

The Blockchain of Oz

Specifying Blockchain Failures for Scalable Protocols Offering Unprecedented Safety and Decentralization

Alejandro Ranchal-Pedrosa

*A thesis submitted to fulfill requirements for the degree of
Doctor of Philosophy*

School of Computer Science

Faculty of Engineering

The University of Sydney

September 2023

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Alejandro Ranchal Pedrosa
September 19, 2022

Authorship Attribution

The results presented in this dissertation were published in the following publications and can be found in the relevant chapters:

- (1) Chapter 2 provides background and preliminaries for the rest of the chapters, and thus contains material from [3.1], [3.2], [4.1], [4.2], [5.1], [5.2] and [6.1], along with some novel work that will be part of upcoming publications. The model and related work outlined in this chapter is designed by me, although inspired from recent works, and credited accordingly.
- (2) Chapter 3 contains material published in [3.1] and [3.2]. In [3.1], I was the author who proposed the attack, implemented it and measured its impact in the testnet, and proposed improvements and countermeasures. In [3.2], I proposed the solution, proved its correctness and complexity measures, and co-wrote its formalization and model.

[3.1] Cristina Pérez-Solà, Alejandro Ranchal-Pedrosa, Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, and Joaquin Garcia-Alfaro. “LockDown: Balance Availability Attack Against Lightning Network Channels”. In: *Financial Cryptography and Data Security*. 2020

[3.2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. “Platypus: Offchain Protocol Without Synchrony”. In: *IEEE NCA*. 2019

- (3) Chapter 4 contains material published in [4.1] and [4.2]. In both, I proposed the concept, model, theorems and proofs, and designed and analyzed all protocols.

[4.1] Alejandro Ranchal-Pedrosa and Vincent Gramoli. “Rational Agreement in the Presence of Crash Faults”. In: *IEEE International Conference on Blockchain (Blockchain)*. 2021

[4.2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. “TRAP: The Bait of Rational Players to Solve Byzantine Consensus”. In: *ACM AsiaCCS*. 2022

- (4) Chapter 5 contains material published in [5.1], and material under submission and available as preprint in [5.2]. This chapter is also extended with novel work not publicly

available before, that will be used as part of a journal publication. In [5.1], I proposed the concept, the novel fault model, formalized the model, proposed and designed the Basilic class of protocols, and proved its complexity and correctness, as well as the impossibility proofs. In [5.2], I proposed the concept, the fault model, formalized the problem, proved all theorems, implemented the system and attacks, tested them and gathered results, analyzed the impact of attacks and proposed a solution to minimize such impact. The additional work for the journal is entirely of my doing.

[5.1] Alejandro Ranchal-Pedrosa and Vincent Gramoli. “Basilic: Resilient Optimal Consensus Protocols With Benign and Deceitful Faults”. In: *IEEE CSF*. 2023

[5.2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. *ZLB: A Blockchain to Tolerate Colluding Majorities*. 2020. URL: <https://arxiv.org/abs/2007.10541>

- (5) Chapter 6 contains material published in [6.1], as well as additional work that builds upon previous chapters (specially Chapter 5), and we are thus awaiting publication of [4.2] before making the rest of the content of Chapter 6 available. I came up with the model, problem statements, protocol design and optimizations, analysis of safety and liveness, and implications to previous chapters.

[6.1] Alejandro Ranchal-Pedrosa and Vincent Gramoli. “Leveraging Democracy to Optimize Distributed Random Beacons”. In: *ACM ConsensusDay*. 2022

Supplementary technical reports with material included in this thesis containing additional information in their respective publications:

[3.2 – 2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. *Platypus: a Partially Synchronous Offchain Protocol for Blockchains*. 2019. URL: <https://arxiv.org/abs/1907.03730>

[4.1 – 2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. *Rational Agreement in the Presence of Crash Faults*. 2021. URL: <https://arxiv.org/abs/2111.01425>

[4.2 – 2] Alejandro Ranchal-Pedrosa and Vincent Gramoli. *TRAP: The Bait of Rational Players to Solve Byzantine Consensus*. 2021. URL: <https://arxiv.org/abs/2105.04357>

[5.1 – 1] Alejandro Ranchal-Pedrosa and Vincent Gramoli. *Basilic: Resilient Optimal Consensus Protocols With Benign and Deceitful Faults*. 2022. URL: <https://arxiv.org/>

[abs/2204.08670](#)

In addition to the statements above, in cases where I am not the corresponding author of a published item, permission to include the published material has been granted by the corresponding author.

Alejandro Ranchal Pedrosa

Signature:

Date: September 19, 2022

Associate Professor Vincent Charles Gramoli

Signature:

Date: September 19, 2022

(Auxiliary) Professor Alan Fekete

Signature:

Date: September 19, 2022

*This thesis is dedicated to my parents,
Ana Pedrosa Ruiz and Gabriel Ranchal Alcudia,
for their endless support and love for their children.*

Acknowledgments

A PhD is an endeavor that does not (and should not) go unnoticed to the PhD student and their surroundings. For the author of this thesis, this journey has consisted of many different challenges and joys, both in the personal and professional aspect. Despite my animadversion to look back and openly express feelings, it is only fair to acknowledge those who walked with me along the way.

First and foremost, I would like to thank the older members of my family. To my mother, Ana, despite sometimes loving me against my own interests, your human qualities are beyond describable, as I am sure anyone who ever meets you would recognize. To my dad, Gabriel, for your sacrifice and love for your children. To my siblings, Gabi and Tamara, for being excellent role models for your annoying younger brother, and for being there when most needed. I would also like to thank the younger members of my family, both my nephews, Alfonso and Alejandro, and my dogs, Akiles and Ulises, for bringing unending joy and happiness. I would also like to thank my extended family, specially Alfonso, Alba, Loli, Jesus and Andrea, for being there for me throughout the journey.

I would like to thank my supervisor, Vincent Gramoli, it has been an absolute pleasure working with you, and it is only through your supervision that I managed to improve myself beyond my own expectations. I must thank everyone in the Concurrent Systems Research Group, specially Chris, Gauthier, Daniel and Deepal. Special thanks to Chris Natoli, for being an outstanding 'veteran' PhD student and person, I cannot thank you enough for your insights.

It is also my pleasure to thank my friends, starting with my European friends, Borja, Hendrik, Mariajo, Anaïs, Lou, Mathilde, Jake, Philip, Tena, Victor, Guille, Sergio, Andres, Adri, Stef, Ali, Rasmus, Ida, Gabriella, Daniela, Isaac, Bea, Ana, Carlos, Jaime, Irina, Beth, Bryony, Simone, Nikolai, Monika, Ikenna and Karolina. Without your friendships I would have never taken the courage to keep traveling and improving my career. I can only hope that our paths keep crossing at least as often as they have. I would also like to thank the friends that became my Australian family, specially Alon, Nisha, Yotam, Rohit, Lola, Kuba, Rafa and Suki. You converted this strange pandemic times into much more than just a rough patch.

I cannot forget to thank my former supervisors, fellow students and colleagues, specially Sara Tucci and Manuel Carro, for your exceptional guidance and understanding in giving form to the potential of a young, bold student. Similarly, thank you to all my colleagues at the University of Sydney, fellow PhD students and postdocs, for sharing the stress and suffering involved in the making of this dissertation.

Finally, I wish to thank my partner, Mar, for enduring my idiosyncrasy, laughing at my terrible humor, and making an incredible effort at trying to help me through this journey in any way you could. Thank you for everything.

Abstract

Blockchains have starred an outstanding increase in interest from both business and research since Nakamoto's 2008 Bitcoin. Unfortunately, many questions in terms of results that establish upper-bounds, and of proposals that approach these bounds. Furthermore, the sudden hype surrounding the blockchain world has led to several proposals that are either only partially public, informal, or not proven correct.

The main contribution of this dissertation is to build upon works that steer clear of blockchain puffery, following research methodology. The works of this dissertation converge towards a blockchain that for the first time formally proves and empirically shows deterministic guarantees in the presence of classical Byzantine adversaries, while at the same time pragmatically resolves unlucky cases in which the adversary corrupts an unprecedented percentage of the system. This blockchain is decentralized and scalable, and needs no strong assumptions like synchrony.

For this purpose, we build upon previous work and propose a novel attack of synchronous offchain protocols. We then introduce Platypus, an offchain protocol without synchrony. Secondly, we present TRAP, a Byzantine fault-tolerant consensus protocol for blockchains that also tolerates up to less than half of the processes deviating. Thirdly, we present Basilic, a class of protocols that solves consensus both against a resilient-optimal Byzantine adversary and against an adversary controlling up to less than $2/3$ of combined liveness and safety faults. Then, we use Basilic to present Zero-loss Blockchain (ZLB), a blockchain that tolerates less than $2/3$ of safety faults of which less than $1/3$ can be Byzantine. Finally, we present two random beacon protocols for committee sortition: Kleroterion and Kleroterion⁺, that improve previous works in terms of communication complexity and in the number of faults tolerated, respectively.

Foreword

The title, *The Blockchain of Oz*, derives not only from the fact that *Oz* is often used to refer to Australia, where the author did the works of this dissertation, but also from the magical country introduced in the novel *The Wonderful Wizard of Oz*. Some characters of the novel are in a quest to find a missing piece of their own, such as a scarecrow, a cowardly lion, or a tin woodman, longing for brains, courage, or a heart, respectively. Other characters claim false promises, such as the so-called wizard who is actually a man pretending to have supernatural powers.

The current blockchain ecosystem greatly resembles this novel. First, because of the heterogeneity of its inhabitants. Second, because some of these blockchains, while already useful, lack important properties intrinsic to their design. Third, because the blockchain ecosystem is subject to a significant amount of false, unproven, or yet-to-be-fulfilled claims in an effort to gain funding and influence. The title thus depicts the journey of the author towards a blockchain that delivers its promises, with proven properties and claims.

Contents

List of Figures	xii
List of Tables	xiii
1 Introduction	1
1.1 Objectives	3
1.2 Contributions	5
1.3 Outline	6
2 Background & Preliminaries	9
2.1 Background	9
2.1.1 Consensus	9
2.1.2 Blockchain trilemma	11
2.1.3 Scalability	12
2.1.4 Security	13
2.1.5 Decentralization	14
2.2 Preliminaries	17
2.2.1 Fault model	17
2.2.2 Communication network	18
2.2.3 Authenticating messages	18
2.2.4 Send, receive and deliver	18
2.2.5 Adversary	19
2.2.6 Solving consensus	19
2.2.7 State machine replication	21
2.2.8 Committee sortition	23
2.2.9 Blockchain trilemma	24
3 Layer-2 Without Synchrony	29
3.1 Why layer-2 that rely on synchrony are more vulnerable	30
3.1.1 Abstract vulnerability	30
3.1.2 Lockdown attack	31
3.2 Model	36
3.3 Secure childchains without synchrony	37

3.3.1	Overview	38
3.3.2	Creating a Platypus chain	38
3.3.3	Closing a Platypus chain	39
3.3.4	Aborting a closing attempt	39
3.4	Correctness & optimal resilience	42
3.5	Theoretical analysis	47
3.6	Improvements & discussion	47
3.6.1	Crosschain payments	48
3.6.2	Platypus for sidechains	49
3.6.3	Attacks	50
3.7	Summary	51
4	Rationality for Blockchains' Consensus	53
4.1	Rational model	56
4.1.1	Robustness	59
4.1.2	Punishment strategy	60
4.1.3	Rational agreement	60
4.2	Impossibility results	60
4.2.1	Baiting strategies	61
4.2.2	Rational agreement is impossible without a baiting strategy	62
4.2.3	Impossibility in the presence of rational and crash players	64
4.3	TRAP: reaching rational agreement	65
4.3.1	Overview: consensus with a baiting strategy	65
4.3.2	Baiting component: the BFTCR protocol	67
4.3.3	Financial component: deposits & reward	72
4.4	Bridging the gap: crash and rational as Byzantine players	76
4.4.1	From Byzantine to crash players	76
4.4.2	From crash to Byzantine players	77
4.5	Summary	80
5	ZLB, a Blockchain Tolerating Colluding Majorities	83
5.1	Byzantine-deceitful-benign fault model	85
5.2	Impossibility of consensus in the BDB model	87
5.2.1	Impossibility bounds	88
5.2.2	Impossibility bounds per voting threshold	89
5.3	Basilic, resilient-optimal consensus in the BDB model	90
5.3.1	Actively accountable consensus problem	91
5.3.2	Basilic Internals	92
5.3.3	Basilic's fault tolerance in the BDB model	100
5.3.4	Basilic's correctness	101
5.3.5	Basilic's complexities	108
5.3.6	Solving eventual consensus with Basilic	110

5.4	The Zero-Loss Blockchain	112
5.4.1	Longlasting Blockchain	112
5.4.2	Slowly-adaptive adversary	113
5.4.3	Pool of process candidates	113
5.4.4	The Zero-Loss Blockchain	113
5.4.5	The Zero-Loss Blockchain proofs	121
5.4.6	Comparative table	125
5.4.7	Experimental evaluation	126
5.5	A Zero-Loss payment application	131
5.5.1	Assumptions	132
5.5.2	Theoretical analysis	133
5.5.3	Discussion on probabilistic synchrony	134
5.6	Summary	135
6	Kleroterion⁺, Randomness With Colluding Majorities	137
6.1	The need for a random beacon	138
6.1.1	The need to depreciate future iterations	138
6.1.2	Depreciating future iterations with a random beacon	139
6.2	Kleroterion: a democratic random beacon	141
6.2.1	Additional model	142
6.2.2	The Pinakion protocol	143
6.2.3	The (unoptimized) Kleroterion protocol	145
6.2.4	Optimizations and observations	152
6.2.5	Benefits of democratic, leader-based protocols	158
6.3	Kleroterion ⁺ : tolerating colluding majorities	159
6.3.1	Secure random beacon and accountable PVSS problems	159
6.3.2	Pinakion ⁺ and Kleroterion ⁺	161
6.4	Using Kleroterion ⁺ for committee sortition	162
6.4.1	Probability of randomness of the random beacon	164
6.4.2	Comparison with the state of the art	166
6.5	Summary	166
7	Conclusion	169
7.1	Outcome of Research Objectives	170
7.2	Future Work	172
	Notations	175
	Bibliography	179
A	Lockdown Attack	193
A.1	Attack design	193
A.1.1	Adversarial knowledge	195

A.1.2	AER minimization	195
A.1.3	TBT maximization	195
A.2	Experimental results	196
A.2.1	Simulation assumptions	197
A.2.2	Attack simulation results	198
A.3	Simnet network	201
A.4	Countermeasures to handle the attack	203
B	TRAP protocol: discussions	205
B.1	Extended example figure	205
B.2	Paying a reward at no cost to non-deviants	205
C	Basilic Additional Results	209
C.1	Impossibility of consensus without active accountability	209
C.2	Extended complexities of Basilic	209
D	ZLB Additional Results	211
D.1	Number of branches and deceitful ratio	211
D.2	Fixed superblock size	212
D.3	Bitmask of binary consensus attack	213
E	Discussion: safety vs. performance of Kleroterion⁺	215

List of Figures

3.1	Example of an attack in a synchronous offchain protocol.	31
3.2	Example scenario of the Lockdown attack.	33
3.3	Simple scenario of the Lockdown attack with adversary and external node. . . .	34
4.1	Example execution of the TRAP protocol.	55
4.2	Rational generals example.	63
5.1	Number of faults tolerated per voting threshold in the BDB model.	91
5.2	Basilic execution example for a committee of $n = 4$ processes.	93
5.3	Basilic's tolerance to faults per voting threshold, compared to previous work. .	102
5.4	ZLB architecture.	114
5.5	The phases of ASMR.	115
5.6	Throughput of ZLB compared to recent works.	127
5.7	Disagreements caused by attackers with artificial delays.	128
5.8	Time to detect, exclude and include processes; and time to catch up.	130
5.9	Disagreements caused by attackers with catastrophic, artificial delays.	131
5.10	Required finalization blockdepth w for zero loss.	134
6.1	Kleroterion execution example with $n = 4$ processes.	147
6.2	Kleroterion optimized aggregation example.	155
6.3	Kleroterion optimized RBV-broadcast example.	155
6.4	Example attack on Kleroterion ⁺ by adversary controlling t_s faults.	163
6.5	Probability of the adversary breaking randomness of the random beacon.	165
6.6	Committee size per percentage of faulty processes for safety.	167
A.2.1	Number of blocks for which the Lockdown attack locks balances.	200
A.3.2	Simnet scenarios maximizing A 's locked balanced in one payment.	202
B.1.1	Extended example execution of the TRAP protocol.	206
D.1.1	Deceitful ratio for a number of branches a in a blockchain fork.	211
D.2.2	Throughput and latency of ZLB.	212
D.3.3	Disagreements per Hamming distance of the disagreeing bitmasks.	213

List of Tables

3.1	Attack results for different balance distributions.	35
5.1	Complexities of naive implementations of Basilic protocols.	108
5.2	Complexities of Basilic compared to other works.	111
5.3	Comparative table of ZLB with previous work.	126
6.1	Complexities of naive RBV-broadcast, Pinakion and Kleroterion.	153
6.2	Normalized and amortized per route complexities of Kleroterion and SPURT. .	156
6.3	Comparison of distributed random beacons.	158
A.2.1	Parameters that help infer the Lightning Network implementation of a node. .	198
A.2.2	Number of nodes per implementation of the Lightning Network.	198
A.3.3	Default parameters for different implementations.	203
C.2.1	Complexities of Basilic protocols before GST.	210

Chapter 1

Introduction

Since Nakamoto’s 2008 Bitcoin proposal [1], the field of blockchains has grown steadily in both financial and research interests. The popularity of blockchains and cryptocurrencies, and their interest as an investment option, has however led to an abuse of the term and disregard of its foundations in the design of several works in the blockchain ecosystem. At the time of writing, blockchains are the central dedication of several experts from various fields, such as distributed systems, graph theory, networks, game theory, economy or cryptography, but also business, law, policy-making, or even journalism and sociology. However, unlike other technologies, such as the Internet or the first computers, that originated from multiple funded research efforts, blockchains arose in a community forum of technological enthusiasts that decided to experiment with their ideas. These unusual beginnings have led to many works going to market without a clear understanding of the implications of these proposals, in terms of security, usability or originality. Examples of this are the multiple blockchains that are in fact almost identical forks of Bitcoin [2, 3, 4, 5], some of which are driven by memes (the so-called memecoins) [6, 7], many of them holding market caps well over US\$1 billion at the time of writing. In fact, as of November 2021, the number of cryptocurrencies set at 70,000, with about 100 new ones created every day [8].

Most blockchains do, however, share a common backbone: a distributed append-only ledger that orders blocks of transactions. Although all blockchains are relatively young, their fundamentals can be traced back to the second half of the 20th century. For example, the aforementioned definition of a blockchain fits as a particular case of a State Machine Replication (SMR) [9, 10], whereby a set of processes execute the SMR protocol to output ordered commands that are proposed as inputs. Blockchains also share an immediate connection with the problem of consensus [11, 12, 13], in which a set of processes execute a consensus protocol in order to agree on the same output, given multiple proposed inputs.

Unlike SMRs, which are general solutions intended for both privately related entities (e.g. data centers of the same company) and sets of independent entities (e.g. voting systems), blockchains are intended to be executed by a large number of independent processes, each with their own interests and incentives. This characteristic of blockchains asks for an analysis of them from different fields, like distributed systems, cryptography, game theory, or economy.

The most significant conjecture specific to blockchains is the so-called *blockchain trilemma* [14].

The trilemma states an impossibility of designing a blockchain guaranteeing the following three properties:

- **Security**, in that blockchains should provide their service in the presence of system and network failures,
- **Scalability**, in that users should not be affected by an increase of demand of the service, and
- **Decentralization**, in that all users of the service can also offer it.

This unproven, although intuitive, conjecture of blockchains exemplifies the broad range of research directions existing in the blockchain ecosystem. Security thus considers the properties that are ensured in the presence of network and system failures, scalability focuses on the performance of the blockchain and its capacity to provide its service to millions of users, and decentralization ensures that the service does not belong to and is not controlled by any subset of these users.

Common security considerations involve the network model in the presence of an adversary that can either cause crashes (crash faults) or arbitrary faults (Byzantine faults). Dwork et al. [13] already showed that it is only possible to solve consensus if less than half of the processes crash. This result applies even assuming authenticated messages and synchronous communication, in which all sent messages are delivered within a known bounded time Δ . Prior, Lamport et al. [12] had already proven that it is impossible to solve consensus even in the presence of just one crash fault in asynchrony, i.e. without a bound on the delivery of sent messages. Dwork et al. [13] also proved that it is impossible to solve consensus if there are $n/3$ Byzantine faults in a partially synchronous network, where n is the number of processes. Partial synchrony defines an unknown bound on the delay of messages. These results for consensus are relevant for blockchains because it has recently been shown that a secure blockchain has consensus number $+\infty$ [15], meaning that it must solve consensus, and thus the impossibility results that apply to consensus also apply to blockchains.

First generation blockchains were designed with Proof-of-Work (PoW) [1, 16, 6, 4, 2, 3]. PoW enables decentralization, as any user has a chance of being the first to solve a cryptographic puzzle, and thus offering the service. However, Bitcoin and other PoW-based blockchains are known to be either not scalable, deciding only up to 7 transactions per second (tps), or not secure if the scalability is increased by reducing the difficulty of the puzzle. In addition to this, the Bitcoin scalability problem ¹ cites that the requirement of all users to store all transactions of the blockchain poses a risk of centralization of these blockchains, as many users would not be able to implement full nodes of the network.

For this reason, newer generation blockchains tend to build upon the results on consensus protocols from the state of the art [17, 18, 19, 20, 21, 22, 23, 24, 25, 26]. These blockchains normally have a dedicated set of processes, a committee, that executes a consensus protocol in order to agree on the current block. The performance of these solutions is significantly better

¹https://en.wikipedia.org/wiki/Bitcoin_scalability_problem

than that of Bitcoin, with results as high as 60,000tps [27, 28]. These results are achieved even if they are limited by the need to execute consensus, which is known to require to exchange at least $\mathcal{O}(n^2)$ bits per decision [29]. While these proposals are normally safe and scalable, the challenge comes from the decentralization side. Decentralization is normally obtained by executing a committee rotation protocol that rotates the committee such that all users can be selected. However, committee rotation protocols are believed to offer decentralization at a trade-off with security and scalability [14].

