612–613, 1979.

[111] Barak Shani. A note on isogeny-based hybrid verifiable delay functions. Cryptology ePrint Archive, Report 2019/205, 2019.

[112] Yonatan Sompolinsky and Aviv Zohar. Bitcoin's security model revisited. arXiv preprint arXiv:1605.09193, 2016.

[113] Mario Stipcevic and Çetin Kaya Koç. True random number generators. In *Open Problems in Mathematics and Computational Science*, pages 275–315. Springer, 2014.

[114] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *38th IEEE Symposium on Security and Privacy (SP '17)*, pages 444–460. IEEE Computer Society, 2017.

[115] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *37th IEEE Symposium on Security and Privacy (SP '16)*, pages 526–545. IEEE Computer Society, 2016.

[116] Carmela Troncoso, Marios Isaakidis, George Danezis, and Harry Halpin. Systematizing decentralization and privacy: Lessons from 15 years of research and deployments. *Proceedings on Privacy Enhancing Technologies*, (4):404–426, 2017.

[117] Jelle van den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. Vuvuzela: scalable private messaging resistant to traffic analysis. In *25th ACM Symposium on Operating Systems Principles (SOSP '15)*, pages 137–152. ACM, 2015.

[118] Benjamin Wesolowski. Efficient verifiable delay functions. In *38th Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT '19)*, pages 379–407. Springer, 2019.

[119] David Isaac Wolinsky, Henry Corrigan-Gibbs, Bryan Ford, and Aaron Johnson. Dissent in numbers: Making strong anonymity scale. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI '12)*, pages 179–182. USENIX Association, 2012.

[120] Gavin Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[121] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *38th ACM Symposium on Principles of Distributed Computing (PODC '19)*, pages 347–356. ACM, 2019.