A committee rotation protocol can either rely on (i) a deterministic rotation [22, 21] (by means of iteratively hashing a seed, for example), (ii) a rotation based on an election [18, 30], or (iii) randomness [31, 32, 33, 17], or a mix of these. Deterministic committee sortition protocols are vulnerable to an adversary as the adversary can predict the output (and use it to its own advantage). Protocols based on an election are vulnerable to an adversary iteratively strictly increasing the proportion of the committee it controls, a direct consequence of fairness being impossible in the presence of just 2 Byzantine faults [34]. The third approach based on randomness relies on the implementation of a distributed random beacon protocol that periodically generates random outputs [35, 36, 37, 38, 39, 24, 40]. Random beacons have been proven at least as hard to solve as consensus [41], with some resilient-optimal solutions ensuring its correctness as long as the committee contains at most $n/3 - 1$ Byzantine faults in partial synchrony, some of these solutions also being optimal in the communication complexity lower bound of consensus [29]. The drawback of these protocols is that there is however a probability of one iteration of this protocol selecting a committee containing at least $n/3$ Byzantine faults, after which all committees are selected by the adversary.

In this dissertation, we focus on blockchains based on consensus and random beacons in partial synchrony. The works that we present here represent meaningful improvements on security and decentralization while offering competitive performance and complexity measures compared with the state of the art. The combined solutions of this dissertation converge towards a novel blockchain that provides an unprecedented level of security and decentralization, tolerating even an adversary controlling up to less than $2n/3$ processes of a committee, or even 50% of all users with very high probability. We prove the optimal resilience of this blockchain in the classical model, and take pragmatic approaches to prove its security in realistic, newer models, and in the presence of rational players with realistic utilities for blockchain applications. We also prove the optimal resilience of our protocols in these new models besides the classical tolerance to Byzantine faults.

We hope that the models, problems, proofs, conjectures, protocols and implementations of this dissertation bring more works for blockchains that are disruptive in nature, but that inherit from the advances of previous works in their respective fields of research.

1.1 Objectives

The main objective of this thesis is to contribute to the design of safe, scalable and decentralized blockchains. In doing so, we identify a number of objectives, that we list here below.

Objective 1: formally define blockchains and its properties

Since blockchains arose from an Internet community, sometimes disconnected from previous research, their first definitions were self-contained without referencing structures, properties and systems from many works that preceded blockchains. The company-driven ecosystem of blockchains has also prioritized the development of systems on the basis of conjectures and visions at the cost of formality and lack of proven properties and claims.

One of the main objectives of this dissertation is thus to set an accurate formal model of blockchains and their extensions (such as layer-2, sidechains or crosschain payments), and abstracting properties that gather the views and goals of blockchains encompassing the heterogeneity of proposals in the blockchain community.

As a result, we list the three main properties for all public blockchains that constitute the so-called blockchain trilemma [14], serving as the backbone of this dissertation, as sub-objectives. Hence, we list as sub-objective to contribute towards the 1.1 security, 1.2 scalability, and 1.3 decentralization of blockchains through our research.

In addition, we will identify and formally define other problems and properties that directly concern blockchains throughout this dissertation.

Objective 2: state and formally prove impossibility bounds and trade-offs, and propose sensible metrics for comparison

A natural step after having formally stated a model and properties for a blockchain system is to delimit the bounds under which a model can solve the desired problem. Formally proving impossibility bounds that apply to all blockchains that want to satisfy a specific set of properties is thus a central objective of this dissertation.

In fact, some of these properties compose trade-offs that require blockchains to make compromises. We also encompass in this objective to propose metrics that help parameterize where each of these blockchains lies in these trade-offs, in an effort to properly analyze the blockchain ecosystem and help the reader understand the advantages and drawbacks of design decisions.

Objective 3: design and prove solutions with competitive metrics and bounds

Having formalized the blockchain problem and properties, and after narrowing it down to impossibility proofs and trade-offs, the main objective is that of designing protocols that are optimal, making sensible design decisions that are properly justified. This is done by using the metrics previously proposed for the known trade-offs to show that these solutions improve some of the metrics and are competitive in all other metrics, compared to previous works.

Objective 4: implement and test proposed solutions in a real environment

The last objective of this dissertation is to implement the proposed solutions with the goal of testing them in a real world environment, obtaining final validation from real tests. These tests must include comparisons of sensible metrics for the desired properties with previous works, which must all be tested in the same settings.

1.2 Contributions

We present a myriad of contributions that address all of our objectives. The narrative of these contributions is divided in the main property of the blockchain trilemma that each of these contributions address. First, we concentrate on security to present a novel attack that reinforces previous research pointing out the dangers of assuming a synchronous communication network. Then, we take this justification to present a scalable protocol for blockchains' layer-2 networks that does not assume synchrony. We continue with security and successfully address its trade-off with scalability by proposing a class of protocols that tolerates the strongest adversary to date, at an optimal communication complexity. Finally, we aim at decentralization by extending our previous results and proposals for secure scalable blockchains. Our goal is to ensure decentralization at a sensible compromise between the trade-offs and impossibilities of the blockchain trilemma. The outcome of our contributions converge thus towards a blockchain that allows for a level of configuration to readjust the trade-offs between these three properties.

To summarize, this dissertation presents the following contributions:

1. **The Lockdown attack.** (Objectives 1, 1.1, 3 and 4.)

We extend the list of known problems of synchronous blockchain protocols by introducing a novel attack against synchronous off-chain state channel networks that promote privacy. The attack depicts a scenario whereby an adversary can lock the total funds directly related to any node of the network with only 10% of them.

2. **The Platypus offchain protocol.** (Objectives 1, 1.2, 2, and 3.)

Following the Lockdown attack we present Platypus, an offchain protocol for blockchains without synchrony. Platypus is a childchain, an offchain protocol that is also a blockchain. We formalize the offchain, childchain, and sidechains problems, present the Platypus protocol and prove its correctness, as well as an impossibility proof showing Platypus' optimal resilience.

3. **The TRAP rational agreement protocol.** (Objectives 1, 1.1, 2, and 3.)

The tolerance to faults and system failures of blockchains emerges from its consensus protocol. Classical results show that it is impossible to solve consensus if more than a third of participants are Byzantine. Nevertheless, we propose a game theoretical model for blockchains in order to present the TRAP protocol for consensus, a protocol that solves consensus in the presence of an adversary controlling less than a third of Byzantine faults and also up to less than half of rational participants interested in causing a disagreement. We also prove that it is impossible to solve this problem without implement a baiting strategy, a novel definition of a type of punishment strategy that requires collaboration from within the coalition in order to succeed.

4. **The Basilic class of consensus protocols.** (Objectives 1, 1.1, 2, 3.)

While the TRAP protocol provides a significant improvement for consensus in blockchain applications, it requires a proper modeling of the incentives motivating rational players

to cause a disagreement. For this reason, we present the novel Byzantine-deceitful-benign failure model, in which deceitful faults will always be interested in causing a disagreement, while benign faults can always prevent termination. Then, we present an impossibility result that extends the traditional bounds of Byzantine fault tolerance to the BDB model. Finally, we present the Basilic class of protocols, a resilient-optimal class of protocols for the problem of consensus in both the BDB model and the classical model of tolerance to Byzantine faults. We also prove the Basilic class protocols to have optimal communication complexity.

5. **The Zero-loss Blockchain (ZLB).** (Objectives 1, 1.1, 2, 3 and 4.)

With the previous advances from presenting Basilic, we then present Zero-loss Blockchain (ZLB), a blockchain that tolerates a majority of the system trying to cause a disagreement. ZLB tolerates transient disagreements in partial synchrony by eventually resolving them and punishing provably detected fraudsters responsible for the disagreement. We also prove ZLB to ensure that only faulty users suffer the consequences of their attacks in probabilistic synchrony.

6. **The Kleroterion and Kleroterion⁺ random beacon protocols.** (Objectives 1, 1.1, 2 and 3.)

While the previous results of TRAP, Basilic and ZLB improve security, translating these results to public, permissionless blockchains without assuming probabilistic synchrony requires the use of a random beacon protocol in order to randomly rotate the committee of the blockchain. We first formally justify the need for a random beacon protocol for committee sortition to then present Kleroterion, a resilient-optimal random beacon protocol that exchanges a number of bits per network channel independent of the size of the participants in the protocol without requiring a trusted setup. Then, we extend Kleroterion to tolerate colluding majorities thanks to the advances from ZLB by proposing Kleroterion⁺. Kleroterion⁺ concludes the blockchain trilemma of our proposed blockchain, which contains Platypus for scalability, with Basilic, TRAP and ZLB for security, and Kleroterion or Kleroterion⁺ for decentralization.

1.3 Outline

The remainder of this dissertation is organized as follows:

- Chapter 2 presents the background of previous works and introduces preliminary concepts, formal definitions and assumptions that will be used throughout the dissertation.
- Chapter 3 begins by presenting the Lockdown attack for synchronous offchain channel networks, as well as the problem of synchronous offchain protocols. Platypus is then presented, an offchain protocol that does not require synchrony.
- Chapter 4 presents the rational agreement problem and TRAP, a protocol to solve consensus in the presence of an adversary controlling less than a third of Byzantine faults and

also less than half of rational participants interested in causing a disagreement.

- Chapter 5 extends TRAP by defining the BDB model, and introduces the Basilic class of resilient-optimal protocols under this model. ZLB is then presented, a blockchain to tolerate colluding majorities by tolerating partial disagreements.
- Chapter 6 consolidates the results of this dissertation by justifying the need for a random beacon protocol for committee sortition. Kleroterion is then presented, a random beacon protocol that provides the best complexity metrics to date with the standard tolerance to faults for bias-resistance and unpredictability. Kleroterion exchanges a number of bits per network channel independent of the size of the participants in the protocol, except for one message of size n sent by the leader of the epoch and for the reconstruction phase, and without requiring a trusted setup. We then present Kleroterion⁺, a random beacon protocol that tolerates colluding majorities.
- Chapter 7 concludes the dissertation and discusses future work.

Chapter 2

Background & Preliminaries

In this chapter, we first detail the related work of the blockchain ecosystem, to then formally state definitions and assumptions that we use throughout this dissertation.

2.1 Background

Bitcoin [1] set the stage for an emergence of various blockchains, each with unique designs tailored for different use cases. Bellow, the core concepts of these blockchains are detailed, exploring their significance in system and protocol design, and how they fit into the related work of this dissertation.

2.1.1 Consensus

The problem of a set of n processes (a committee) solving consensus, i.e. satisfying agreement, termination and validity, has been central to distributed systems since the 1980s [11]. Typically, consensus protocols are proven to tolerate at most a fraction t of faults, these being either arbitrary (Byzantine) or crash faults. Consensus lies at the core of a myriad of blockchains [42, 43, 26, 17, 19, 18, 22, 20, 21, 25, 44] and, in fact, it has been proven necessary [45] in order to provide certain features and properties of blockchains, such as irrevocably confirming blocks [15]. These blockchains are thus designed taking into account the assumptions and bounds of their consensus protocols. Dwork et al. [13] showed that it is only possible to solve consensus if less than half of the processes crash. This result applies even assuming authenticated messages and synchronous communication, in which all sent messages are delivered within a known bounded time Δ . Prior, Lamport et al. [12] had already proven that it is impossible to solve consensus even in the presence of just one crash fault in asynchrony, i.e. without a bound on the delivery of sent messages.

While an asynchronous protocol can not provide any guarantee in terms of tolerance to faults, a synchronous one cannot provide tolerance to events that delay communication beyond Δ . A popular assumption that lies between synchrony and asynchrony is *partial synchrony* [13], in which sent messages are delivered to their recipients within bounded, but unknown time. The popularity of this model arises due to it closely reflecting the behavior of the Internet,

where communications often perform as expected, but occasionally delay significantly. In this model, Dwork et al. [13] proved that it is impossible to solve consensus if there are at least $n/3$ Byzantine faults, or instead at least $n/2$ crash faults.

2.1.1.1 Accountable consensus

Consensus protocols can be used for a broad range of applications, each facing different challenges. In order to maintain consistency and availability, blockchains must account for a potentially greater amount of safety attacks than normally ensured by protocols designed to execute consensus across data centers controlled by the same company [46]. In the case of blockchains, The incentives intrinsic to their applications and decentralized nature have led to several works studying other properties that can be guaranteed in the presence of stronger adversaries than those defined by the aforementioned classical impossibility results. Recent works [47, 48] define the property of accountability, stating that if honest processes fail to reach agreement, they can then identify the faults responsible. Unfortunately, no previous work specifies how to resolve the disagreement to satisfy agreement again and punish faulty processes.

The field of multi-party computation presents work that addresses accountability. Covert security ensures that honest processes detect misbehavior with a minimum probability, in an effort to deter an adversary from deviating (covert adversary) [49, 50, 51, 52]. Covert security with public verifiability guarantees that honest processes can prove this misbehavior to other honest processes [53, 54, 55]. This can be seen as a probabilistic case of accountability, since faults are detected only with some probability. Unfortunately, none of these works in covert security guarantee output delivery without synchrony in the presence of Byzantine faults.

2.1.1.2 Rationality for consensus

Besides other guarantees and properties in the presence of Byzantine adversaries, some works assume an heterogeneity of processes, modeling particular applications in order to tolerate a stronger adversary. The first natural step is to consider the rationality that motivates participants. These works describe a protocol as a recommended strategy for all players within a game, and try to design such protocol so as to make it an equilibrium in the game of all possible deviations from the protocol.

Some works focused on the conditions under which termination and validity is obtained for a non-negligible cost of communication and/or local computation [56, 57]. Several research results focus more particularly on agreement, with some deriving from the BAR (Byzantine-Altruistic-Rational) model. However, these works considered either no Byzantine players [58, 59], no coalitions of rational players [60], synchrony [58, 61, 62, 63, 64] or solution preference [61]. By assuming a larger payoff for agreeing than for disagreeing, solution preference requires that rational players never have an incentive to sabotage agreement.

Some results consider the problem of consensus in the presence of rational players but do not consider failures [58]. Leader election [65], which can be used to solve consensus indirectly, and consensus proposals [34] focus on ensuring fairness defined as all players having an equal probability of their proposal being decided.

Some proposals study consensus and mix faulty players with rational players [66, 67]. However, they consider the synchronous communication model. The idea of mixing rational players with faulty players has already been extensively explored in the context of secret sharing and multi-party computation [68, 69, 70, 71]. In particular, the central third-party mediator that is typically relied upon was implemented with synchronous *cheap talks* [71], that are communications of negligible cost through private pairwise channels. This extension was indeed illustrated with a secret sharing protocol for $n > k + 2t$, where k is the number of rational players and t the number of Byzantine processes. It was later shown [72] that mediators could be implemented with asynchronous cheap talks if $n > 3(k + t)$.

2.1.1.3 Other faults

Considering rational players requires defining their utilities, and thus some properties may not be guaranteed if these utility functions do not correspond with those of the real players that these rational models represent. As a result, some works introduce new faults that, contrary to rational players, can always deviate, but whose set of possible deviations is restricted compared to Byzantine faults. An example is the crash-fault process, restricted to only deviate into crashing at any time, but not, for example, into equivocating.

Flexible BFT [73] offers a failure model and theoretical results to tolerate $\lceil 2n/3 \rceil - 1$ alive-but-corrupt (abc) processes. An abc process behaves maliciously only if it knows it can violate safety, but behaves correctly otherwise. Additionally, their fault tolerance requires a commitment from users to not tolerate a single Byzantine fault in order to tolerate $\lceil 2n/3 \rceil - 1$ abc faults, or to instead tolerate no abc faults if users decide to tolerate $t = \lceil n/3 \rceil - 1$ Byzantine faults. Neu et al.'s ebb-and-flow system [74] is available in partial synchrony for $t < n/3$ and satisfies finality in synchrony for $t < n/2$, being t the number of Byzantine faults. They also motivate the design of new faults to tolerate equivocations in their recent availability-accountability dilemma [75]. The recent YOSO (You Only Speak Once) [76] also follows this line of thought for permissionless systems by envisioning protocols with committee sortition in which each processes only communicates one with the rest.

Upright [77] tolerates $n = 2u + r + 1$ faults, where u and r are the numbers of commission and omission faults, respectively. Upright tolerates $n/3$ commission faults or $n/2$ omission faults. Anceaume et al. [78] tolerate $t < n/2$ Byzantine faults for the problem of eventual consensus (i.e. tolerating transient disagreements), at the cost of not tolerating even $t = 1$ Byzantine fault for deterministic consensus.

2.1.2 Blockchain trilemma

The main challenges and desirable features of blockchains have widely been simplified into a three-way trade-off, known as the *blockchain trilemma* [14]. This trilemma establishes a trade-off between three properties: security, scalability, and decentralization. First, security refers to the properties that a blockchain guarantees in the presence of attacks against the underlying network, by members of the system, or a mix of both. Second, scalability defines the capacity of a blockchain to produce high throughput, measured in decided values per second

(usually transactions per second, or tps). Third, decentralization ensures that every user of the blockchain service can also offer it. We formally define these properties in Section 2.2.

2.1.3 Scalability

The throughput of the Bitcoin mainnet has been measured at 7tps on average¹. This poor performance is in stark contrast to those of direct competitors that are centralized, with Visa performing at around 7000 tps in any given year². This scalability problem is no stranger to other blockchains, mostly affected by the need to replicate state across hundreds or thousands of processes before proceeding on to the next block to finalize. Several solutions aim at increasing scalability, one of the biggest challenges of blockchains. In fact, most blockchains consume more resources without offering better performance as the number of participants increases.

2.1.3.1 Layer-2

Although some research results demonstrated that blockchain performance can scale with the number of participants [79, 31, 27, 80, 81, 23, 28], these rare solutions do not have other appealing properties, like privacy, built in. As a result, blockchain extensions that offer scalability and privacy have been put forward, in what is known as *layer-2*, or *offchain protocols*. These protocols are characterized by allowing certain operations to take place outside of the blockchain, whereas these operations would normally be written on the blockchain. Examples of these protocols are *state and payment channels* [82] in which two parties can perform several offchain payments with one another; *channel networks* [82] that allow users to relay payments in a network of channels; *channel factories* [83] that open multiple channels in one transaction, saving storage and fees; and *childchains* [84] which are secondary blockchains pegged to the existing, so called “parent”, blockchain.

Offchain payment networks. The *Lightning Network* [82], a network of inter-connected 2-party channels that can route state changes, known as payment or state channels, is the most renowned layer-2 system. Unfortunately, the Lightning Network relies on synchrony, similarly to other networks and channels [85, 86, 87]. This makes these systems vulnerable to a number of attacks such as *channel exhaustion* and *payment grieving* attacks [88, 89].

Privacy issues are also reported in recent literature of payment channels. Tang et al. [90] address the impact of using payment channels w.r.t. privacy preservation. Since users need to route their transactions using other nodes (users in the payment network), they must find paths through the payment network, and with enough pre-allocated funds to route their transactions. This poses the problem of hiding the balance of each payment channel node. Our previous work [91] shows the difficulty of hiding such balances, by uncovering a balance discovery attack that can be used to deanonymize the precise balance of each network payment node, hence leading to the de-anonymization of network transactions. Tang et al. and Malavolta et al. [90, 92] assume that the adversary is passive, i.e., the adversary observes only the public information

¹https://en.wikipedia.org/wiki/Bitcoin_scalability_problem

²<https://usa.visa.com/dam/VCOM/global/about-visa/documents/aboutvisafactsheet.pdf>

released in the network, whereas previous works by Malavolta et al. and Ross et al. [93, 94] consider active adversaries acting as corrupt relay nodes, trying to learn the destination of other nodes transactions. The congestion attack [95] overloads payment routes during their expiration time, which can mean up to 14 days.

Sidechains & childchains. A childchain is a blockchain whose creation and termination is defined and performed in another blockchain (also known as the parentchain). Childchains were introduced with the concept of sidechains [96]. A sidechain targets crosschain payments not necessarily in the parent-child hierarchy. Childchains were first formalized for an efficient childchain protocol in a semi-synchronous model [84, 97]. Unfortunately, their notion of semi-synchronous communication considers that every message gets delivered in a non-null bounded amount of time Δ , which remains a synchrony assumption [13]. The term ‘semi’ is used by the authors to denote the fact that the bound Δ is not null. This notion differs from partial synchrony [13] where the bound is unknown.

2.1.3.2 Mainnet improvements

Sharding [98], hybrid consensus [99] and consensus on superblocks [23, 28] also aim at scaling blockchains, with some partially synchronous proposals [100, 101]. Compared to sidechains and childchains, however, these constructions hardly consider privacy requirements and crosschain payments.

Set Byzantine consensus. Previous research works propose new blockchain and consensus protocols that are specifically designed to tackle the scalability problem. The set Byzantine consensus [102, 103, 23, 28] (SBC) problem was designed with applications like this in mind. The SBC problem changes the property of validity to decide on an union of proposals, instead of on just one proposal. This means that blockchains that solve SBC can decide a so-called *superblock* per iteration of consensus that merges multiple proposals from different processes at once, resulting in an even greater throughput than that of Visa.

Crosschain payments. Many protocols propose generic crosschain payments. Atomic cross-chain swaps [104, 105, 106, 107] typically rely on synchronous Hashed Timelock Contracts [108], while others focus on a crash model, rather than a Byzantine one [107]. Polkadot [43] reuses the idea to manufacture a common parentchain, to perform payments asynchronously. Crosschain deals [109] allow for auctions or relaying payments, with both a synchronous and a partially synchronous protocol. The crosschain deals problem tolerates that the protocol aborts even if the only processes proposing abort are Byzantine.

2.1.4 Security

The security of blockchains covers a broad range of attacks ranging from attacks on the communication network [110, 111, 112, 113], to censoring the service for specific users [114], or even writing the entire blockchain from scratch [16].

Explaining all attacks is out of the scope of this dissertation. Instead, we focus on those attacks that are intrinsic to the blockchain problem abstraction. Following the CAP (consistency,

availability, partition-tolerance) theorem [115], we show thus attacks that affect consistency (agreement) and/or availability (termination).

Attacks on agreement try to split the state of the blockchain into two or more states, and have two or more disjoint subsets of the processes and/or users disagree on the current state. This attack is commonly known as a *double-spending*³, and the split of the blockchain into multiple blockchain states is referred to as a *fork*.

This attack incarnates several forms and names depending on the particular way to cause a disagreement (e.g. withholding blocks [116, 117, 118, 119], bribing processes [120], sending conflicting transactions [121]), and in the type of blockchain wherein the attack is performed (e.g. Proof-of-Work [122], Proof-of-Stake [123], Proof-of-Authority [124]). Some of these protocols can reinforce agreement by requiring many (or even all) processes to confirm a particular state. However, if processes wait for replies from other processes that have crashed, then the system risks not being available [74, 115]. In fact, we have already mentioned the impossibility bounds to satisfy both termination and agreement for consensus [13, 12], leading to the same bounds in blockchains that execute consensus [23, 28, 125, 26, 25, 24, 19, 126], or in any blockchain that satisfies strong prefix (i.e. irrevocable decisions) [15].

In an effort to discourage misbehavior, some blockchains ask for a deposit, also known as *stake*, from each process. This stake is *slashed* from a process if honest processes can prove its misbehavior. Although Buchman et al. [127] aimed at slashing processes without accountability, the authors have recently incorporated accountability in [128]. Balance [129] adjusts the size of the deposit to avoid over collateralizing but we are not aware of any system that implements it. SUNDR [130] assumes honest users that communicate directly to detect Byzantine faults. Polygraph [47] solves accountable consensus without slashing. FairLedger [131] assumes synchrony in order to detect faulty processes. Sheng et al. [132] consider forensics support as the ability to make processes accountable for their actions to users.

Freitas de Souza et al. [133] reconfigure processes in a lattice agreement after detection. The Casper [134] algorithm incurs a penalty in case of double votes, while Shamis et al. [135] propose to store signed messages in a dedicated ledger so as to punish processes in case of misbehavior. However, both require less than $n/3$ faulty processes to terminate.

2.1.5 Decentralization

The third side of the blockchain trilemma is decentralization. Again, we focus here on decentralization in consensus-based blockchains. In most of these blockchains the committee is intended to be a subset of the total number of participants in the system [42, 43, 26, 17, 19, 18, 22, 20, 21, 25, 44]. This allows for a constant performance independent of the number of participants in the system. Some of these blockchains keep a static committee [22, 28], in what is referred to

³In reality, a double-spending attacks involves the attackers misleading goods or service providers by spending their funds multiple times (once per decided value in the disagreement) in exchange of goods or services immediately after causing a disagreement. Once the fork is resolved/detected, the attackers have already gotten their goods services in exchange and the victims of the attack cannot use the funds they were paid for (except one of them). We however abuse notation by using interchangeably the term disagreement attack and double-spending attack.

as consortium blockchains. However, these solutions are not entirely decentralized: the subset of processes in the committee is in control of the blockchain, and they are, in this sense, no different from a system maintained by a handful of interconnected businesses.

As a result, in order to keep decentralization in these systems, several of these blockchains implement *committee sortition* protocols to rotate the members of the committee. These committee sortition protocols are periodically executed in order to replace the processes in the committee, in an effort to prevent any subset of blockchain users from controlling the blockchain except for a limited period of time.

There are three major types of distributed committee rotation protocols, which we refer to as elected, random and deterministic.

2.1.5.1 Elected committee

The elected committee rotation consists in having some voting protocol in a committee (or multiple committees) democratically elect the next committee [18, 30]. These approaches require assuming *fairness*, in that all of the proposals from all processes have the same probability of being decided, in order to guarantee that the adversary does not eventually control the rotation. Halpern et al. [34] proved that fairness is impossible even in synchrony and in the presence of only one rational player and one Byzantine player, and therefore the same result applies even in the presence of just two Byzantine players. The implication of not ensuring fairness translates in the adversary iteratively increasing its representation of the committee (as its input is more likely to be selected as output than its relative power in the committee), until it controls the entire committee.

2.1.5.2 Deterministic committee

A purely deterministic approach to committee sortition [22, 21] (by means of hashing a seed, or leaving a static committee, for example) produces unbiased random rotations, but they are predictable. This means that the adversary will know exactly in which consensus iterations they will have control of the committee, and for how many rotations, which they can use to their advantage to perform attacks (e.g. double-spending attack).

2.1.5.3 Random committee

Finally, random committee rotation approaches take inputs from processes as the source of randomness [31, 32, 33, 17], implementing *random beacon* protocols. Compared to an elected committee, this type of rotation is not vulnerable to the over-representation of a small adversary. Nevertheless, it has the same vulnerability for a big enough adversary: since the outputs of these protocols typically depend on the input from $\lceil n/3 \rceil$ processes, an adversary controlling just $1/3$ of the committee in one iteration can take control of the source of randomness from that iteration on.

Syta et al. [36] propose a random beacon tolerating less than $n/3$ Byzantine faults in asynchrony relying on a setup based on distributed key generation (DKG). Contrary to random

beacons with a setup based on common reference string (CRS), those relying on DKG require executing the setup phase after every rotation. Many protocols implement a mix of purely deterministic and random committee sortitions, where every R iterations there is a degree of randomness, and the committees in all the R following iterations are deterministically selected from the same random output, where R is typically large (e.g. $R = 10^3$ [26]). Regardless of this determinism within the R consecutive iterations, even the random output can be predicted, biased, or both. For example, random outputs based on VRFs to select valid proposers, such as those of Algorand [26], Polkadot [19, 20], Ouroboros Praos [25] or Elrond [136], are subject to an adversary selecting the random output that best fits their criteria among the available ones. That is, these beacons are vulnerable to an adversary of size $n/3$, since they can influence the beacon output from among the valid proposers, without being held accountable for it [137]. Algorand [26] relies on a weak synchrony assumption for safety, and a strong synchrony assumption for liveness. While strong synchrony refers to the classical synchronous definition where there is a known bound for the communication delay, weak synchrony states the need for synchronous periods of length s_h (e.g. hours) for every non-synchronous periods of length s_d (e.g. a day). Aleph et al. [138] use DKG in order to implement an asynchronous randomness beacon, removing the requirement of a trusted dealer setup present in the HoneyBadger's common coin [139]. Gao et al. [140] propose a common coin with $\mathcal{O}(\lambda n^3)$ bits and constant asynchronous rounds in order to solve asynchronous Byzantine agreement. RandSolomon [141] propose a random beacon tolerating less than $n/3$ Byzantine faults with linear message complexity, but it uses a non-standard deterministic encryption scheme, which could leak information to an eavesdropper.

Synchronous random beacon protocols [142, 38, 39, 1, 24, 143, 40, 144, 145, 146, 147, 148, 41, 41] also range from a variety of primitives and setup assumptions, but all of them tolerate at most $n/2$ Byzantine faults. Protocols based on Proof-of-Delay rely on strong and new assumptions about verifiable time-lock puzzles or verifiable delay functions [149, 150, 144].

Some works implement random beacons, or varieties of the random beacon problem, under a myriad of denominations, such as randomness beacons [138], global coins [151, 152], common coins [139, 32, 140], random number generators [41, 141], or coin tossing protocols [148, 143].

2.1.5.4 Secret-sharing

Secret-sharing protocols allow processes to exchange information that is only revealed either once the adversary has committed to certain information, or after it is too late for the adversary to influence the output.

Secret-sharing as a source of randomness. These protocols have been proven useful in blockchains for the implementation of random beacon protocols. Das et al. [31] propose a publicly-verifiable secret sharing (PVSS) protocol based on a $(\lceil n/3 \rceil, n)$ -threshold secret sharing scheme such that any $n/3$ processes can reconstruct the secret, but no $n/3 - 1$ of them can, with a communication complexity of $\mathcal{O}(\lambda n^2)$ per beacon output. Unfortunately, it does not tolerate $n/3$ Byzantine faults.

Kokoris Kogias et al. [32] present a high-threshold asynchronous verifiable secret sharing

scheme with a dual threshold where the reconstruction threshold is some k for $\lceil n/3 \rceil < k \leq \lceil 2n/3 \rceil - 1$. This way, the secret can only be reconstructed if k processes participate in the reconstruction, while allowing honest processes that did not participate in the sharing phase to recover their share with the help of $\lceil n/3 \rceil$ other processes. Unfortunately, an adversary controlling $\lceil n/3 \rceil$ of the processes could recover k secret shares of other participants and reconstruct the secret. Alhaddad et al. [33] propose an asynchronous verifiable secret sharing (AVSS) protocol with optimal communication complexity in the same model. Tomescu et al. [153] propose a PVSS with share recovery solution with an optimistic constant number of cryptographic operations per process, while Trek et al. [154] offer a resilient-optimal asynchronous complete secret sharing protocol of quasi-linear computation and communication complexity. Boyle et al. [148] present a synchronous VSS protocol that tolerates less than $n/3$ Byzantine faults, while Schindler et al. [40] present a PVSS protocol under the same model and fault tolerance. However, none of these works tolerate an adversary controlling $\lceil n/3 \rceil$ processes.

Secret-sharing in other applications. The ability to share secrets can be useful in more aspects of the blockchain environment. For example, when considering rational players and partial synchrony, ensuring to rational players that their strategy will not be discovered by faulty processes, or by other rational players, provides assurance that the rational player will not be discouraged from playing these strategies, as is showed by Breidenbach et al.’s submarine commitments [155], in which players are rewarded for proving some knowledge, which players can hide in a seemingly normal transaction, but that later is proven to hide the desired information.

Additionally, other protocols based on zero-knowledge proofs [156] provide a similar property, although they explicitly reveal the existence of an information to prove, which gives an additional advantage to other players to also claim the same knowledge.

2.2 Preliminaries

We formally state here the assumptions and definitions that we will use throughout this dissertation.

A *protocol* is a set of instructions that, when followed by all participants under the assumptions listed by the protocol, implements a desired functionality \mathcal{F} (such as implementing a blockchain). A committee is the set $N = \{p_0, p_1, \dots, p_{n-1}\}$ of $|N| = n$ processes that execute the protocol. We abuse notation by referring to a protocol σ for functionality \mathcal{F} if σ implements \mathcal{F} . We speak of users to refer to participants that use the system, but that are not part of the committee. We denote an element x sampled uniformly at random from a finite set \mathcal{S} by $x \xleftarrow{\$} \mathcal{S}$. We denote vectors using bold face lowercase letters such as \mathbf{y} .

2.2.1 Fault model

Processes that always follow the protocol are *honest*. Typically, faulty processes are of one of the following types: *Byzantine* and *crash*. Byzantine processes can behave in any arbitrary way, whereas crash-fault processes follow the protocol, but they may crash at any point during an execution, even having sent a broadcast message only to a subset of its intended recipients.

Throughout this dissertation, we will reserve the symbol t to refer to the number of Byzantine faults, or of crash faults when we explicitly state it, and also $t_\ell = \lceil n/3 \rceil - 1$ to denote the optimal number of Byzantine faults tolerated by a protocol for consensus in partial synchrony [13]. As we will define more types of faults later in this dissertation, as well as processes following rational behavior, we use the term *faulty* to refer to all processes that are neither honest nor rational. We will use f to denote the number of processes controlled by the adversary in the committee.

2.2.2 Communication network

The partially synchronous model sits in between the synchronous and asynchronous models by assuming that all sent messages are delivered by its recipients within an unknown, but bounded time. This model not only reflects the behavior of the Internet, but also balances tolerance to faulty processes and to a faulty network, for a total of up to t_ℓ Byzantine faults tolerated for the consensus problem, or $\lceil n/2 \rceil - 1$ crash faults [13]. In contrast, assuming synchronous communications means tolerating $\lceil n/2 \rceil - 1$ Byzantine faults but being vulnerable to network delays, while asynchrony is insufficient to solve consensus, as shown in Section 2.1. For these reasons, we assume the partially synchronous model throughout this dissertation, except for the Lockdown attack, in which we showcase a new type of attack that can be performed on to the synchronous Lightning Network (see Chapter 3), further justifying the need for weaker communication assumptions. In particular, our partially synchronous assumption specifies a known bound Δ on the communication delay that will hold after an unknown Global Stabilization Time (GST) [13]. Processes communicate through private pairwise channels.

2.2.3 Authenticating messages

We make standard cryptographic assumptions [157, 158]. We assume a standard public-key infrastructure (PKI), common to all processes, that associates the identities of processes with their public-keys (one per process). We refer to λ as the security parameter, i.e., the number of bits of the keys. We formalize negligible functions measured in the security parameter λ , which are those functions that decrease asymptotically faster than the inverse of any polynomial. Formally, a function $\epsilon(\kappa)$ is negligible if for all $c > 0$ there exists a κ_0 such that $\epsilon(\kappa) < 1/\kappa^c$ for all $\kappa > \kappa_0$ [159]. Many of the claims and proofs of this dissertation require cryptography to sign messages, meaning that they hold except with $\epsilon(\lambda)$ negligible probability of the messages being decrypted without holding the decryption key.

2.2.4 Send, receive and deliver

Messages can be sent and received, but we also consider broadcast primitives that contain two functions: a broadcast function that allows process p_i to send messages through multiple channels across the network, and a deliver function that is invoked at the very end of the broadcast primitive to indicate that the recipient of the message has received and processed the message. There could be however multiple message exchanges before the delivery can happen. As we will specify some of these broadcast primitives, we attach the name of the protocol as

a prefix to the broadcast and deliver function to refer to a message broadcast or delivered using that protocol, such as AARB-broadcast, AARB-deliver, ABV-broadcast, ABV-deliver, RBV-broadcast and RBV-deliver, as we detail later in this dissertation.

2.2.5 Adversary

We model processes as probabilistic polynomial-time interactive Turing machines (ITMs) [160, 35, 161]. A process is an ITM defined by the following protocol: it is activated upon receiving an incoming message to carry out some computations, update its states, possibly generate some outgoing messages, and wait for the next activation. The adversary \mathcal{M} is a probabilistic ITM that runs in polynomial time (in the number of message bits generated by honest parties). \mathcal{M} can control the network to read or delay messages, but not to drop them. It can also take control and corrupt a coalition of processes, learning its entire state (stored messages, signatures, etc.). It takes control of receiving and sending all their messages. Furthermore, it can deliver the messages from honest processes and users instantly, and its messages can be delivered instantly by any honest process or user.

2.2.6 Solving consensus

We define in this section problems that are critical to this dissertation. The building block for this dissertation is the problem of a committee of processes reaching agreement on a value, i.e. the consensus problem [12]. We detail this problem in Definition 2.2.1.

Definition 2.2.1 (Consensus). A protocol solves the consensus problem if it satisfies the following three properties:

- **Termination.** Every honest process eventually decides on a value.
- **Agreement.** No two honest processes decide on different values.
- **Validity.** If all honest processes propose the same value, no other value can be decided.

Termination and agreement ensure that honest processes terminate the protocol and agree on the output, while validity excludes trivial solutions by protocols that ignore inputs.

2.2.6.1 Reliable broadcast

Some consensus protocols solve consensus by first executing an instance of another type of protocol known as reliable broadcast [162, 23, 163]. Suppose a process p_s , known as the *source*, that wants to broadcast a value v to all honest processes, such that either all or none of the processes deliver it. A protocol σ for reliable broadcast guarantees that if an honest process delivers a value v from p_s at the end of σ , then all other honest processes also delivered v and only v . We formalize this problem in Definition 2.2.2. We have detailed part of the terminology for reliable broadcasts protocol in Section 2.2.4.

Definition 2.2.2 (Reliable Broadcast). A protocol solves the reliable broadcast problem if it satisfies the following properties:

- **RB-Unicity.** Honest processes RB-deliver at most one value v from the source p_s .
- **RB-Validity.** Honest processes RB-deliver v if v was previously RB-broadcast by p_s .
- **RB-Send.** If p_s is honest and RB-broadcasts v , then honest processes RB-deliver v .
- **RB-Receive.** If an honest process RB-delivers v , then honest processes RB-deliver v .

2.2.6.2 Accountable consensus

Previous work introduced signatures in consensus protocol messages, guaranteeing that for a disagreement to occur, some processes must sign conflicting messages, and once these messages are discovered by an honest process, such process can prove the fraudsters to the rest of honest processes through *proofs-of-fraud* (PoFs) [47]. This is defined in the property of accountability, one of the properties of the accountable consensus problem shown in Definition 2.2.3.

Definition 2.2.3 (Accountable Consensus). A protocol σ solves the accountable consensus problem if σ solves consensus and it satisfies the following accountability property:

- **Accountability.** If two honest processes output disagreeing decision values, then all honest processes eventually identify the processes responsible for that disagreement.

The problem of accountability applies to a situation that should not be possible if the protocol σ solves consensus, since it must satisfy agreement. In general, though, accountability for consensus protocols applies in the presence of a stronger adversary. This means that the protocol would solve consensus in the presence of an adversary, typically controlling at most t_ℓ Byzantine faults, and that it satisfies accountability when agreement is not satisfied, i.e. if the adversary controls more than t_ℓ faults, typically $t_s = \lceil 2n/3 \rceil - 1$. As with t_ℓ , we will use the value $t_s = \lceil 2n/3 \rceil - 1$ throughout this dissertation. We however keep the problem definition abstract in this section, and will specify it later in this dissertation (Definition 5.3.1).

2.2.6.3 Set Byzantine consensus

Although the protocols present in this dissertation are abstracted and may be of independent interest, the purpose of our proposals are intended primarily for the blockchain environment. For this reason, when our solutions require to specify a particular consensus protocol, we often choose to build upon an implementation of a variant of consensus (Definition 2.2.4) useful for blockchains [23, 139, 164] where the validity property requires the decided value to be a subset of the union of the proposed values, hence allowing us to decide more proposals per iteration of consensus. Protocols that solve this variant can however easily solve consensus as defined in Definition 2.2.1.

Definition 2.2.4 (Set Byzantine Consensus). Assuming that each honest process proposes a set of values, the *set Byzantine consensus* (SBC) problem is for each of them to decide on a set in such a way that the following properties are satisfied:

- **SBC-Termination.** every honest process eventually decides a set of values;

- **SBC-Agreement:** no two honest processes decide on different sets of values;
- **SBC-Validity:** the decided set of values is a subset of the union of the proposed values;
- **SBC-Nontriviality:** if all processes are honest and propose the same value v , then the decided set is $\{v\}$.

SBC-Termination and SBC-Agreement are common to their analogous in Definition 2.2.1, while SBC-Validity states that the decided set must contain proposed values, and SBC-Nontriviality is necessary to prevent trivial algorithms that decide a pre-determined value from solving the problem.

2.2.6.4 Eventual consensus

The eventual consensus (\diamond -consensus) abstraction [165] captures eventual agreement among all participants. It exports, to every process p_i , operations `proposeEC1`, `proposeEC2`, ... that take multi-valued arguments (honest processes propose valid values) and return multi-valued responses. Assuming that, for all $j \in \mathbb{N}$, every process invokes `proposeECj` as soon as it returns a response to `proposeECj-1`, the abstraction guarantees that there exists $k \in \mathbb{N}$, such that the following properties are satisfied:

- **\diamond -Termination.** Every honest process eventually returns a response to `proposeECj` for all $j \in \mathbb{N}$.
- **\diamond -Integrity.** No process responds twice to `proposeECj` for all $j \in \mathbb{N}$.
- **\diamond -Validity.** Every value returned to `proposeECj` was previously proposed to `proposeECj` for all $j \in \mathbb{N}$.
- **\diamond -Agreement.** No two honest processes return different values to `proposeECj` for all $j \geq k$.

2.2.7 State machine replication

A State Machine Replication (SMR) [166, 167] is a replicated service that accepts deterministic commands submitted by nodes and totally orders these commands, using a consensus protocol so that, upon execution of these commands, every honest replica ends up with the same state despite *Byzantine* or malicious replicas.

More formally, we follow the terminology of França Rezende et al. [168] to assume that each user u_i holds a possibly unbounded log, which contains entries, ordered starting from the entry at position 0. We use $\ell_{u_i}[j]$ to refer to the log entry of u_i at position j , the operator $\ell_{u_i} \parallel c$ appends command c to u_i 's log. For a command at position j of u_i 's log to be executed by u_i , u_i must execute first all commands at positions $k < j$.

- **Validity.** A command is appended once and only if it was submitted before.
- **Stability.** If $\ell_{u_i}[j] = c$, $c \neq \perp$ holds at some point in time, it is also true at any later time.

- **Consistency.** For any two users u_i and u_j , if $\ell_{u_i}[k]$ and $\ell_{u_j}[k]$ are both not \perp , then they are equal.

Notice that according to this definition, compliant with that of strong prefix [15], Bitcoin does not deterministically satisfy the definition of SMR in partial synchrony. Also, notice that in this dissertation we will allow temporarily breaking these properties to explore tolerance to stronger adversaries, in what we will refer to as *accountable SMR* (ASMR).

2.2.7.1 Blockchain

Inspired by the model of Gazi et al. [97], we refer to a blockchain $\Omega = \langle b_i \rangle$ as an SMR maintaining a sequence of *blocks* that determines its current state. We denote $\Omega[i]$ as the i^{th} block of Ω , and i as the block's *blockheight*, $\Omega[-i]$ being the i^{th} latest block of Ω , with $-i$ its *blockdepth*. Decided values are added to a block b that is then written in Ω . For ease of exposition, we further specify blockchains by requiring the decided values of a block to be *transactions*, as most blockchains decide on sets of transactions. Each process *proposes* blocks containing transactions, which can later be decided and finalized. We speak of a block being *final* at blockheight i if it has been appended to the blockchain in the aforementioned SMR terminology, i.e. such that it will not be removed from that blockheight. We say that a block is *decided* when it has been appended to a blockheight i of the blockchain, but it is not yet final, in that it can suffer reorganization and be removed from that blockheight upon processes finalizing a different block at i .

Transactions. A transaction is a tuple $tx = \langle I, O \rangle$ where I is a list of inputs and O a list of outputs [169]. Outputs are stored in an Unspent Transaction Output (UTXO) pool until a transaction that consumes it as one of its inputs gets written in Ω . We model the outputs as $o_i = \langle s_i, \mathbb{C}_{o_i} \rangle$ where the set $s_i = \{(u_i, \text{conds}_{u_i})\}$ defines the conditions conds_{u_i} for the user u_i to spend the coin \mathbb{C}_{o_i} ($\mathbb{C}_{o_i} \geq 0$). In order to spend a coin, the associated conditions conds_{u_i} must be fulfilled so that only one user, among multiple candidate ones, can spend this coin.

Ownership. We say that user u_i owns coin \mathbb{C}_i if there exists a list of conditions conds_{u_i} such that u_i can spend \mathbb{C}_i . As such, let \mathbb{C} be the set of coins, \mathcal{T} the set of discrete timeslots (e.g. blockheight), and M the set of users, with $m = |M|$, then ownership is a function $\varphi : \mathbb{C} \times \mathcal{T} \rightarrow M$ that takes a coin and returns its owner at a particular time.

Accounts. We define an account $z \in \mathfrak{Z}$ as an instance of only one user u_i , $\gamma(z) = u_i$, where $\gamma(z)$ is a function that returns the user that controls account z . An account belongs to a particular blockchain, one account is controlled by only one user, but one user can have multiple accounts, either in the same or in different blockchains.

Transferring coins. A transaction may transfer one or more coins. We refer to u_i transferring a coin \mathbb{C}_i to u_j if u_i spends it to u_j . We can define a transfer of a coin as a change of ownership. That is, let $u_i, u_j \in M$, $z_i, z_j \in \mathfrak{Z}$ such that $\gamma(z_i) = u_i$, and $\gamma(z_j) = u_j$, and let $\mathbb{C}_i \in \mathbb{C}$, $\delta_i \in \mathcal{T}$, such that $\varphi(\mathbb{C}_i, \delta_i) = u_i$, then the transfer relation TR to u_j is such that $z_i TR_{\delta_{i+1}, \mathbb{C}_i} z_j \iff \varphi(\mathbb{C}_i, \delta_{i+1}) = u_j$.

Notice we can define the *transitive closure* of the transfer operation as follows:

$$TR^+ = \left\{ (z_i, z_j) \in \mathcal{Z}^2 : \exists \mathbf{c}_i \text{ s.t. } \varphi(\mathbf{c}_i, \delta_i) = u_i \text{ and } \varphi(\mathbf{c}_i, \delta_j) = u_j, \right. \\ \left. \text{for some } \delta_i, \delta_j \in \mathcal{T}, i < j \text{ where } \gamma(z_i) = u_i, \gamma(z_j) = u_j \right\} \quad (2.1)$$

2.2.8 Committee sortition

As we detailed in Section 2.1, several blockchains rotate the committee following the outputs of a random beacon protocol, a distributed protocol that periodically outputs a random number. The output Z of a random beacon is characterized by its randomness, in that Z is unpredictable and unbiased. We define formally the random beacon problem in Definition 2.2.5 [40, 170, 36, 171].

Definition 2.2.5 (Random beacon). A protocol σ solves the random beacon problem if it satisfies all of the following properties:

- **Agreement:** All honest processes agree on the same random output Z .
- **Availability:** Every honest process eventually outputs one value Z .
- **Verifiability:** If an honest process decides Z , then every honest process can verify it.
- **Unpredictability:** No process can predict the value of Z with probability greater than randomly guessing the secret before at least one honest process decides Z .
- **Bias-resistance:** No process can fix $c \geq 0$ bits of Z with probability greater than $\epsilon(c) + \epsilon(\lambda)$.

The first two properties, namely agreement and availability, are similar to the consensus problem, although typically referred to as agreement and termination. Verifiability states that all honest processes can verify the validity of the output, which extends the simpler property of validity in the consensus problem. In fact, the well-known impossibility results for consensus [13] are proven to also apply to the random beacon problem [41]. The properties of unpredictability and bias-resistance guarantee the randomness of the output with respect to all processes.

Analogously to how many consensus protocols actually start by executing one or more instances of a reliable broadcast protocol, many random beacons also start by executing n instances of a publicly verifiable secret-sharing (PVSS) protocol, or a similar variant. Similarly to how protocols for reliable broadcast allow a source process p_s to reliably broadcast a value, PVSS protocols allow a process known as the dealer p_d to share a *secret* value s , ensuring that s can only be revealed if enough processes are able to reveal it. We inspired from previous works to formalize protocols for PVSS [172, 173, 40, 31].

PVSS protocols are structured in four phases [31]:

1. **Setup:** The dealer p_d generates and publishes the parameters of the scheme. Every process p_i publishes a public key pk_i and withholds the corresponding secret key sk_i .
2. **Distribution:** The dealer creates one encrypted *secret share* $\mathbf{c}_d = \{c_{i,d}\}$ of the secret s with pk_i for each process p_i , along with a proof \mathbf{v}_d that these are indeed valid encrypted shares of some secret.

3. Verification: Each process (or an external verifier) verifies that the secret shares \mathbf{c}_d are indeed valid shares of some secret.
4. Reconstruction: In this phase, each process p_i decrypts their respective share $c_{i,d}$ with their secret key sk_i , obtaining their *decrypted share* $s_{i,d} = pk_i^{c_{i,d}}$, and shares $s_{i,d}$ along with a (non-interactive) zero-knowledge proof that $s_{i,d}$ is a correct decryption of $c_{i,d}$. Each process (or an external verifier) verifies the decrypted shares, and applies a reconstruction procedure to recover the original secret s_d shared by the dealer p_d .

We formalize the PVSS problem in Definition 2.2.6.

Definition 2.2.6 (Publicly verifiable secret-sharing). Let a dealer p_d share a secret s with $n - 1$ additional processes following a protocol σ . Then, σ solves the PVSS problem with security parameter λ if it satisfies the following properties:

- *Verifiability*: If the check in the verification step returns 1, i.e. succeeds, then with probability at least $1 - \epsilon(\lambda)$ the encryptions \mathbf{c} are valid shares of some secret. Furthermore, if the check in the Reconstruction phase passes then the communicated values \mathbf{c} are indeed the shares of a secret distributed by the dealer.
- *Correctness*: if p_d is honest, then with probability at least $1 - \epsilon(\lambda)$ the checks in the verification and reconstruction steps succeed, and honest processes can reconstruct s .
- *Secrecy*: If p_d is honest, then the probability of \mathcal{M} learning any information about p_d 's secret s prior to the reconstruction phase is at most $\epsilon(\lambda)$.
- *Agreement*: Honest processes do not reconstruct different secrets, even if p_d is faulty, with probability at least $1 - \epsilon(\lambda)$.

2.2.9 Blockchain trilemma

In this section, we state the blockchain trilemma. For this purpose, we first define the blockchain's *upload speed* as the number of distinct transactions per second (tps) that a user sends to processes, and we use Ω_{up} to refer to the upload speed of the blockchain Ω (i.e. the number of distinct transactions sent by any user to processes per second). We use Ω_{do} to refer to the *download speed* of the blockchain, which is the number of finalized transactions per second. All users can verify finalized blocks locally for correctness, in that all transactions in that block are valid given the current state of the blockchain. One can then define the *verification speed* of a user in transactions per second. We then speak of the *verification speed* of a blockchain Ω , noted Ω_{ver} as the average of the verification speed of all of its users.

Having defined these terms, we now propose a definition of the blockchain trilemma. The purpose of this definition is to set a goal for blockchains, in that a perfect blockchain would guarantee these three properties.

Definition 2.2.7 (Blockchain trilemma). Let Ω be a blockchain. Then Ω solves the blockchain trilemma if it satisfies the following properties:

- **Scalability.** Ω finalizes blocks such that $\Omega_{do} \geq \min(\Omega_{ver}, \Omega_{up})$.
- **Decentralization.** Any user can become a process and have one of his proposed blocks finalized with non-null probability.
- **Security.** Ω is an SMR.

Security and the definition of finalized blocks is the property that ensures that every blockchain is also an SMR, while scalability measures performance and decentralization ensures the minimum requirement for the permissionless nature of blockchains. It is easy to see that designing and implementing a non-trivial (i.e. such that the number of users is $m > 1$) blockchain that satisfies these three properties in the presence of a Byzantine adversary controlling some of the users is not immediate. In fact, one could argue that it is impossible (as it has been often done for the blockchain trilemma). However, two of the three properties are easily obtainable. First, in order to satisfy scalability and decentralization without security, one can simply let each process decide individually. Second, scalability and security can be satisfied by relying on a trusted third party. Third, a trivial solution that does not ever finalizes any block satisfies security and decentralization without scalability.

Thus, the blockchain trilemma intuitively applies as expected. In fact, a combination of previous results already provide an intuition of the impossibility of the trilemma as defined in this dissertation. We show this in the following conjecture.

Conjecture 2.2.1. It is impossible to solve the blockchain trilemma.

Intuition. A blockchain that guarantees that all users agree on the ordering of all finalized blocks (security) has consensus number $+\infty$ [15]. A protocol that solves distributed consensus has bit complexity at least $\mathcal{O}(n^2)$ [29]. A model that maps this lower-bound on bit complexity to the size of each finalized block, and assumes a computation time to verify these blocks that is less costly than exchanging $\mathcal{O}(n^2)$ bits, would thus indicate that the three properties are impossible to be satisfied together. \square

As such, we propose instead metrics that measure the scalability, security and decentralization of blockchains.

Security metrics. We measure security by comparing the set of assumptions under which the blockchain satisfies its properties. Particularly, we focus in this dissertation on the adversarial model, and on the communication network. Our proposals start by at least tolerating t_ℓ Byzantine faults in partial synchrony, but, as we will show, we extend beyond these classical bounds. We also justify the need for partial synchrony and for tolerating powerful adversaries.

Decentralization metrics. A decentralization metric has already been slightly studied by the state of the art [26]. Intuitively, this metric requires an analysis of the probability that an adversary (possibly dynamic) controlling initially a percentage of users (and not processes), manages to take control of the committee sortition protocol so that it can either predict or bias future committees. We present such analysis in Chapter 6.

Scalability metrics. In order to quantify the scalability of a blockchain, we inherit from previous results and metrics that quantify the communication complexities of the protocol. In particular, as already noted, a blockchain that is secure can only finalize blocks after its processes exchange at least $\mathcal{O}(n^2)$ bits. However, we follow the approach of recent works [79, 31, 27, 80, 81, 23, 28] in order to characterize specific metrics for the case of blockchains, in addition to the classical bit, message and time complexities. We propose three modifications to consider the complexities of blockchains:

1. **Normalized complexities.** Consider two blockchains, A and B, which have the same bit and message complexities, but A finalizes $\Omega(n)$ blocks in one iteration of the protocol while B finalizes $\mathcal{O}(1)$ (suppose all blocks have the same number of transactions). Clearly, A produces more outputs for the same message and bit complexities. The normalized message and bit complexities reflect this greater throughput of A by providing the complexity metrics divided by the number of outputs in one iteration of the protocol. Thus, if A finalizes $\Omega(n)$ blocks in an iteration of a protocol with a bit complexity of $\mathcal{O}(n^3)$, while B produces $\mathcal{O}(1)$ outputs with a bit complexity of $\mathcal{O}(n^2)$, then these protocols have the same normalized bit complexity of $\mathcal{O}(n^2)$. This is the case of multiple recent proposals [27, 80, 81, 23, 28].

2. **Amortized complexities.** Similar to normalization, amortization looks at the number of outputs produced after n iterations of the protocol, and divides the complexities by the number of outputs. For example, some leader-based protocols can apply a round-robin rotation of the leader every time the protocol fails to produce an output, meaning that if there are $t_\ell = \mathcal{O}(n)$ Byzantine processes (and assuming synchrony or after GST), then $\Omega(n)$ blocks are produced in n iterations. In this example, however, this result requires an assumption on the adversary being static, or slowly-adaptive [100], since otherwise in the worst-case the adversary can always initially corrupt the leader. Amortization has already been presented by recent works [79, 31, 132].

3. **Per route complexities.** Suppose two blockchains whose servers communicate via the Internet. Suppose that both blockchains, A and B, have the same time, message and communication complexity per output, even normalized and amortized, and suppose the bit complexity is $\mathcal{O}(n^2)$. However, protocol A uses $\mathcal{O}(n)$ of these pairwise channels, sending $\Omega(n)$ bits through each of them, while protocol B distributes the bits sending $\mathcal{O}(1)$ bits across each of the $\Omega(n^2)$ network channels. In this case, protocol B is not affected by the bottleneck on network bandwidth on the exploited network routes, meaning that it should scale better since both are implemented on the Internet or, more generally, on a wide area network (WAN). This is in fact the behavior that has been measured by recent works [27, 80, 81, 23, 28], which have at-first comparable complexity metrics, but that scale better due to their distribution of shared bits across more channels.

There are already a number of works that remark the importance of an implementation that evenly scatters the bits exchanged across the entire network [27, 80, 81, 23, 28]. The protocols of these proposals are either leader-less or democratic. The distinction between *leader-less* and *democratic* protocols is that leader-based, democratic protocols may have a leader that

proposes the output, as long as this proposed output is a subset of the inputs, and these inputs are provided by multiple processes [27, 80]. In contrast, leader-less protocols are not only democratic but they also do not have a leader, i.e. a single process that proposes a specific subset of the inputs. We argue in Chapter 6 the advantages of leader-based democratic protocols compared to leader-less protocols [81, 23, 28].

Chapter 3

Layer-2 Without Synchrony

The efforts of the blockchain community to design decentralized, secure and scalable blockchains have led to many proposals. However, the vast majority of them came to light through mediums without peer review, in many cases shared online in the form of an informal white paper. For the problem of scaling blockchains, this has led to some works that scale at the cost of less security, through the abuse of strong assumptions and threat models. Layer-2, or offchain protocols, were created with the aim of connecting and scaling first generation blockchains. As we mentioned in Chapter 2, previous works have already warned against the dangers of current layer-2 proposals, specially due to their abuse of synchrony and their trade-off between a low synchrony bound for scalability and a large synchrony bound for security.

In this chapter, we abstract the trade-off for synchronous offchain protocols derived from an abuse of the synchrony assumption, to then introduce, implement and test on the Lightning Network a novel attack, the Lockdown attack, that stems from our previous work [83, 91]. We then set the first stone towards secure offchain protocols by proposing and proving the correctness and optimal resilience of the first offchain protocol without synchrony to date, the *Platypus* protocol.

Summary. In summary, we present the following contributions in this chapter:

- i) We outline the trade-off between security and scalability derived from synchrony in layer-2.
- ii) We present the Lockdown attack, a novel attack against offchain payment channels, factories and networks.
- iii) We implement the Lockdown attack in the Lightning Network testnet, showing that it successfully prevents users from performing payments regardless of the implementation and parameters of the Lightning Network.
- iv) We present a novel formalization and definition of the offchain, childchain, and sidechains problems.
- v) We present the Platypus protocol, the first offchain protocol that is scalable and safe without a synchrony assumption.

- vi) We prove the correctness of Platypus for the offchain, childchain, and sidechains problems, and show that it is resilient optimal.

Chapter outline. We explain the vulnerabilities associated with synchrony in Section 3.1, where we also present a novel attack against synchronous offchain protocols, the Lockdown Attack. Section 3.2 extends the model to introduce the offchain problem. We present Platypus, a novel childchain protocol that solves the offchain problem without assuming synchrony, in Section 3.3, and show its correctness and optimal resilience in Section 3.4. We analyze Platypus' complexity in Section 3.5, and discuss improvements to Platypus and its applications to sidechains in Section 3.6. Finally, Section 3.7 concludes the chapter.

3.1 Why layer-2 that rely on synchrony are more vulnerable

By relying on offchain computation, layer-2 (or offchain protocols) avoid communicating and/or storing information directly in the blockchain, hence bypassing an important bottleneck, but also limiting transparency of selected transactions to ensure privacy. Channel networks and channel factories offer private and fast payments, but they can only perform payments if users have an existing route of channels with one another. As a result, their scalability and privacy are actually subject to proper handling of the network topology and vulnerable to routing attacks [91].

There is, therefore, great interest in designing proper childchain protocols that allow blockchains to host the creation and destruction of other smaller blockchains that depend on them. Unfortunately, as far as we know all childchains [96, 97] use timelocks that only work under the assumption that the communication is *synchronous*, in that every message gets delivered in less than a known bounded amount of time [13], an assumption that is known to be violated at times over the internet. But more dramatically, assuming synchrony exposes offchain protocols to various attacks, like Denial-of-Service or Man-in-the-Middle, that are common practice to double spend [112].

3.1.1 Abstract vulnerability

To illustrate the problem of synchronous offchain protocols, consider an execution using time-locks¹ illustrated in Figure 3.1 in which time increases from top to bottom. First, Alice transfers € coins to Bob outside of the chain, or *offchain*, before Bob acknowledges the transfer. Bob can then take actions in response to this transfer thinking, wrongly, that he will have sufficient time to prove the fraud if Alice tries to claim back the coins. Let us consider that Alice is Byzantine and claims back the ownership of the coins, which triggers a *timelock*, a safe guard delay during which the coins are locked to give an opportunity to other participants to prove fraudulent activity before the coins are transferred back. As part of the protocol, Bob gets notified but due to an unforeseen delay, he does not manage to prove the fraud before the end of the timelock.

¹Although most implementations use timelocks like here presented, some other proposals are synchronous and use no timelocks. Note however that synchronous protocols are inherently vulnerable to non-synchronous periods

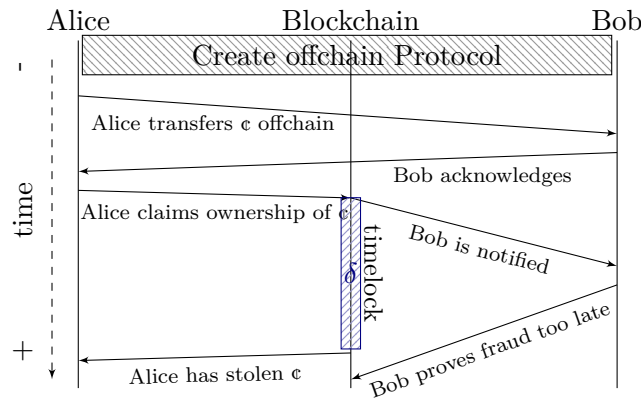


Figure 3.1: Alice can steal Bob’s coin if Bob messages are delayed such that Bob’s reply takes longer than the timelock δ .

Its ϵ coins are thus stolen. In fact, the problem of synchronous offchain protocols derives in a critical trade-off: timelocks should be as small as possible for scalability, but they should be large to guarantee security.

3.1.2 Lockdown attack

More than just theorizing with the problems of synchrony, we illustrate the problem with an attack on the first and most popular offchain payment network, the Lightning Network [82] (LN), which we call the Lockdown attack. This attack is a continuation from our previous work [91] to showcase the dangers of synchrony in layer-2 and question whether these proposals allow for scalability and private transactions. We shortly describe the attack and results in this section, and refer to Appendix A for a detailed description.

The LN is a separated peer-to-peer (p2p) network connected to the main Bitcoin p2p network with users that run a LN software client [174, 175, 176]. Each user maintains multiple p2p connections in the LN and also a connection in the Bitcoin main p2p network.

The core underlying concept for our proposed attack is the *multihop* approach. Payments in the LN between users that do not share a direct payment channel have to be routed through a multihop path, also known as a payment route. Since this describes a graph, we speak in this section of a *node* to refer to the LN client run by a user. The source node of a payment is the payer, and the end node is the recipient. In all current LN implementations, such a route is constructed by the source node that performs the payment. To allow these constructions, nodes in the LN maintain a topology structure of the LN graph that is used for route discovery. LN implementations, given a target node, return the most suitable route based on the number of hops and the fees each hop charges for routing the payment. However, being the LN a p2p environment, there is no deterrent for a source node to compute the payment route at his choice with the information he has available.

In the multihop approach, payments at each individual payment channel cannot be performed exactly in the same way than that with a single hop. An intermediate user has to enforce he would receive the payment from the preceding node once he has performed the payment to the next one, otherwise he would lose the amount of the payment. The enforcement

of this type of atomic exchange between all the nodes of the path (i.e., all simple one-hop payments have to be completed or none can be processed) is performed using Hashed Timelock Contracts (HTLCs) [86]. In a (synchronous) HTLC between the sender A and the receiver B , A can deposit coins that can be redeemed by B if B can perform a digital signature and provide a preimage of a hash value. Furthermore, the deposit performed by A has an expiration date after which A can retrieve the deposit providing a digital signature. For a two-hop payment, $A \leftrightarrow B \leftrightarrow C$, the idea is that C generates a random value x and sends $h(x)$ to A where $h(x)$ returns the hash of a value x . A performs the single hop payment to B with an HTLC based on $h(x)$ and B also performs the single hop payment to C with an HTLC based on the same value $h(x)$. In that way, since C knows x , he can redeem the transaction from B , but redeeming the transaction implies revealing the value of x . This implies that B may also redeem the payment from A .

When node B_1 performs a payment to node B_m in the LN using the route $B_1 \rightarrow B_2 \rightarrow \dots \rightarrow B_m$, the atomicity needed in such operation implies that all route payments cannot be executed until the last node of the route, B_m , provides the corresponding preimage x of the $h(x)$ included in the HTLC. In a normal scenario, B_m reveals this preimage as soon as he receives the payment in his channel, because he wants to collect the payment. However, if the payment gets stuck for any reason in node B_i , all payments from node B_1 to node B_i will be locked. To bound the locking time, B_1 sets a total timelock, dependent on the synchrony bound. The time frame for the payment, determined as an absolute blockheight value, and known as its *expiration blockheight*, θ , limits the time that these funds will be locked in case the payment does not succeed.

When the payment is being routed, every node of the route also decreases the value θ . Each node of the LN advertises for each of its channels, the value δ that will be used for decreasing θ at each hop. With this public information, the payer creates the route with an initial θ ensuring that after subtracting each δ of each intermediate node, the last node will not obtain a expired time, that is $(\theta - \sum_{i=2}^m \delta_i) > 0$. Notice that this mechanism allows the payer to bound the time a payment will be locked but, without any other mechanism, a malicious payer could lock the funds of intermediate nodes by setting a large initial value θ . To avoid this situation, each node sets his own T_{max} value that bounds the locking time of a payment. Then, when a node receives a payment as an intermediate node route, if $\theta > T_{max}$ the node will refuse to route the payment and the payer will have to choose another route.

3.1.2.1 Attack overview

The proposed attack is focused on a target victim A , a node of the LN. The goal of the adversary is to block the victim A as a middle node in multipath payments. By achieving this goal, an adversary may obtain a dominant position in the LN, given that blocking some selected nodes may let the adversary be the main gateway to route payments. This allows the adversary to have a dominant position that can be exploited either in order to gather information or just to increase the benefits as a LN gateway node. We defer to Appendix A a formal specification of the design of the attack and the adversarial knowledge to deploy it (Section A.1). Also, to

simplify the description of the attack, we delay to Appendix A some of the maximum values that LN implementations introduce (Section A.3), and how these values impact the cost of real attacks (Section A.2). For the remainder of this chapter, we use the following notation: we assume that the victim node is A , the adversary is \mathcal{M} and A has open channels with a list of n different nodes, denoted by B_i for $i = 1, \dots, n$. Furthermore, we denote by C_{AB_i} the capacity of the channel that A and B_i have opened, i.e. how many coins are placed in the channel, and by $balance_{AB_i}$ (resp. $balance_{B_iA}$) the balance that A (resp. B_i) has in this channel, i.e. how many of the coins belong to whom. We denote C_{attack} the capacity that \mathcal{M} has to hold in channels in the LN to perform the attack.

To describe our attack, we use a simple scenario where the victim A is a hub between two users, B_1 and B_2 , as depicted in Figure 3.2. Capacity values are $C_{AB_1} = o_1 + o_4$ and $C_{AB_2} = o_2 + o_3$, being o_i the balances in each direction for each channel. The objective of the adversary \mathcal{M} is to disrupt the availability of A by blocking the availability of either incoming links or outgoing links, that is rendering $o_1 = 0$ and $o_3 = 0$ or $o_2 = 0$ and $o_4 = 0$.

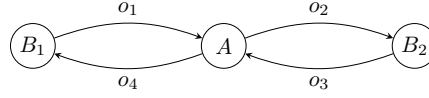


Figure 3.2: Example scenario of the Lockdown attack, where victim A is a hub between two users B_1 and B_2 . The objective of the adversary \mathcal{M} is to disrupt the availability of A , by blocking the availability of either incoming links or outgoing links, that is, rendering $o_1 = 0$ and $o_3 = 0$ or $o_2 = 0$ and $o_4 = 0$.

To perform the attack, \mathcal{M} opens a channel with A as depicted in Figure 3.3a. The attack complexity depends on the balances between A and B_i and we can distinguish the two following cases:

Shorter loop – The first case is when $o_1 \leq o_4$ and $o_3 \leq o_2$. Notice that with these conditions, $o_1 + o_3 \leq o_2 + o_4$, which means that \mathcal{M} would prefer to block incoming paths to A , as this way C_{attack} is lower than by blocking outgoing connections. As such, \mathcal{M} can block all incoming paths by performing two single payments with a short loop. The first payment will follow the route $M \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow M$ with value o_1 and the second payment will follow the route $M \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow M$ with value o_3 . With these payments $balance_{B_1A} = balance_{B_2A} = 0$. Notice that with this scenario the channel that \mathcal{M} has to open with A to perform the attack needs a capacity² $C_{attack} = C_{MA} = 2(o_1 + o_3)$.

Longer loop – In case either $o_1 > o_4$ or $o_3 > o_2$, then the adversary needs to proceed in a different way.³ Without loss of generality, assume that $o_1 > o_4$ and $o_3 \leq o_2$ and also that $o_1 + o_3 \leq o_2 + o_4$ so \mathcal{M} would prefer to block incoming paths to A . With this balance distribution, \mathcal{M} can perform the aforementioned short loop to block the incoming path from B_2

²The capacity that \mathcal{M} has to open with A is the double of the payment value, as the payment is performed by \mathcal{M} but also has to return to \mathcal{M} to extend the time that the payment is blocked for.

³Notice that if both inequations hold, then $o_1 + o_3 > o_2 + o_4$ and \mathcal{M} would prefer to block outgoing paths as in the “Shorter loop” case.

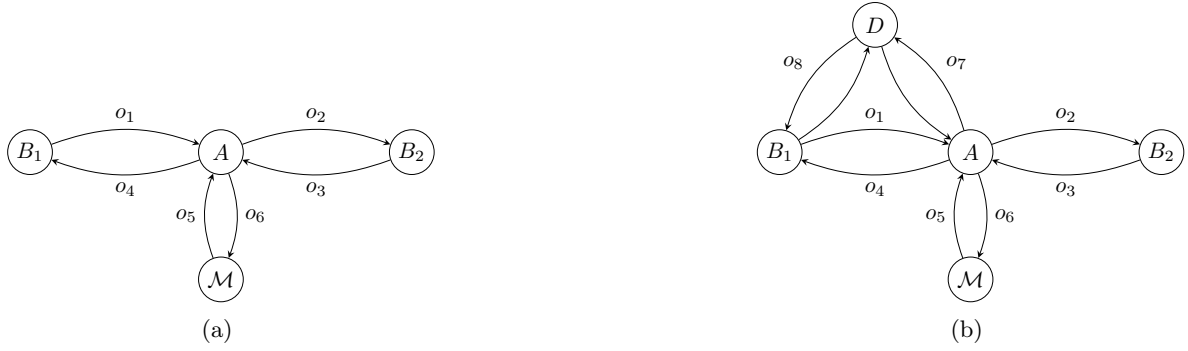


Figure 3.3: (a) Simple scenario of the Lockdown attack, in which the adversary \mathcal{M} opens channels with A to lock its funds with B_1 and B_2 . (b) Simple scenario with an external node, in which the adversary can use the external node D to lock the remaining balances.

by performing the payment of value o_3 following the route $\mathcal{M} \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow \mathcal{M}$. However, given that $o_1 > o_4$, \mathcal{M} cannot perform a payment route $\mathcal{M} \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow \mathcal{M}$ with value o_1 since the channel AB_1 has $balance_{AB_1} = o_4 < o_1$. At most, \mathcal{M} can perform a payment with value o_4 through the path $\mathcal{M} \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow \mathcal{M}$. This payment locks o_4 but some balance is still available in the channel, precisely $o_4 - o_1$. For \mathcal{M} to lock that capacity of the channel, as the path $A \rightarrow B_1$ is already exhausted, \mathcal{M} needs another path from A to B_1 with capacity $o_4 - o_1$ and that exact direction. Figure 3.3b shows a simple example in which there exists a node D with open channels with A and B_1 and such that $balance_{AD} = o_7 \geq o_4 - o_1$ and $balance_{DB_1} = o_8 \geq o_4 - o_1$. In that case, \mathcal{M} can perform a second payment with value $o_4 - o_1$ with route $\mathcal{M} \rightarrow A \rightarrow D \rightarrow B_1 \rightarrow A \rightarrow \mathcal{M}$. This payment will lock the remaining funds of $B_1 \rightarrow A$.

The hard assumption of the existence of node D can be relaxed with the existence of multiple possible paths that all together can route the total $o_4 - o_1$ value. Notice, however, that the payment graph topology hardly determines the existence of such paths.

3.1.2.2 Attack results

We perform a test in a simnet controlled environment to validate that our claims are correct and that the routes generated in our attack containing loops can effectively be deployed in the three most relevant available implementations of the Bitcoin LN, namely lnd [175], cl [176] and eclair [177].

For the network topology, we take a snapshot of the topology of the LN⁴ of the Bitcoin mainnet on July, 9th, 2019 at 12:00 CET time. To execute the attack algorithm, the adversary needs to complement the information of the network graph with further data. The information needed is: the balance of each channel and the values T_{max} for each node of the network.

Regarding the balances, they can be obtained by executing the attack we described in our previous work [91]. However, instead of performing such an attack, we have assigned the

⁴This can be obtained, for instance, with the instruction `describegraph` of the lnd implementation.

balances of each channel using different statistical distributions, trying to reproduce the different scenarios that could be found in the network. In order to assign balances to channels, we proceed in the following way: for each channel, first the balance of one of the nodes is randomly selected using one of the selected distributions, and taking the capacity of the channel as the maximum possible value to generate. Then, the balance of the other node in the channel is set as the remaining balance (that is, the capacity minus the balance). Five different distributions are used to assign balances to channels: *deterministic*, *uniform*, *normal*, *exponential*, and *beta*. The *deterministic* distribution always assigns half of the capacity of the channel to each of the nodes; the *normal* distribution is used with $\mu = 0.5$ and $\sigma = 0.2$; the *exponential* distribution uses $\alpha = 1$; and the *beta* distribution $\alpha = \beta = 0.25$. We detail further the setup for the simulations in Appendix A (Section A.2.2).

Distribution	Implementation	<i>AER</i>	Blocked capacity	Channels needed	\widetilde{TBT}
Beta	lnd	0.291	86.86 %	67.8	0.31
	c-lightning	0.203	82.23 %	49.7	0.07
	eclair	0.584	86.30 %	133.8	0.05
Deterministic	lnd	0.200	100.00 %	52.0	0.44
	c-lightning	0.100	80.40 %	26.0	0.13
	eclair	0.500	100.00 %	129.0	0.09
Exponential	lnd	0.229	92.55 %	55.1	0.36
	c-lightning	0.135	80.22 %	34.0	0.09
	eclair	0.512	92.30 %	123.4	0.07
Normal	lnd	0.230	96.79 %	62.5	0.39
	c-lightning	0.149	84.26 %	39.0	0.11
	eclair	0.479	96.89 %	134.9	0.08
Uniform	lnd	0.268	93.05 %	69.4	0.35
	c-lightning	0.149	82.87 %	45.6	0.09
	eclair	0.557	93.02 %	140.3	0.07

Table 3.1: Attack results for different balance distributions.

Table 3.1 shows that the attack is effective in all scenarios (implementations and balance distribution). Notice that in the worst-case, for a Beta distribution in which the node runs an eclair implementation, the *Attack Effort Ratio* (AER), which describes the ratio between the capacity needed to perform the attack and the capacity that the attack blocks, is 0.584. This means that half of the capacity of the victim suffices for the attacker to block 86.30% of the victim’s capacity. In fact, the percentage of the victim capacity blocked is large for all the scenarios, never below 80%. Moreover, Table 3.1 also shows the *Normalized Total Blocked Time* (\widetilde{TBT}). We detail how we calculate these values in Appendix A (Section A.1), which results in 0 if the attack does not lock any balance, and in 1 if the attack maximizes the time during which the balance is locked (per implementation). The \widetilde{TBT} also shows that lnd implementations

allow the adversary to block more capacity over time.

We provide more simulation results in Appendix A (Section A.2.2), as well as possible countermeasures (Section A.4). Nevertheless, as we have already noted, this attack and others stem directly from the abuse of the synchrony assumption. For this reason, we present first a model that we will use to formalize the offchain problem in Section 3.2, and illustrate the Platypus protocol in Section 3.3, the first offchain protocol without synchrony.

3.2 Model

In this section, we define additional assumptions to the model already defined in Section 2.2, that we will use throughout this chapter.

- *Threshold signatures.* Our model requires accounts to authenticate with a cryptographic primitive enabling non-interactive aggregation, such as those of previous works [97, 178]. For simplicity, and without loss of generality, we assume that accounts are not reusable. In particular, the same coins should not go back to the same process in the same account, to prevent a variant of the ABA problem (see Section 3.6). In the remainder of this chapter, we abuse the term process as an account that the process owns, unless stated otherwise.

- *Minimal transfers.* Given a sequence $seq = \{z_j TR_{\delta_{j+1}, \mathfrak{e}_i} z_{j+1}\}_{j=c}^{d-1}$ of transfers over some time range $[\delta_c, \delta_d]$, between creation time δ_c and destruction time δ_d , for coin \mathfrak{e}_i , we refer to the minimal transfer as the single transfer $z_c TR_{[\delta_c, \delta_d], \mathfrak{e}_i} z_d$, which is always an element of the transitive closure (see Section 2.2.7.1). For a set of operations defined over all coins within a time range $[\delta_c, \delta_d]$, we denote the minimal transfer set TR^- as the set of all minimal transfers, which is at least a set of idempotent transfers of the form $z_i TR z_i$.

- *Offchain problem.* Given a blockchain Ω of P processes, the offchain protocol consists of executing a sequence seq of transfers offchain. First, processes $Q \subsetneq P$ must create an offchain protocol Γ by writing a transaction in the original chain Ω —effectively depositing funds from Ω into Γ . Then they transfer coins offchain between themselves using Γ . Finally they can destroy this protocol Γ . To this end, the offchain protocol consists of at least two main procedures, *creation* and *bulk close*. We will explain later how the participation in Γ is made dynamic using splice in and splice out procedures to accept new participants and for existing participants to leave Γ , respectively. After a series of transfers in Γ , processes can propose to bulk close it by *proposing COMMIT*. Processes *decide to COMMIT*, in that they effectively agree to accept these transfers and to close and destroy Γ , or *decide to ABORT*, in that they disagree with the transfers and refuse to close Γ . We formalize this offchain problem in Definition 3.2.1.

Definition 3.2.1 (Offchain). A protocol solves the *offchain problem* if it satisfies the following properties:

- **Termination.** Every honest process decides COMMIT or ABORT on some sequence of transfers seq for which some process proposed COMMIT.
- **Agreement.** no honest process decides COMMIT on two different sequences seq and seq' .

- **ABORT-Validity.** if an honest process proposes ABORT for a sequence seq , then all honest processes decide ABORT for this sequence seq .
- **COMMIT-Validity.** if no honest process proposes ABORT for sequence seq for which some process proposed COMMIT, then all honest processes decide COMMIT for sequence seq .

Notice that, in our definition, aborting is implicit and proposing ABORT is not an input of our algorithm as we will see in Algorithm 3. In particular, COMMIT-Validity can be ensured by requiring a process to provide a valid proof of fraud (PoF) when proposing to abort, the invalidity of the PoF allows honest processes to ignore the ABORT proposal and its validity guarantees that all honest processes will observe this PoF. Finally, note that our termination does not imply that the offchain protocol gets closed. Instead, it means that all honest processes decide either to COMMIT and close the protocol or to ABORT, not closing the protocol. This is not a problem since, as we will explain in Algorithm 5, any honest process can cash out the coins that it knows it owns at any moment.

In order to achieve privacy, we introduce another property stating that some decisions of the offchain protocol do not have to be written in the blockchain:

- COMMIT-Privacy/Lightness: If honest processes decide COMMIT on a sequence seq of transfer operations made in Γ between δ_c and δ_d , then $\forall p \in P \setminus Q$, p only learns/stores TR^- , the minimal transfer set of seq .

A *childchain* Ψ is a particular class of offchain protocol in that it is a blockchain Ψ that is created by another blockchain Ω , known as its *parentchain*, and that implements an offchain protocol Γ .

3.3 Secure childchains without synchrony

In this section we present Platypus, a novel childchain protocol that solves the offchain problem without assuming synchrony. Platypus consists of both an offchain protocol and a childchain that are denoted respectively Γ and Ψ in the remainder of the chapter. Given a parentchain Ω , processes can use the protocol Γ by depositing funds from Ω to Ψ , that effectively creates the childchain. Then transfers can be done directly on Ψ offchain before the bulk close takes place.

As Platypus interacts with two blockchains, each with a different set of users and committee, we extend the model of Section 2.2.1 in this section. The parentchain Ω and childchain Ψ have a set M_Ω of $|M_\Omega| = m_\Omega$ users and M_Ψ of $|M_\Psi| = m_\Psi$ users, respectively, with a committee $N_\Omega \subseteq M_\Omega$ of $|N_\Omega| = n_\Omega$ processes and a committee $N_\Psi \subseteq M_\Psi \subseteq M_\Omega$ of $|N_\Psi| = n_\Psi$ processes, respectively. Note that n_Ψ is the number of all processes joining the Platypus protocol. The consideration of two committees also requires an adaptation of the aforementioned t_ℓ for Byzantine behavior (see Section 2.2.1). The adversary \mathcal{M} is thus bound to two constraints, which limit the maximum number of Byzantine faults that it can control, being these $t_0 = \lceil n_\Omega/3 \rceil - 1$ and $t_1 = \lceil n_\Psi/3 \rceil - 1$.

Although we do not provide an implementation of the blockchain Ψ (resp. Ω), we assume that Ψ (resp. Ω) is secure (i.e. it uses deterministic consensus to not fork): a blockchain assuming partial synchrony and t_1 (resp. t_0) Byzantine processes suffices [23, 28].

3.3.1 Overview

Γ is depicted in three main procedures: a creation (Algorithm 1), a bulk close (Algorithm 2) and an abort (Algorithm 3). (Splice in and splice out procedures are deferred to Section 3.6). Processes can ABORT or COMMIT sequences of transfers done in Ψ . In particular, a process proposes ABORT by creating an abort transaction (and sharing it) in line 7 of Algorithm 3 and proposes a COMMIT in line 9 of Algorithm 2. A process decides COMMIT at line 10 (Algorithm 2) only after a number of processes propose COMMIT and decides ABORT at line 11 (Algorithm 2) only when there exists a valid abort transaction.

3.3.2 Creating a Platypus chain

Users can create a Platypus chain by publishing a transaction on Ω . After that transaction is final, the funds referred to in this transaction are locked and ready to be used by the Platypus protocol Γ . In general, a Platypus creation transaction (tx_{plcr}) is a transaction that:

- Has a new Platypus id ($plid$) that uniquely identifies it.
- Specifies a consensus protocol for the Platypus blockchain Ψ to decide on a new block. W.l.o.g., we assume DBFT [23] to be the default protocol.
- Specifies a number $h_0 > f$ of processes required to create Ψ , where f is the total number of faults. For simplicity and to match with the optimal result (see Theorem 3.9), we choose $h_0 = \lfloor 2n_\Psi/3 \rfloor + 1$.
- Defines a new function `abort(...)` that specifies when a user can decide ABORT on the protocol (such as a Platypus bulk close transaction being aborted).
- Specifies a set of processes and their balances that go in Ψ through this transaction.
- Once written in Ω , the funds can only be spent in Ψ .

Algorithm 1 shows the protocol to create a Platypus chain. The call to `num_signers(tx)` returns the amount of signers of tx , while the call to `verify(tx, {msg})` verifies the validity of the transaction and signed messages. We define two main interactions of the Platypus protocol with both the childchain and the parentchain: sending transactions and reading transactions. The Platypus protocol Γ sends transactions to Ω or Ψ by invoking `send({ Ω, Ψ }, tx)` and `acsend({ Ω, Ψ }, tx)`—standing for “atomic commit send”. In the former, the function returns once the transaction is written in the corresponding blockchain or a transaction that spent the same funds has been written (meaning this transaction became invalid), while the latter returns ABORT or COMMIT and the respectively written transaction in a response message. This

response is received by all processes as it is a result of the Platypus blockchain. Reading transactions is performed by the call to `is_written($\{\Omega, \Psi\}, tx$)` that returns True or False depending on whether the transaction was written or not in the blockchain. Messages are signed to prevent Byzantine processes from adding third parties without the agreement of honest processes.

3.3.3 Closing a Platypus chain

A Platypus bulk close transaction splices all funds out of the Platypus blockchain without compromising its security (agreement), and without requiring all processes to join together in its destruction (termination). It is still a normal transaction in the Platypus blockchain, meaning that it requires enough processes h_0 to agree with writing it in Ψ . Algorithm 2 shows the protocol to bulk close a Platypus chain. A Platypus bulk close transaction signed by some processes returns back the updated balances of all processes in the parentchain Ω , unless it is aborted. Once written in both Ψ and Ω , the coins can be spent only in Ω .

3.3.4 Aborting a closing attempt

A Platypus bulk close transaction with insufficient signatures can either be a valid, ongoing Platypus bulk close, or an attempt to commit fraud. To prevent this, and guarantee termination and ABORT-validity, we introduce the abort transaction.

A transaction may be invalid if it spends a coin formerly owned by a user, but that was transferred to another user later in Ψ . The abort function runs for every Platypus bulk close transaction received that is not valid, i.e. that spends some input already spent. If the transaction is not valid due to signatures not matching, then it will not be written in the parentchain, so the abort function ignores this case.

Therefore, a user can see that a transaction tx is not valid if an old owner claims ownership of a spent coin in tx , as checked by `coins_spent(...)`, shown in Algorithm 3. Notice that, while a COMMIT requires h_0 processes to commit to the transaction (such as a Platypus bulk close transaction), any user $u \in M_\Psi$ can create a valid abort transaction. The call to `extract_spent(tx)` returns the coins that were spent. The call to `get_block_min_blockheight(C_S)` returns the block of minimum blockheight out of all the blocks that store a transaction spending each of the spent coins, i.e. proofs-of-fraud (PoFs). Finally, `processes(b/tx)` returns the set of processes that signed block b or transaction tx .

Intuitively, this algorithm proves invalidity by iterating through Ψ , looking for processes that validated both this bulk close and some progress in Ψ that conflicts with it (i.e. some blocks that spent some of the coins). This set of processes is the set of *fraudsters*. Other processes that only validated the transaction might simply have had an old view of the Platypus chain, under the partially synchronous model. Nonetheless, the existence of such block is enough to create the abort transaction, even if the set of fraudsters is empty.

The iteration starts from the block with minimum blockheight of all the blocks that show that some coin c was transferred from p_i to p_j , for some p_i that claims ownership of c , in line 3. The algorithm then continues to account for fraudsters.

Algorithm 1 Platypus creation procedure.

\triangleright State of the algorithm
 Ω , the parentchain
 Γ , the Platypus protocol
 M_Ω , the set of users in the parentchain
 $M_\Psi \leftarrow \perp$, the set of users in the Platypus chain
 $N_\Psi \leftarrow \perp$, the set of processes in the Platypus chain
 n_Ψ , the amount of processes required in Ψ
 \mathbb{C}_i , coins that belong to process p_i
 job_i , Boolean defining if p_i is PROCESS or just USER
 $plid$, the Platypus chain identifier
 $msg_i = \langle \mathbb{C}_i, plid, job_i, \sigma_i \rangle$, signed message to join.
 σ_i , signature of msg_i by p_i
 $tx_{plcr} \leftarrow \perp$, the Platypus creation transaction

\triangleright PHASE 1: process p_0 initiates request
 1: $msg_0 \leftarrow \text{sign}(\langle \mathbb{C}_0, plid, job_0 \rangle)$
 2: $\text{multicast}(msg_0)$ to M_Ω

3: \triangleright PHASE 2: Rest of processes who want to join reply
 4: **when** msg_0 is received from p_0
 5: $msg_i \leftarrow \text{sign}(\langle \mathbb{C}_i, plid, job_i \rangle)$
 6: $\text{multicast}(msg_i)$ to M_Ω

\triangleright PHASE 3: Process $p_i \in N_\Psi$ gathers enough processes
 7: **when** msg_j is received from p_j **and** $p_j \notin M_\Psi$
 8: $\{M_\Psi, \mathbb{C}_{M_\Psi}\} \leftarrow \{M_\Psi \cup \{p_j\}, \mathbb{C}_{M_\Psi} \cup msg_j.\mathbb{C}_j\}$
 9: **if** ($msg_j.job_j = \text{PROCESS}$ **and** $p_j \notin N_\Psi$) **then**
 10: $\{N_\Psi, \mathbb{C}_{N_\Psi}\} \leftarrow \{N_\Psi \cup \{p_j\}, \mathbb{C}_{N_\Psi} \cup msg_j.\mathbb{C}_j\}$
 11: **if** ($|N_\Psi| = n_\Psi$) **then** \triangleright enough processes to start transaction
 12: $tx_{plcr} \leftarrow \text{createPlatypusTx}(\mathbb{C}_{M_\Psi}, \mathbb{C}_{N_\Psi}, plid)$
 13: $tx_{plcr} \leftarrow \text{sign}_i(tx_{plcr})$
 14: $\text{multicast}(tx_{plcr}, \{msg_k\}_{p_k \in M_\Psi})$ to N_Ψ

\triangleright PHASE 4: $p_i \in N_\Psi$ signs and broadcasts until it gets enough signatures
 15: **when** ($tx_{plcr}, \{msg_j\}_{p_j \in M_\Psi}$) is received **and not is** $_written(\Omega, tx_{plcr}, plid)$
 16: **if** ($\text{verify}(tx_{plcr}, \{msg_j\})$) **then** $tx_{plcr} \leftarrow \text{sign}_i(tx_{plcr})$
 17: **if** ($\text{num_signers}(tx_{plcr}) < \lfloor 2n_\Psi/3 \rfloor + 1$) **then**
 18: $\text{multicast}(tx_{plcr}, \{msg_j\})$ to N_Ψ
 19: **else** $\Gamma.\text{send}(\Omega, tx_{plcr})$ \triangleright enough signatures

Algorithm 2 Platypus bulk close procedure.

▷ State of the algorithm

Ω, Ψ, Γ , the blockchain, Platypus blockchain and protocol

M_Ψ, N_Ψ , the set of users and processes in Ψ

C_i , the coins that belong to process p_i

$tx_{plcl} \leftarrow \perp$

▷ PHASE 1: Some process p_0 creates and broadcasts

1: $tx_{plcl} \leftarrow \text{createBulkCloseTx}(C_{M_\Psi})$

2: $tx_{plcl} \leftarrow \text{sign}_i(tx_{plcl})$

3: **multicast**(tx_{plcl}) to N_Ψ

▷ PHASE 2: $p_i \in N_\Psi$ signs and broadcasts transaction

4: **when** tx_{plcl} is received **and not** **is_written**(Ψ, tx_{plcl})

5: **verify**(tx_{plcl})

6: $tx_{plcl} \leftarrow \text{sign}_i(tx_{plcl})$

7: **if** (**num_signers**(tx_{plcl}) < $\lfloor 2|N_\Psi|/3 \rfloor + 1$) **then**

8: **multicast**(tx_{plcl}) to N_Ψ

9: **else** $r \leftarrow \Gamma.\text{acsend}(\Psi, tx_{plcl})$

▷ get back tx_{plcl} , or tx_{abort}

▷ PHASE 3: $\Gamma.\text{acsend}(\Psi, tx_{plcl})$ generates a response, any p_i can send to Ω

when r is received

10: **if** ($r.type = \text{ABORT}$) **then** $\Gamma.\text{send}(\Omega, r.tx_{abort})$

11: **else if** ($r.type = \text{COMMIT}$) **then** $\Gamma.\text{send}(\Omega, r.tx_{plcl})$

Algorithm 3 Abort procedure.

\triangleright State of the algorithm
 Ω, Ψ, Γ , the blockchain, Platypus blockchain and protocol.
 $\mathbb{C}_S \leftarrow \perp$, the subset of spent coins from \mathbb{C}
 $b_p \leftarrow \perp$, integer s.t. $\Psi[b_p]$ proves some coin was spent
 $vPoF \leftarrow \perp$, proofs-of-fraud of processes
 $tx_{abort} \leftarrow \perp$

```

1: function abort( $tx_{plcl}$ )
2:    $\mathbb{C}_S \leftarrow \text{extract\_spent}(tx_{plcl})$ 
3:    $b_p \leftarrow \text{get\_block\_min\_blockheight}(\mathbb{C}_S)$ 
4:    $vPoF \leftarrow \emptyset$ 
5:   for each  $block$  in  $\Psi[b_p, \dots, -1]$  do
6:      $vPoF.append(\text{processes}(block) \cap \text{processes}(tx_{plcl}))$ 
7:    $tx_{abort} \leftarrow \text{createAbortTx}(tx_{plcl}, b_p, vPoF)$ 
8:    $\Gamma.send(\Omega, tx_{abort})$ 
9:    $\Gamma.send(\Psi, tx_{abort})$ 
10: end function

```

\triangleright all $u_i \in M_\Psi$ run abort when receiving any invalid tx_{plcl}

```

11: when  $tx_{plcl}$  is received
12: if  $\text{coins\_spent}(tx_{plcl})$  then  $\triangleright$  some coins in  $tx_{plcl}$  were spent, invalid
13:   abort( $tx_{plcl}$ )

```

3.4 Correctness & optimal resilience

In this Section, we analyze the correctness of the protocol. To consider its correctness, we must prove that the protocol satisfies all the properties of offchain protocols, as defined in Section 3.2. We start by proving the proper bootstrapping of a Platypus chain, i.e. the adversary never locks the algorithm or gains enough relative power in the committee, in Theorems 3.1 and 3.2, Corollary 3.1 and Lemma 3.1. Then, we prove the properties of offchain protocols when closing a Platypus chain in Theorems 3.3, 3.4, 3.5 and 3.6, and Lemmas 3.2 and 3.3. Following, we prove COMMIT-Privacy/Lightness in Theorem 3.7, and prove that the Platypus protocol solves the offchain problem in Theorem 3.8. Finally, we show that Platypus is resilient-optimal in Theorem 3.9.

Theorem 3.1. Algorithm 1 terminates.

Proof. The algorithm waits for enough Platypus creation signed messages $\{msg_i\}$ from processes (line 11) and to get enough signatures from processes for the Platypus creation transaction (line 17). Since we assume there are at least n_Ψ processes that explicitly state that want to get in Ψ as processes by the bounds t_1 and t_0 , the first condition is met to terminate. That is, an honest process will eventually produce and broadcast a valid Platypus creation transaction with signed $\{msg_i\}$ messages of each of the users that committed to participate in such transaction.

As for the signatures of the tx_{plcr} transaction, notice only h_0 of the n_Ψ are required to sign the

transaction for it to become valid and create the Platypus blockchain Ψ . Since $f < n - h_0$, and only one transaction can be written in Ω , we have that this condition is guaranteed if and only if there are enough signatures from honest processes. Therefore, the protocol terminates. \square

Theorem 3.2. Let Ψ be a Platypus blockchain created by Algorithm 1, and let $u_i \in M_\Omega$ be an honest user. If $u_i \in M_\Psi$ then u_i explicitly stated to be in M_Ψ by sharing a signed Platypus creation message msg_i .

Proof. We prove this by contradiction. Suppose a tx_{plcr} creation transaction such that some coins $Coins_i$ from process u_i are included, without u_i sending a signed Platypus creation message msg_i . Suppose that transaction was written in Ω , creating the Platypus blockchain Ψ . For such transaction to be written in Ω , it must be valid, i.e. it must hold at least h_0 signatures from processes. Since $f < h_0$, at least $h_0 - f$ honest processes signed and verified such transaction (line 16). However, honest processes could not validate such transaction without verifying its content (line 16), which includes verifying all the signed messages from all processes whose coins are involved in tx_{plcr} . Therefore, this is impossible without u_i sending a signed Platypus creation message msg_i . \square

Corollary 3.1. Let Ψ be a Platypus blockchain created by Algorithm 1, and let $u_i \in M_\Omega$ be an honest process. If $u_i \in M_\Psi$ then u_i explicitly stated to be in M_Ψ .

Notice that, in Algorithm 1, the 'only if' direction of Theorem 3.2 and Corollary 3.1 is not necessarily true, should there be more than n_Ψ processes that reply to join. This does not affect the correctness of the protocol though.

Lemma 3.1. Let Ψ be a Platypus blockchain created by Algorithm 1, then its Platypus creation transaction tx_{plcr} has $|N_\Psi| = n_\Psi$ processes and was signed by h_0 of them.

Proof. Given $f < h_0$ and h_0 signatures from distinct processes are required for a Platypus creation transaction to be valid, we have that some honest processes validated it. These honest processes verify that there are n_Ψ processes, and by Theorem 3.2 all processes explicitly stated they wanted to join as processes. Without enough signatures the algorithm does not terminate, since messages keep being sent (line 18), and tx_{plcr} is not yet written in Ω (which is a condition in line 15). By Theorem 3.1 we know that the algorithm terminates. Thus, a valid tx_{plcr} receives h_0 signatures, of which some processes could only have signed if n_Ψ processes were in the transaction as processes. \square

Theorem 3.3. Algorithm 2 guarantees the termination property.

Proof. The protocol only waits for responses 4 times: to get coins from at least h_0 signatures (line 7), and for the transaction to get in the Platypus blockchain and parentchain (lines 9, 10 and 11). All these steps are independent of one another, i.e. not the same processes are required in each step. Therefore, we consider them independently. Since $n - f \geq h_0 \geq 2n_\Psi/3 + 1$, we have that, regardless of what the adversary decides to do, h_0 honest processes will eventually send enough signatures, and coins. Since both the Platypus blockchain and parentchain consensus

protocols tolerate t_ℓ Byzantine faults, the calls that wait for a reply will terminate if $f \leq t_1$ and $f \leq t_0$, thus generating a response in the Platypus protocol (line 9), which could be either a COMMIT or an ABORT. Therefore, an honest process decides COMMIT or ABORT as the result of the call to `acsend(...)` in line 9. In either case, the protocol continues sending the proper transaction to the parentchain (lines 10 and 11), which also terminates. \square

Lemma 3.2. In Algorithm 2, given a bulk close transaction listing a sequence seq that process p_i proposed to COMMIT, either all honest processes of the Platypus chain Ψ decide ABORT to include the transaction in the Platypus blockchain, or all honest processes decide COMMIT.

Proof. We prove this by contradiction. First, notice that, for a process to propose COMMIT on a Platypus bulk close transaction, it is necessary to provide a block where that transaction was written in the Platypus blockchain. We consider the following network partition into three sets: F , the set of the coalition of size $f \leq t_1$, Q_1 and Q_2 . We consider that, at some point, all processes in Q_1 signed a block b_1 to validate a Platypus bulk close transaction, whereas processes in Q_2 validated a different block b_2 that spent from one of the same outputs (conflicting transactions). For one honest process to propose COMMIT, it is necessary that b_1 was validated by at least $h_0 \geq 2n_\Psi/3 + 1$ processes. Analogously, for one process to propose ABORT, it has to provide valid proof through a block b_2 validated by at least $h_0 \geq 2n_\Psi/3 + 1$ processes, in which some coins were spent from the owners claimed in the Platypus bulk close. A COMMIT proposal is undecided for as long as a valid ABORT is proposed, or enough processes validate the COMMIT attempt.

In this case, we consider that one honest process proposes ABORT, meaning that it has a valid abort transaction, i.e. b_2 was validated by at least $2n_\Psi/3 + 1$ processes. Therefore, $|Q_2 \cup F| \geq 2n_\Psi/3 + 1$. However, if another honest process committed to block b_1 , then block b_1 has $2n_\Psi/3 + 1$ processes. Thus, $|Q_1 \cup F| \geq 2n_\Psi/3 + 1$. Recall that $|F| = f \leq t_1$ and therefore $|Q_1| > t_1$ and $|Q_2| > t_1$, but this is impossible since $F \cup Q_1 \cup Q_2 = N_\Psi$ and $Q_1 \cap Q_2 = Q_1 \cap F = F \cap Q_2 = \emptyset$, and each account is only used once. It follows that only Q_2 or only Q_1 had enough processes, and thus there are two possible outcomes: either some honest processes propose and decide ABORT (after which all will decide ABORT), or instead some processes decide COMMIT (leading all other processes to decide COMMIT once they update their view of the childchain, since they do not decide ABORT). \square

Lemma 3.3. A Platypus bulk close transaction (COMMIT) can only be valid in Ω if it is already written in Ψ .

Proof. For this, we assume that the transaction is sent to the parentchain without it being fully signed (i.e. beyond the threshold h_0) in the Platypus chain. A Byzantine process can try to send directly to the parentchain a not fully signed Platypus bulk close transaction (i.e. a Platypus bulk close transaction that was not written in the Platypus blockchain). However, this transaction is not valid in the parentchain until it receives enough signatures. Notice that any process in the parentchain (i.e. Platypus blockchain processes too) can eventually see this transaction, and generate a valid ABORT proof, or try to get it written in the Platypus

blockchain and then generate a valid COMMIT. Therefore, this proof is analogous to that of Lemma 3.2. \square

Theorem 3.4. Algorithm 2 guarantees the agreement property.

Proof. By Lemma 3.3 we know that all COMMIT decisions are firstly written in Ψ . Then, Lemma 3.2 shows that all processes in M_Ψ reach the same decision to write in Ψ . We only have left the case that an ABORT is decided without it being written in the Platypus blockchain Ψ . We need to prove that if that ABORT is decided then no process decided COMMIT. An ABORT outside of Ψ can only happen if a process p_i tried to COMMIT directly to Ω a Bulk close transaction that is not valid. Then, another process p_j generated a valid proof of fraud included in an abort transaction, that ended up in an ABORT decision. Analogously to the proof of Lemma 3.2, we have a valid proof of fraud that gathers at least one conflicting transaction written in a previous block in Ψ , and therefore validated by at least h_0 processes. With the same approach used in Lemma 3.2, it is possible to prove that it is not possible for p_j to propose a valid ABORT if one honest process p_i decided COMMIT. Once a COMMIT is decided by enough processes, the funds go back to the blockchain in the bulk close transaction of the sequence committed. Therefore, another sequence in another bulk close transaction will not be COMMIT-decided by any honest process. Hence, the agreement property is guaranteed. \square

Theorem 3.5. Algorithm 2 guarantees the COMMIT-validity property.

Proof. Lemma 3.3 shows that the only way to get something committed is to first write it in Ψ , while Lemma 3.2 proves that either all or no honest process decide COMMIT on a sequence. If no honest process proposes ABORT and, by Theorem 3.3, they guarantee termination, then they must COMMIT. \square

Theorem 3.6. Algorithm 2 guarantees the ABORT-validity property.

Proof. If an honest process proposes ABORT in Ψ , then by Lemma 3.2 all honest processes decide ABORT. All honest processes in Ψ also agree on an ABORT generated to a COMMIT outside of Ψ , as already shown in the proof of the agreement property (Theorem 3.4). \square

Theorem 3.7. Algorithm 2 guarantees the COMMIT-Privacy/Lightness property.

Proof. First, we consider the case that a Platypus bulk close transaction was successfully written in the parentchain (i.e. a COMMIT). W.l.o.g. we assume this to be the second transaction (after the Platypus creation transaction) to be written in the parentchain relating this Platypus chain Ψ , i.e. that no previous abort transactions were written. Let δ_c be the time when the Platypus chain was created, δ_d the time when the Platypus chain was closed. This Platypus bulk close transaction has been validated in the Platypus blockchain Ψ , verifying all the operations were correct. The parentchain processes that are not in the Platypus chain have no knowledge of the

Platypus chain other than its Platypus creation transaction that was written in Ω . Therefore, a Platypus bulk close transaction with enough signatures from processes, and valid signatures, seems correct from the point of view of Ω . Therefore, only this information, along with the list of coins and owners, is provided to the parentchain. This means that parentchain processes only stored the list of owners and coins at δ_c , and received a different list of owners and their coins at δ_d . They can tell which coins changed ownership between δ_c and δ_d , but they cannot tell if there were more owners in between. Thus, they can only see the minimal transfer set.

Whereas the COMMIT-Privacy/Lightness property considers COMMITs, if abort transactions took place in between δ_c and δ_d , a few more operations might be revealed to the parentchain to prove invalidity in abort transactions. However, changing δ_c to the time of the last ABORT, the proof remains valid. \square

Theorem 3.8 (Correctness). The Platypus Protocol solves the offchain problem.

Proof. The proofs for Algorithm 1 guarantee that n_Ψ processes are requested at all times (Lemma 3.1), all of which explicitly stated to participate as processes (Corollary 3.1), with guaranteed termination if there are enough processes n_Ψ (Theorem 3.1), i.e. the Platypus chain is properly bootstrapped and the security assumptions remain at the end of Algorithm 1. Once this bootstrapping takes place, the inner consensus of Ψ guarantees the consensus properties given the assumption $f \leq t_1$, with the same set n_Ψ of processes and using the same h_0 as threshold for Byzantine behavior. Finally, given this bootstrapping and consensus protocol, we showed above that Algorithm 2, which closes the Platypus chain, guarantees termination, agreement, ABORT-validity, COMMIT-validity and COMMIT-lightness/privacy. Therefore, Platypus solves the offchain problem. \square

The following theorem shows that our construction works in the strongest coalition the adversary can form.

Theorem 3.9 (Optimal resilience). It is impossible to perform a transfer operation in a (consensus-based) offchain protocol with partial synchrony if $f > t_1$ or $f > t_0$.

Proof. If $f > t_0$ then the adversary can corrupt the consensus protocol [13], and thus the offchain protocol would not be correct. Hence, suppose $f > t_1$ while still $f \leq t_0$. Let there be at least one coin \mathfrak{c} transferred from account z_a to account z_b in transaction tx in the offchain protocol Γ (i.e. not the trivial case of closing after opening), and also suppose there is another transaction tx' that transfers the same coin \mathfrak{c} from account z_a to z_c , $z_c \neq z_b$. It is clear that only one can be decided.

We proceed by contradiction. We assume that there exists a correct offchain protocol that decides to perform a transfer operation. Consider a partition of processes N_Ψ into three disjoint sets Q_1, Q_2, Q_3 , with each of them containing between 1 and f processes. First consider the following scenario A: processes in Q_1 and Q_3 are honest and propose to perform transaction tx , and processes in Q_2 are Byzantine. It follows that $Q_1 \cup Q_3$ must decide tx at some time T_A , for if they did not decide tx there would be a scenario in which processes in Q_2 are honest and also propose to decide tx , but messages sent from processes in Q_2 are delivered at a time

greater than T_A , having processes in $Q_1 \cup Q_3$ already decided abort. This would break the COMMIT-validity property.

Consider now scenario B: processes in Q_1 are Byzantine, and processes in Q_2 and Q_3 are honest and propose to perform transaction tx' . By the same approach, $Q_3 \cup Q_2$ decide to perform tx' at a time T_B .

Now consider scenario C: processes in Q_1 and Q_2 are honest, and processes in Q_3 are Byzantine, the messages sent from processes in Q_1 are delivered by processes in Q_2 at a time greater than $\max(T_A, T_B)$, and the same for messages sent from processes in Q_2 to processes in Q_1 . Then, for processes in Q_1 this scenario is identical to scenario A, deciding to perform tx , while for processes in Q_2 this is identical to scenario B, deciding to perform tx' , which leads to a disagreement. This yields a contradiction. \square

3.5 Theoretical analysis

In this section, we analyze the communication, message and time complexity of the Platypus protocol, ignoring the complexity of the underlying blockchain. We consider the calls to $\text{acsend}(\Psi, tx)$ and $\text{send}(\{\Psi, \Omega\}, tx)$ to have the same complexities as one multicast to all processes $N_{\{\Psi, \Omega\}}$ of the blockchain that receives the transaction tx .

Message complexity. The message complexity of Algorithms 2, 4 and 5 is $\mathcal{O}(n_\Psi^2)$ and that of Algorithm 3 is $\mathcal{O}(m_\Psi \cdot n_\Psi)$. We conjecture that the complexity could however be reduced to $\mathcal{O}(n_\Psi)$ at some points, leveraging non-interactive aggregation of the processes signatures and messages, but certain calls to $\text{acsend}(\dots)$ and $\text{send}(\dots)$ would still have a complexity of $\mathcal{O}(n_\Psi^2)$, as they can be executed by all processes. The same applies to Algorithm 1, with the exception that Phase 2 has a message complexity of $\mathcal{O}(m_\Psi \cdot m_\Omega)$, thus being the complexity of Platypus.

Communication complexity. The message size is $\mathcal{O}(m_\Psi)$ in lines 18 and 14 of Algorithm 1, leading to a communication complexity of $\mathcal{O}(\max\{m_\Psi \cdot m_\Omega, n_\Psi^3\})$, because of phases 2 and 3 of the algorithm. Line 7 of Algorithm 3 also has a message size of $\mathcal{O}(n_\Psi)$, leading to a communication complexity of $\mathcal{O}(m_\Psi \cdot n_\Psi^2)$, although the set of processes can be removed if no punishments are considered. The rest of messages have constant size in all algorithms, thus their communication complexity is the same as their message complexity.

Time complexity. The time complexity is $\mathcal{O}(n_\Psi)$ due to Phase 4 of Algorithm 1 and Phase 2 of Algorithm 2. Algorithms 3, 4 and 5 have constant time complexity. Again, we conjecture that, leveraging non-interactive aggregation, the time complexity can be reduced to constant time.

3.6 Improvements & discussion

In this section, we consider additional features of the Platypus chain, and its usage for the general sidechains problem, which we also define.

3.6.1 Crosschain payments

A crosschain payment can be of two types, either a payment to a parentchain, or a payment through a parentchain to another childchain. With the above-shown protocol, a payment to a parentchain would require a Platypus bulk close transaction, and a new Platypus creation transaction. We describe an extension of the protocol to perform payments without closing and reopening Platypus chains.

3.6.1.1 Users' splice-in & splice-outs

Splice-in and Splice-out transactions allow users to get their funds into and out of the Platypus chain, respectively.

– *Splice in.* Splicing in allows users to join a Platypus chain. Algorithm 4 shows the Splice in protocol for a process p_i that wants to join Ψ . A splice in transaction tx_{spin} must be written in both Ω and Ψ , after which the funds can only be spent in Ψ . Since this transaction takes place after the Platypus chain has been created, it requires some validation by both sets of processes.

Algorithm 4 Splice in algorithm for process p_i .

▷ State of the algorithm
 Ω, Ψ, Γ , the blockchain, Platypus blockchain and protocol
 \mathbb{C}_i , coins that belong to process p_i
 $plid$, the Platypus chain identifier
 $tx_{spin} \leftarrow \perp$, the splice in transaction

▷ p_i creates and waits for transaction to write
1: $tx_{spin} \leftarrow \text{createSpliceInTx}(\mathbb{C}_i, plid)$
2: $tx_{spin} \leftarrow \text{sign}_i(tx_{spin})$
3: $\Gamma.\text{send}(\Omega, tx_{spin})$
4: $\Gamma.\text{send}(\Psi, tx_{spin})$

– *Splice out.* The same way users can splice into an existing Platypus chain, they can get their funds back in the parentchain. Again, this is a sensible operation that requires proper synchronization between both Platypus chain and parentchain so as to protect against fraud.

The splice out transaction allows processes to leave a Platypus chain before it is closed, retrieving their funds back in the parentchain. In this case, we require first the transaction to be finalized in Ψ before being considered for the parentchain. Algorithm 5 shows the splice out protocol for a process p_i . This protocol is rather a simplification of Algorithm 2. It creates and tries to write a splice out transaction tx_{spou} , that can be aborted with an abort transaction tx_{abort} .

3.6.1.2 Processes' splice-in & splice-outs

Notice that the adversary could gain enough relative power either by splicing in or by honest processes splicing out. One way to prevent this is by keeping the set of processes intact regardless

Algorithm 5 splice out for process p_i .

\triangleright State of the algorithm
 Ω, Ψ, Γ , the blockchain, Platypus blockchain and protocol
 \mathbb{C}_i , the coins that belong to process p_i
 $tx_{spou} \leftarrow \perp$, the splice out transaction

$\triangleright p_i$ creates and waits for transaction to write in Ψ

```

1:  $tx_{spou} \leftarrow \text{createSpliceOutTx}(\mathbb{C}_i)$ 
2:  $tx_{spou} \leftarrow \text{sign}_i(tx_{spou})$ 
3:  $r \leftarrow \Gamma.\text{acsend}(\Psi, tx_{spou})$   $\triangleright$  get back  $tx_{spou}$  or  $tx_{abort}$ 
4: if ( $r.type = \text{ABORT}$ ) then  $\Gamma.\text{send}(\Omega, r.tx_{abort})$ 
5: else if ( $r.type = \text{COMMIT}$ ) then  $\Gamma.\text{send}(\Omega, r.tx_{spou})$ 
  
```

of the funds each process has after Platypus creation. An additional feature of the protocol might provide explicit delegation of the process set to other users by means of a committee sortition protocol (Section 2.2.8). We explore further a protocol for committee sortition in Chapter 6.

Another alternative may allow users and the set of processes to splice in and splice out at will. In this set, processes should take great care at identifying the probability of an adversary gaining enough relative power depending on the protocol that sorts the committee (Section 2.1.5). If the probability of an adversary gaining enough relative power reaches a certain threat threshold, either by the set of processes reducing significantly, based on the funds at stake or any other information used for heuristics, processes can generate a Platypus bulk close transaction and safeguard all users' funds. In practice, this information is probably based on heuristics (e.g. total stake left, percentage of stake held by one account, etc.). From a theoretical perspective, this variation requires the assumption that the adversary never gains enough relative stake such that $\text{stake}(f) \geq \text{stake}(n_\Psi)/3$.

3.6.1.3 Crosschain payments with splice-in & splice-outs

A crosschain payment in between two blockchains with Platypus is a payment of one user from/into an existing Platypus chain to/from its parentchain, or in between two Platypus chains that share a common parentchain. In Section 3.6.2, we generalize such definition. Regardless of the particular conditions and assumptions for splice-ins and splice-outs, we illustrate in this section how these transactions would work.

* *Crosschain payment from/to parentchain.* This case is trivial using Algorithm 4 or 5, respectively.

* *Crosschain payment between childchains.* This is performed with a splice out into the common parentchain, followed by a splice in into the recipient.

3.6.2 Platypus for sidechains

The childchain definition from Section 3.2 can easily be generalized for sidechains by clearly decoupling Ψ from the protocol, and stating different committees $P \neq Q$ instead of $P \subsetneq Q$. We define sidechain protocols as a superset of offchain protocols, defined in Section 3.2. A sidechain

protocol facilitates a payment across blockchains, i.e. a *crosschain payment*. If two or more blockchains intend to perform crosschain payments, we refer to them as being sidechains. They may or may not be in a parent-child hierarchy.

* *Sidechain protocol*. Given two blockchains, Ω of P processes and Ψ of Q , $P \neq Q$ a sidechain protocol Π is an offchain protocol that enables transfers in between all accounts z_q, z_p such that $\gamma(z_q) = q \in Q, \gamma(z_p) = p \in P$. To reflect Ω and Ψ being independent, and this possibility of transferring, we define the following property:

– COMMIT-Matching Knowledge: If an honest process decides COMMIT on a sequence seq of transfer operations in Π between Ω and Ψ , then $\forall p \in P$, p knows a subset seq_1 and $\forall q \in Q$, q knows a subset seq_2 , such that seq_1 and seq_2 are two minimal transfer sets, $seq_2 \cap seq_1 = \emptyset$, and it exists one surjective application $f : seq_1 \times seq_2 \rightarrow TR^-(seq_1 \cup seq_2) \cup \{0\}$ defined as follows:

$$f(z_a TR z_b, z_c TR z_d) = \begin{cases} (z_a TR z_d) & \text{if } \gamma(z_b) = \gamma(z_c) \\ 0 & \text{otherwise} \end{cases} \quad (3.1)$$

Also, since the coins are different in different blockchains, we identify coins by their value when calculating the minimal transfer set TR^- . Intuitively, for a transaction in seq_1 exists a transaction in seq_2 such that both are transitive (that is, the receiver of one is the sender of the other). If that was not to happen, some of the transactions in seq_1 , or in seq_2 , would have nothing to do with a payment in between two sidechains.

If $Q \subsetneq P$ then seq_1 is just a set of idempotent transfers of the form $z_a TR z_b$, with $\gamma(z_a) = \gamma(z_b)$, since all $p \in Q$ are also in P , and thus COMMIT-Privacy/Lightness is a particular case scenario of the COMMIT-Matching Knowledge property.

Similarly, if $P \not\supseteq Q$, $P \not\supseteq Q$ then Ω and Ψ are not in the parent-child chain hierarchy.

* *Crosschain payments*. This is solved by our protocol if both sidechains have a common parentchain, as shown in Section 3.6.1.3. In general, for a crosschain payment between two unrelated blockchains Ω_1 and Ω_2 , with sets of processes N_{Ω_1} and N_{Ω_2} , they can perform the payment manufacturing an additional blockchain Φ :

– Create a common parentchain Φ with $N_\Phi \supseteq N_{\Omega_1} \cup N_{\Omega_2}$, extend both their blockchains to adopt the Platypus protocol, and perform the payment as explained in Section 3.6.1.3. In this case, if the adversary tries to double spend the crosschain payment in Ω_2 , or in Ω_1 , then, as long as $f < |N_\Phi|/3$, the funds will remain in the parentchain Φ .

– Create a common Platypus chain Φ , with $N_\Phi \subseteq N_{\Omega_1} \cap N_{\Omega_2}$, and perform the payment. In such a case, should $f < |N_{\Omega_1}|$ and $f < |N_{\Omega_2}|$, then the adversary could not double spend the funds in Φ and splice out to both Ω_1 and Ω_2 .

3.6.3 Attacks

Many of the common attacks for synchronous offchain protocols are not applicable in the partially synchronous Platypus [82, 91, 83]. Theorem 3.9 shows how if the adversary is such that $f > t_1$ then it can perform a colluding processes attack. We also introduce the *ABA-transfer* attack. If a coin \mathfrak{c} was transferred from process p to process q , and later on again to process p , q can try to ABORT any close/splice out in which \mathfrak{c} does not belong to him, by using as

proof the deprecated transfer. To cope with this attack, we use session keys in this document, as mentioned in Section 3.2, thus having two different accounts. Another possible solution involves committing to merkle trees and requiring any ABORT to provide a merkle tree T such that the merkle tree T' of the COMMIT attempt is included $T' \subseteq T$ as part of the PoF.

3.7 Summary

In this chapter, we outlined the problem of synchrony for layer-2 protocols, in that assuming synchrony for layer-2 derives in a trade-off between security and scalability. We further exemplified this trade-off in a novel attack, the Lockdown attack. We showed that the Lockdown attack is present in a variety of synchronous offchain protocols, from payment channels, factories, networks to even childchains. Then, we presented a formalization of the offchain problem, as well as of the childchain problem and the sidechains problem. Following, we presented Platypus, a novel offchain protocol without synchrony.

Platypus solves the childchain problem, and can also be used to solve the sidechains problem. We proved the correctness of Platypus, and its optimal resilience, and showed its message, bit and time complexities. Platypus lays the first stone towards obtaining both security and scalability solving consensus without synchrony. In Chapter 4, we continue this line of work by focusing on designing protocols for consensus that increase the traditional tolerance to faults by considering rational behavior.

Chapter 4

Rationality for Blockchains’ Consensus

In Chapter 3, we explored a general solution that could be attached to any blockchain to increase scalability without sacrificing security. The Platypus protocol allows blockchains to be extended via a childchain, and to perform secure payments between sidechains. We further justified the need for scalable solutions without synchrony by outlining attacks and performing a novel attack, known as the Lockdown attack.

However, Platypus is a protocol that ensures scalability and security via transfers between two blockchains, but only provided these blockchains ensure the same level of security. There are a number of consensus protocols that developed into blockchains with the well-known resilient-optimal results of at most t_ℓ Byzantine faults without synchrony (recall that we defined $t_\ell = \lceil n/3 \rceil - 1$ in Section 2.2.1). Unfortunately, this t_ℓ bound is constraining blockchains, where participants’ decisions on assets can incur large sums of assets being stolen anonymously. For example, in 2020, these assets incentivized players to deviate from their blockchain protocol by forcing a disagreement to fork the blockchain, leading to a double-spending of US\$70,000¹ and US\$18² million in Bitcoin Gold and US\$5.6 million in Ethereum Classic³.

Interestingly, some blockchains already require participants to deposit cryptocurrency assets in the form of staking, which could be used by the consensus protocol to disincentivize misbehavior (Section 2.1.4). Unfortunately, these blockchains do not prove the correctness of their protocols in the presence of rational players, only arguing that their incentives should suffice. Furthermore, previous works did not explore the intrinsic characteristics of blockchains in order to propose blockchain-specific consensus protocols that would allow for greater tolerance to coalitions trying to double spend, perhaps due to the complexity of this problem, that requires a mixture of knowledge from distributed systems, game theory and cryptography. In this chapter, we successfully address this challenge to propose the first consensus protocol without synchrony that tolerates up to less than half of the participants colluding together, by reusing the recent advances in cryptography that allowed for accountability in consensus, and

¹<https://news.bitcoin.com/bitcoin-gold-51-attacked-network-loses-70000-in-double-spends/>

²<https://news.bitcoin.com/bitcoin-gold-hacked-for-18-million/>

³<https://news.bitcoin.com/5-6-million-stolen-as-etc-team-finally-acknowledge-the-51-attack-on-network/>

considering the rationality of players alongside faulty players.

Considering rational players alongside the well-known Byzantine faults to ensure the agreement property of the consensus problem allows us to evaluate protocols against coalitions of players that are incentivized to deviate for their own benefit or to break the system, rather than just because of their faulty nature.

Our result. In this chapter, we show that a *baiting strategy*, that incentivizes rational players to bait deviating players into a trap, is necessary and sufficient to solve this *rational agreement* problem by offering a consensus protocol that is robust to a coalition of up to k rational players and t Byzantine faults. Our first contribution is thus to formalize the notion of a baiting strategy and show that this baiting strategy is necessary to solve the rational agreement problem.

We implement this solution in a new protocol, called TRAP (**T**ackling **R**ational **A**greement through **P**ersuasion), that bypasses the requirement of $2n/3$ honest participants [11], by solving consensus when $n > \max(\frac{3}{2}k + 3t, 2(k + t))$. For example if $n = 7$ players, then our solution solves consensus with only 4 honest participants, hence tolerating $k = 1$ rational player and $t = 2$ Byzantine faults. TRAP rewards a single player to expose its coalition by generating PoFs thanks to building upon an accountable consensus protocol (such as Polygraph [47]).

Making initially colluding players decide on whether to betray the coalition regardless of the behavior of the rest is analogous to reducing the extensive-form game into a normal-form game for this particular decision (where a normal-form game defines a game in which all participants must decide without before knowing what the rest decide). In addition, if the reward for exposing the coalition is greater than the individual payoff for causing a disagreement, rational players find that the colluding strategy is strictly dominated by the baiting strategy in the extensive-form game. This shows similarities with the well-known prisoner's dilemma, that is, a game in which all rational players prefer to betray the coalition than to collude, even if cooperation benefits the sum of the gains of the coalition.

More specifically, our protocol “pre-decides” the decisions from an accountable consensus protocol (i.e. Polygraph [47]) that it extends with the *Byzantine Fault-Tolerant Commit-Reveal protocol (BFTCR)*, which consists of two reliable broadcasts and one additional broadcast. As we show in Figure 4.1, by offering a reward the TRAP protocol can convince e rational players to betray the coalition after they helped cause a disagreement on predecisions. First, the coalition exposes itself causing a disagreement on predecisions. Second, the e rational players from the coalition that decide to betray are enough to pause termination of the BFTCR protocol. Third, these players can wait to gather enough PoFs of the disagreement on predecisions, which they will get by the property of accountability (since they caused a disagreement on the output of the Polygraph protocol). Fourth and finally, once they get PoFs to prove the disagreement to honest players and get the reward, these e players commit and reveal the PoFs by sharing them during the BFTCR protocol, after which one will be selected at random to get the reward. We defer further details of this example to Appendix B (Section B.1).

Adding this BFTCR phase ensures the existence of a baiting strategy (*baiting dominance*) and that the protocol still solves agreement even after playing the baiting strategy (*baiting agreement*). We also add an additional property, *lossfree reward*, which states that the increase

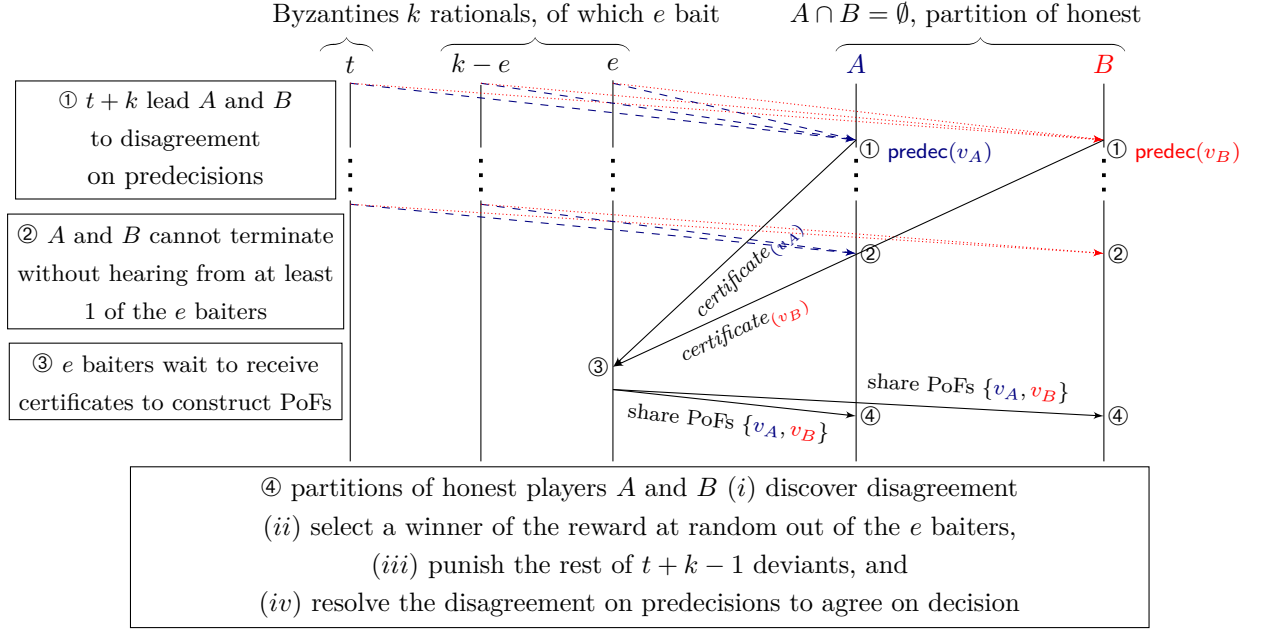


Figure 4.1: Example execution of the TRAP protocol. First, ① all t Byzantine and k rational players collude to cause a disagreement on the output of the accountable consensus protocol, resulting in A and B predeciding different outputs. Then, ② e of the k rational players decide to bait while executing the BFTCR protocol, preventing A and B from deciding their disagreeing predecisions. As such, ③ the e baiters wait until they receive proof of the disagreement on predecisions, to then ④ prove the disagreement by committing to and revealing the proofs-of-fraud in the BFTCR protocol. Hence, neither A nor B decide their conflicting predecisions, but instead reward one of the e baiters, punish the rest of $t + k - 1$ players responsible for the disagreement on predecisions, and resolve the disagreement, deciding one of v_A or v_B , or, depending on the application, merging both.

in utility for baiting rational players comes at no cost to non-deviating players. For this purpose, we introduce a deposit per player, so that the system can always pay the reward by taking the deposits of the proven coalition at no cost for non-deviating players.

Summary. In summary, the work presented in this chapter is:

- i) We adapt previous rational models to the consensus problem in partial synchrony, and to the crash fault model.
- ii) We model baiting strategies as a new type of punishment strategies, and show their relevance to the consensus problem.
- iii) We extend the well-known impossibility proof of consensus in partial synchrony in the Byzantine fault model [13] to show that it is impossible to solve consensus without a baiting strategy in the presence of k rational players and t Byzantine faults for $k + t > t_\ell$, unless there is a baiting strategy.
- iv) We extend the well-known impossibility proof of consensus in partial synchrony in the crash fault model [13] to show that it is impossible to solve consensus without a baiting strategy in the presence of k rational players and t crash faults for $k + 2t \geq n$.
- v) We present TRAP, a protocol that solves consensus in the presence of k rational players and t Byzantine faults for $n > \max(\frac{3}{2}k + 3t, 2(k + t))$.
- vi) We establish a parametrizable, bidirectional relation between tolerating rational players with Byzantine faults and with crash faults, concluding that our TRAP protocol also solves consensus in the presence of $k + t$ rational players and t crash faults for the same values of k and t .

Chapter outline. In Section 4.1 we present our rational model. Section 4.2 presents the formalization of baiting strategies and the new impossibility results. We present the TRAP protocol and its correctness in Section 4.3. Section 4.4 establishes a parametrizable, bidirectional relation between protocols that tolerate crash faults and protocols that tolerate Byzantine faults in the presence of rational players. Finally, we conclude the chapter in Section 4.5.

4.1 Rational model

We adapt the synchronous and asynchronous models of Abraham et al. [71, 72] to our partially synchronous model. In this chapter, we speak of players instead of processes, and speak of player $i \in [n]$, instead of process p_i .

We consider a game played by a set N of $|N| = n$ players, each of type in $\mathcal{T} = \{\text{Byzantine}, \text{rational}, \text{honest}\}$. The game is in *extensive form*, described by a game tree whose leaves are labeled by the utilities u_i of each player i . We introduce the *scheduler* as an additional player that will model the delay on messages derived from partial synchrony.

We assume that players alternate making moves with the scheduler: first the scheduler moves, then a player moves, then the scheduler moves, and so on. The scheduler's move consists

of choosing a player i to move next and a set of messages in transit to i that will be delivered just before i moves (so that i 's move can depend on all the messages i delivers). Every non-leaf node is associated with either a player or the scheduler. The scheduler is bound to two constraints. First, the scheduler can choose to delay any message msg up to a bound, known only to the scheduler, before which he must have chosen all recipients of msg to move and provided them with this message, so that they deliver it before making a move. Second, the scheduler must eventually choose all players that are still playing. That is, if player i is playing at time x , then i is chosen to play at time $x' \geq x$.

Each player i has some *local state* at each node, which translates into the initial information known by i , the messages i sent and received at the time that i moves, and the moves that i has made. The nodes where a player i moves are further partitioned into *information sets*, which are sets of nodes in the game tree that contain the same local state for the same player i , in that i cannot distinguish them. We assume that the scheduler has complete information, so that the scheduler's information sets consist of the singletons.

Since we do not assume synchrony, we need our game to be able to continue even if a faulty player decides not to reply. As such, w.l.o.g. we assume that players that decide not to play will at least play the *default-move*, which consists of notifying the scheduler that this player will not move, so that the game continues with the scheduler choosing the next player to move. Thus, in every node where the scheduler is to play a move, the scheduler can play any move that combines a player and a subset of messages that such player can deliver before playing. Then, the selected player moves, after which the scheduler selects again the next player for the next node, and the messages it receives, and so on. The scheduler alternates thus with one player at each node down a path in the game tree until reaching a leaf. A *run* of the game is then a downward path in the tree from the root to a leaf.

Strategies. We denote the set of actions of a player i (or the scheduler) as A_i (or A_s), and a strategy σ_i for that set of actions is denoted as a function from i 's information sets to a distribution over the actions. We denote the set of all possible strategies of player i as \mathcal{S}_i . Let $\mathcal{S}_I = \prod_{i \in I} \mathcal{S}_i$ and $A_I = \prod_{i \in I} A_i$ for a subset $I \subseteq N$. Let $\mathcal{S} = \mathcal{S}_N$ with $A_{-I} = \prod_{i \notin I} A_i$ and $\mathcal{S}_{-I} = \prod_{i \notin I} \mathcal{S}_i$. A *joint strategy* $\vec{\sigma} = (\sigma_0, \sigma_1, \dots, \sigma_{n-1})$ draws thus a distribution over paths in the game tree (given the scheduler's strategy σ_s), where $u_i(\vec{\sigma}, \sigma_s)$ is player i 's expected utility if $\vec{\sigma}$ is played along with a strategy for the scheduler σ_s . A strategy θ_i *strictly dominates* τ_i for i if for all $\vec{\phi}_{-i} \in \mathcal{S}_{-i}$ and all strategies σ_s of the scheduler we have $u_i(\theta_i, \vec{\phi}_{-i}, \sigma_s) > u_i(\tau_i, \vec{\phi}_{-i}, \sigma_s)$.

A *protocol* in this model is thus the recommended joint strategy $\vec{\sigma}$ whose outcome satisfies the functionality \mathcal{F} for all strategies σ_s of the scheduler, and an *associated game* Γ for that protocol is defined as all possible deviations from the protocol [71]. Note that both the scheduler and the players can use probabilistic strategies.

Failure model. k players out of n can be rational and up to t can be faulty (either all Byzantine or all crash); while the rest of the players are honest. *Honest players* follow the protocol: the expected utility of honest player i is greater than 0 for any run in which the outcome satisfies consensus and they have followed the protocol, and 0 for any other run. *Rational players* can deviate to follow the strategy that yields them the greatest expected utility at any time they are

to move, while *Byzantine players* can deviate in any way, even not replying at all (apart from notifying the scheduler that they will not move). Rational players prefer to terminate and to guarantee validity, but may have an interest in preventing agreement. That is, rational players may see their expected payoff increased if they have a chance at causing a double-spending attack without any punishment associated to it (if, for example, no accountability is ensured by the executed consensus protocol). If the protocol manages to incentivize agreement beyond the benefit from a disagreement attack (via rewards or via punishments if caught through accountability), then rational players prefer to ensure agreement as well. We will detail further the utilities of rational players in Section 4.3.3. A *crash player* i behaves exactly as an honest player, except that it can crash at any time of any run. If i crashes at time x of run y , then it may send a message to some subset of players at time x , but from then on, it sends no further messages (except for playing the default-move).

Bounded gain. We assume that if a coalition manages to cause a disagreement, then it obtains a payoff of at most \mathfrak{G} , which we call the *total gain*. Nevertheless, this total gain may be, for example, the entire market value of the system. In a payment system application in which players agree on a set of transactions to be decided (e.g. blockchains), the total gain \mathfrak{G} is exactly the sum of all the amounts spent in all transactions of a block. We also assume, w.l.o.g., that a coalition with k rational players and t Byzantine players split equally the total gain into k parts, which we call the *gain* $g = \mathfrak{G}/k$. That is, Byzantine players are willing to give all the total gain from causing a disagreement to the rational players that collude (to incentivize the deviation for these rational players). Note that a protocol that tolerates a maximum gain \mathfrak{G} equally split into k parts also tolerates any gain such that the maximum share of the split is \mathfrak{G}/k , but we assume the equal split for ease of exposition.

Disagreements. We speak of the *disagreeing* strategy as the strategy in which players collude to produce a disagreement, and of a coalition *disagreeing* to refer to a coalition that plays the disagreeing strategy. A disagreement of consensus can mean two or more disjoint groups of non-deviating players deciding two or more separate, conflicting decisions [46]. For ease of exposition, we consider in this work only disagreements into two values. Nonetheless, if the size of the coalition is less than half the total number of players $k + t < n/2$ (as is the case for the work that we present) then the coalition can only cause a disagreement into two values [46], whereas greater sizes of a coalition can cause disagreements into multiple values, as we show in Chapter 5.

We let rational players in a coalition and Byzantine players (in or outside the coalition) know the types of all players, so that they know which players are the other faulty players, rational players and honest players, while the rest of the players only know the upper bounds on the number of rational and faulty players, i.e., k and t respectively, and their own individual type (that is, whether they are rational, Byzantine, crash or honest).

Communication cost. As we are in a fully distributed system, without a trusted central entity like a mediator, we assume *cheap-talks*, that is, private pairwise communication channels which incur a negligible communication cost. Honest players are also only interested in reaching consensus, and not in the number of messages exchanged. Similarly, we assume the cost of

performing local computations (such as validating proposals, or verifying signatures) to be negligible.

Cryptography. We require the use of standard cryptography, for which we reuse the assumptions of Goldreich et al. [179]: polynomially bounded players (see Section 2.2.5) and the enhanced trapdoor permutations. In practice, these two assumptions mean that players can sign unforgeable messages, and that they can perform oblivious transfer. Each player has a public key and a private key, and public keys are common knowledge (see Section 2.2.3).

4.1.1 Robustness

Given that a Nash equilibrium [180] only protects against single-player deviations, and our distributed system may be susceptible of a coalition of k rational and t Byzantine players, it is important to consider tolerating multi-player deviations. We thus restate Abraham's et al. [71] definitions of t -immunity, ϵ -(k, t)-robustness and the most recent definition of k -resilient equilibrium [72]. The notion of k -resilience is motivated in distributed computing by the need to tolerate a coalition of k rational players that can all coordinate actions. A joint strategy is k -resilient if not all rational members of a coalition of size at most k can gain greater utility by deviating in a coordinated way.

Definition 4.1.1 (k -resilient equilibrium). A joint strategy $\vec{\sigma} \in \mathcal{S}$ is a k -resilient equilibrium (resp. strongly k -resilient equilibrium) if, for all $K \subseteq N$ with $|K| \leq k$, all $\vec{\tau}_K \in \mathcal{S}_K$, all strategies σ_s of the scheduler, and for some (resp. all) $i \in K$ we have $u_i(\vec{\sigma}_K, \vec{\sigma}_{-K}, \sigma_s) \geq u_i(\vec{\tau}_K, \vec{\sigma}_{-K}, \sigma_s)$.

The notion of t -immunity is motivated in distributed algorithms by the need to tolerate t Byzantine players. An equilibrium $\vec{\sigma}$ is t -immune if non-Byzantine players still prefer to follow $\vec{\sigma}$ despite the deviations of up to t Byzantine players.

Definition 4.1.2 (t -immunity). A joint strategy $\vec{\sigma} \in \mathcal{S}$ is t -immune if, for all $T \subseteq N$ with $|T| \leq t$, all $\vec{\tau} \in \mathcal{S}_T$, all $i \notin T$ and all strategies of the scheduler σ_s , we have $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T, \sigma_s) \geq u_i(\vec{\sigma}, \sigma_s)$.

A joint strategy is an ϵ -(k, t)-robust equilibrium if no coalition of k rational players can coordinate to increase their expected utility by ϵ regardless of the arbitrary behavior of up to t Byzantine players, even if the Byzantine players join their coalition. We illustrate it however with ϵ because of the use of cryptography, that is, in order to account for the (negligible) probability of the coalition breaking cryptography, as was done previously [71]:

Definition 4.1.3 (ϵ -(k, t)-robust equilibrium). A joint strategy $\vec{\sigma} \in \mathcal{S}$ is an ϵ -(k, t)-robust (resp. strongly ϵ -(k, t)-robust) equilibrium if for all $K, T \subseteq N$ such that $K \cap T = \emptyset$, $|K| \leq k$, and $|T| \leq t$, for all $\vec{\tau}_T \in \mathcal{S}_T$, for all $\vec{\phi}_K \in \mathcal{S}_K$, for some (resp. all) $i \in K$, and all strategies of the scheduler σ_s , we have $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T, \sigma_s) \geq u_i(\vec{\sigma}_{N-(K \cup T)}, \vec{\phi}_K, \vec{\tau}_T, \sigma_s) - \epsilon$. We speak instead of a (k, t) -robust equilibrium if $\epsilon = 0$.

We use a recent definition of k -resilient equilibrium [72], which varies slightly the definition of ϵ -(k, t)-robustness. We define here strong resilience and strong robustness to refer to the

stronger versions of these properties [71]. Byzantine fault tolerance in distributed computing is equivalent to our definition of t -immunity in game theory.

Given some game Γ and desired functionality \mathcal{F} , we say that a protocol $\vec{\sigma}$ is a k -resilient protocol for \mathcal{F} if $\vec{\sigma}$ implements \mathcal{F} and is a k -resilient equilibrium. For example, if $\vec{\sigma}$ is a k -resilient protocol for the consensus problem, then in all runs of $\vec{\sigma}$, every non-deviating player terminates and agrees on the same valid value. We extend this notation to t -immunity and ϵ -(k, t)-robustness. The required functionality of our protocol is reaching agreement.

4.1.2 Punishment strategy

We also restate the definition of a punishment strategy [71] as a threat that honest and rational players can play in order to prevent other rational players from deviating. The punishment strategy guarantees that if k rational players deviate, then $t + 1$ players can lower the utility of these rational players by playing the punishment strategy.

Definition 4.1.4 ((k, t) -punishment strategy). A joint strategy $\vec{\gamma}$ is a (k, t) -punishment strategy with respect to $\vec{\sigma}$ if for all $K, T, P \subseteq N$ such that K, T, P are disjoint, $|K| \leq k, |T| \leq t, |P| > t$, for all $\vec{\tau} \in \mathcal{S}_T$, for all $\vec{\phi}_K \in \mathcal{S}_K$, for all $i \in K$, and all strategies of the scheduler σ_s , we have $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T, \sigma_s) > u_i(\vec{\sigma}_{N-(K \cup T \cup P)}, \vec{\phi}_K, \vec{\tau}_T, \vec{\gamma}_P, \sigma_s)$.

Intuitively, a punishment strategy represents a threat to prevent rational players from deviating, in that if they deviate, then players in P can play the punishment strategy $\vec{\gamma}$ and the deviating rational players decrease their utility with respect to following $\vec{\sigma}$. For example, crime sentences are an effective punishment strategy against committing crimes. Not terminating a protocol if just one player deviates can also be a punishment strategy against deviating from the protocol.

We extend the above-defined terms to their analogous crash fault tolerant counterparts by replacing Byzantine players by crash players in all their definitions, in what we refer to as t -crash-immunity, (k, t) -crash-robustness, and (k, t) -crash-punishment strategy.

4.1.3 Rational agreement

In the remainder of this chapter, we are interested in proposing a consensus protocol that is immune to up to $t_\ell = \lceil n/3 \rceil - 1$ Byzantine failures and robust to a coalition of up to k rational and t Byzantine players in what we refer to as the *rational agreement* problem.

Definition 4.1.5 (Rational Agreement). Consider a system with n players, a protocol $\vec{\sigma}$ solves the rational agreement problem if $\vec{\sigma}$ is a t_ℓ -immune protocol for consensus, and is also ϵ -(k, t)-robust for some $k > 0, t > 0$ such that $n \leq 3(k + t)$.

4.2 Impossibility results

In this section, we first present a new type of punishment strategy, called a baiting strategy. We then prove that baiting strategies are pivotal for the rational agreement problem, as the

problem cannot be solved by a protocol unless it implements a baiting strategy. Finally, we prove the analogous impossibility result in the crash model by replacing Byzantine for crash players, showing that resilient-optimal crash fault-tolerant protocols do not tolerate even one rational player.

4.2.1 Baiting strategies

Our solution to agreement in the presence of rational and Byzantine players, presented in Section 4.3.3, consists of rewarding rational players for betraying the coalition. One may wonder whether rewarding rational players in a coalition is the only way to obtain ϵ -(k, t)-robustness that tolerates coalitions of size $n \leq 3(k + t)$ in partial synchrony. To demonstrate the need for a reward, we first formalize a type of (k, t) -punishment strategy, which we call a (k, t, e) -baiting strategy. A (k, t, e) -baiting strategy is a $(k - e, t)$ -punishment strategy such that $k \geq e > 0$, and these e rational players prefer to actually play the baiting strategy than to deviate with the rest of the players in the coalition. That is, e players of the coalition have to play the baiting strategy for it to succeed, and at least e rational players in the coalition prefer to play the baiting strategy than to deviate with the coalition. An example is offering a crime reduction for a criminal to cooperate with law enforcement into catching the criminal group to which it belongs.

Definition 4.2.1 ((k, t, e) -baiting strategy). A joint strategy $\vec{\eta}$ is a (k, t, e) -baiting strategy with respect to a strategy $\vec{\sigma}$ if $\vec{\eta}$ is a $(k - e, t)$ -punishment strategy with respect to $\vec{\sigma}$, with $0 < e \leq k$ and for all $K, T, P \subseteq N$ such that $K \cap T = \emptyset$, $|P \cap K| \geq e$, $P \cap T = \emptyset$, $|K \setminus P| \leq k - e$, $|T| \leq t$, $|P| > t$, for all $\vec{\tau} \in \mathcal{S}_T$, all $\vec{\phi}_{K \setminus P} \in \mathcal{S}_{K \setminus P} - \{\vec{\sigma}_K\}$, all $\vec{\theta}_P \in \mathcal{S}_P$, all $i \in P$, and all strategies of the scheduler σ_s , we have:

$$u_i(\vec{\sigma}_{N-(K \cup T \cup P)}, \vec{\phi}_{K \setminus P}, \vec{\tau}_T, \vec{\eta}_P, \sigma_s) \geq u_i(\vec{\sigma}_{N-(K \cup T \cup P)}, \vec{\phi}_{K \setminus P}, \vec{\tau}_T, \vec{\theta}_P, \sigma_s).$$

Additionally, we speak of a strong (k, t, e) -baiting strategy in the particular case where for all rational coalitions $K \subseteq N$ such that $|K| \leq k$, $|K \cap P| \geq e$ and all $\vec{\phi}_{K \setminus P} \in \mathcal{S}_{K \setminus P}$ we have: $\sum_{i \in K} u_i(\vec{\sigma}_{N-(K \cup P)}, \vec{\phi}_{K \setminus P}, \vec{\eta}_P, \sigma_s) \leq \sum_{i \in K} u_i(\vec{\sigma}, \sigma_s)$. We write (strong) (k, t) -baiting strategy instead to refer to a (strong) (k, t, e) -baiting strategy for some e , with $0 < e \leq k$.

A baiting strategy illustrates a situation where at least e rational players in the coalition may be interested in baiting other $k + t - e$ rational and Byzantine players into a trap: the $k + t$ of them collude to deviate initially, just so that these e players can prove such deviation by playing the baiting strategy, and get a reward for exposing this deviation. Such a strategy has a significant impact in a protocol to implement agreement. A strong baiting strategy consists of a baiting strategy whereby e deviating players following the baiting strategy does not yield greater payoff to the entire coalition as a whole (if such a coalition consists purely of rational players), compared to following the protocol. This prevents a coalition of rational players from colluding together so as to play the baiting strategy on themselves only with the purpose of splitting the baiting reward among the colluding members. Notwithstanding, neither a baiting strategy nor a strong baiting strategy show that if these e players play the baiting strategy,

then the protocol implements the desired functionality. We illustrate the efficacy of baiting strategies to influence the outcome of a protocol in the example of the rational generals, shown in Figure 4.2. We also speak of a (k, t, e) -crash-baiting strategy to refer to the case in which the t players are crash players, and not Byzantine.

4.2.2 Rational agreement is impossible without a baiting strategy

The reason why a (k, t, e) -baiting strategy is relevant to the consensus problem is that without such a strategy it is not possible to obtain a consensus protocol that is (k, t) -robust where $k + t > t_\ell$. We show this result in Theorem 4.1. The proof is similar to that of the impossibility of t -immune consensus under partial synchrony for $t > t_\ell$ [13], since a partition of rational and Byzantine players can exploit two disjoint partitions of honest players to lead them to different decisions. Let us recall that we do not assume solution preference, and thus the payoffs from a disagreement can be significantly greater than those of agreeing for rational players. For the proof of Theorem 4.1, we first show the more general proof of Lemma 4.1.

Lemma 4.1. It is impossible to obtain a protocol $\vec{\sigma}$ that implements agreement, is t_ℓ -immune and (k, t) -robust, with $k \geq 0$ and $t = \max(t_\ell - k + 1, 0)$, unless there is a (k, t, e) -baiting strategy with respect to $\vec{\sigma}$, for $e > \frac{k+t-n}{2} + t_\ell$.

Proof. We refer to Dwork et al.'s [13] work for the impossibility of increasing $t > t_\ell$ and obtaining agreement (i.e., for $k = 0$). For $k > 0$ with $t \leq t_\ell$, assume the contrary: let $\vec{\sigma}$ be a protocol such that there is no (k, t, e) -baiting strategy with respect to $\vec{\sigma}$ and $\vec{\sigma}$ is (k, t) -robust, for $t = \max(t_\ell - k + 1, 0)$, $k > 0$. Since the protocol is t_ℓ -immune and it works under partial synchrony, the protocol must not require more than $n - t_\ell$ players participating in it in order to take a decision, or else the Byzantine players could prevent termination. Consider a partition of the network between 4 disjoint subsets $N = K \cup A \cup B \cup F$, where K are the rational players (there is at least one), F are the Byzantine players, i.e., $|F| + |K| = t + k \geq t_\ell + 1$, and A and B are the rest of the players such that $|A| + |B| \leq n - t_\ell - 1$ and both $|A| + |F| + |K| \geq n - t_\ell$ and $|B| + |F| + |K| \geq n - t_\ell$ hold. Let $\vec{\theta}$ be the strategy in which the rational players in K deviate with Byzantine players in F and achieve a disagreement between players in A and players in B . If the players in F and K are all Byzantine and rational players, then such a disagreement is always possible and the utility for each rational player is, by definition of the model, greater than that of reaching agreement. Notice also that since $t = \max(t_\ell - k + 1, 0)$, if $e > \frac{k+t-n}{2} + t_\ell$ rational players do not deviate to cause such disagreement, we have that at least one of $|A| + |F| + |K| - e < n - t_\ell$ and $|B| + |F| + |K| - e < n - t_\ell$ holds, or both: for this value of e the deviants cannot cause a disagreement. However, this is not true if instead $e \leq \frac{k+t-n}{2} + t_\ell$. It follows that it is necessary to encourage at least $e > \frac{k+t-n}{2} + t_\ell$ rational players to not deviate into causing a disagreement, which means, by definition, that a (k, t, e) -baiting strategy is necessary. \square

Theorem 4.1. It is impossible to obtain a protocol $\vec{\sigma}$ that implements rational agreement unless there is a (k, t, e) -baiting strategy with respect to $\vec{\sigma}$, for $e > \frac{k+t-n}{2} + t_\ell$.

Rational generals example. We illustrate the intuition behind baiting strategies with an example inspired from the Byzantine generals problem [11] that we refer to as the ‘rational generals’ problem: suppose $n = 7$ Ottoman generals need to agree on whether to attack or retreat. If all generals agree on attacking, they will succeed, if they agree on retreat, they can succeed another day. However, if only some of the generals attack, they will lose. There are two Byzantine generals, i.e., $t = 2$, whose goal is for the Ottomans to disagree on their decision for them to lose, and another rational general, i.e., $k = 1$, who has been offered a bribe \mathfrak{G} in order to contribute to the disagreement, but who is willing to betray the Byzantines for a greater income from the Ottomans. Because of accountability, the generals will eventually be able to track the disagreement to both the t Byzantine and k rational generals, but by then the k rational generals will be enjoying their reward \mathfrak{G} in Constantinople, out of reach.

The generals suspect that there might be a bribed rational general ($k = 1$). In an attempt from them to make the rational general talk, they offer a reward $\mathfrak{R} > \mathfrak{G}$ as a bounty for proving the fraud of every other Byzantine and rational general, that is, if the rational general reveals his identity and that of the t Byzantine generals with proofs, then this rational general is spared and rewarded with \mathfrak{R} , while the t Byzantine generals lose all of their capital (i.e., properties and savings) that they own in the Ottoman empire. In this case, the rational general sees a greater incentive to expose both himself and the Byzantine generals. This is an example of a baiting strategy. Additionally, the Ottoman generals will pay \mathfrak{R} with the capital taken from the t Byzantine generals, so the Ottoman empire will not even pay for the reward.

Notice that Ottoman generals must guarantee to the rational general that they will recognize him as the first to expose the coalition (and the only rightful owner of the reward), so that the rational general is not influenced by a threat from the Byzantine generals to steal the reward if he betrays the coalition. That is, the rational general will only bait the coalition if the protocol ensures that the Byzantine generals will not be able to steal the reward from the rational general after seeing that he betrayed the coalition. This is in order to prevent the Byzantine generals from rushing to bait as soon as they learn the rational general is starting to bait, creating a situation in which both Byzantine and rational generals seem to be legitimate baiters of the coalition.

In the extensive game, this means that the rational general must first behave and make moves as if he would cause the disagreement. Then, the rational general will only bait if he gets both enough evidence of the fraud of the deviants and assurance that the Byzantine generals will not outpace him and steal the reward.

Figure 4.2: Rational generals example.

Proof. By definition, every (k, t) -robust protocol for $n \leq 3(k + t)$ must also be (k, t) -robust, for some $k \geq 0$ and $t = \max(t_\ell - k + 1, 0)$. Therefore it derives from Lemma 4.1. \square

Theorem 4.1 shows the need for a baiting strategy to solve rational agreement. In Section 4.3.2 we show the implementation of an additional phase to an accountable consensus protocol in order to provide the functionality of a baiting strategy. In Section 4.3.3 we illustrate the values of a reward and deposit per player to make a strong baiting strategy that at least e rational players will play.

4.2.3 Impossibility in the presence of rational and crash players

In this section, we show that resilient-optimal, crash fault-tolerant (CFT) protocols cannot tolerate even one rational player. Previous results showed that a resilient-optimal CFT protocol tolerates up to $t < n/2$ crash faults [13]. We show in Lemma 4.2 and Theorem 4.2 that this number of crash faults does not allow the protocol to tolerate even one rational player.

Lemma 4.2. Let $\vec{\sigma}$ be a protocol that implements consensus such that there is no (k, t, e) -crash-baiting strategy with respect to $\vec{\sigma}$. Then, it is impossible for $\vec{\sigma}$ to be t -crash-immune and k -resilient for $k + 2t \geq n$, $e > \frac{k-n}{2} + t$.

Proof. If a protocol is t -crash-immune, that means that the protocol must terminate even if t players do not participate in it at all, since t players may have crashed from the beginning. Therefore, the protocol must terminate and decide with the participation of $n - t$ players.

Since the protocol must be able to terminate with the participation of at most $n - t$ players, consider now that there are no crash players and there are exactly k rational players. Let us find a disjoint partition of honest players A and B such that $k + |A| + |B| = n$. If $k + |A| \geq n - t$ and $k + |B| \geq n - t$ then the k rational players can cause a disagreement, which can occur if $k + 2t \geq n$. There is only left to prove that k rational players will try to cause a disagreement. For this, let us consider the minimum number of rational players e that must not try to cause a disagreement for the remaining deviating rational players to not be able to cause a disagreement. That is, for which values we have $k - e + |A| < n - t$ and $k - e + |B| < n - t$, hence resulting in $e > \frac{k-n}{2} + t$. Therefore, the utilities for rational players from causing a disagreement are greater than from causing agreement. This means that at least enough rational players will deviate and cause the disagreement unless there is a (k, t, e) -baiting strategy that prevents e rational players from deviating into a disagreement. \square

Theorem 4.2 follows directly from Lemma 4.2 because every (k, t) -crash-robust protocol must also be t -crash-immune and k -resilient.

Theorem 4.2. Let $\vec{\sigma}$ be a protocol that implements consensus such that there is no (k, t) -crash-baiting strategy with respect to $\vec{\sigma}$. Then, it is impossible for $\vec{\sigma}$ to be (k, t) -crash-robust for $k + 2t \geq n$.

Proof. The proof is analogous to that of Lemma 4.2, since by definition every (k, t) -crash-robust protocol must also be t -crash-immune and k -resilient. \square

Corollary 4.1. Let $\vec{\sigma}$ be a protocol that implements consensus such that there is no $(1, t)$ -crash-baiting strategy with respect to $\vec{\sigma}$ and is t -crash-immune for $t < n/2$. Then, $\vec{\sigma}$ is not 1-resilient.

The results from Lemma 4.2, Theorem 4.2 and Corollary 4.1 show that it is necessary to consider new bounds for CFT protocols in terms of their crash fault tolerance, since their resilient-optimal bounds make them vulnerable to even one rational player. In Section 4.4, we explore the link between crash-robustness and immunity, so as to obtain results for this model with the TRAP protocol that we describe in Section 4.3.

4.3 TRAP: reaching rational agreement

In this section, we present the TRAP (Tackling Rational Agreement through Persuasion) protocol, which solves rational agreement. The TRAP protocol comprises three components:

1. A **financial** component, consisting of a deposit per player \mathfrak{L} , taken at the start of the protocol from each participating player, and a reward \mathfrak{R} , which is given to a player in the event that it provides PoFs for a disagreement on predecisions.
2. An **accountable consensus** component, that pre-decides outputs from an accountable consensus protocol.
3. A **baiting** component, embodied in a novel Byzantine Fault-Tolerant *commit-reveal* (BFTCR) protocol that executes after the accountable consensus protocol. This component terminates either deciding one output (predecision) of the accountable consensus protocol, or resolving a disagreement on predecisions by rewarding one of the deviating players that exposed the disagreement and punishing the rest of deviating players.

We first provide an overview of the properties that we aim at for the TRAP protocol in Section 4.3.1, and the possible runs of the game that derive from implementing a strong baiting strategy for the rational agreement problem with the aforementioned components. We then introduce and prove the correctness of the baiting component, the BFTCR protocol, in Section 4.3.2. Finally, we analyze the financial component, that is, the specific values of reward and deposits, in Section 4.3.3. The accountable consensus component can be any accountable consensus protocol [47, 132, 181], and thus we treat this component as a black box, for the sake of generality.

4.3.1 Overview: consensus with a baiting strategy

We proved in Section 4.2.2 that we need a baiting strategy for a protocol to solve the rational agreement problem.

Before we present the implementation of such a baiting strategy in Section 4.3.2, with additional configurations of the required deposits and reward sizes in Section 4.3.3, we present in this section the basics of our baiting strategy. For this purpose, we focus first on the properties

that we aim at for such a baiting strategy. Then, we showcase all the possible runs of a protocol for consensus that provide such a strong baiting strategy.

Given a protocol $\vec{\sigma}$ that implements accountable consensus and is t_ℓ -immune, we will extend it to implement rational agreement, in that we will prove the three following properties:

- *Baiting dominance*: There is a (k, t, e) -baiting strategy $\vec{\eta}$ with respect to $\vec{\sigma}$, for $e > \frac{k+t-n}{2} + t_\ell$.
- *Baiting agreement*: $\vec{\eta}$ implements agreement.
- *Lossfree reward*: $\vec{\eta}$ is a strong baiting strategy.

Baiting dominance states the necessary condition that a baiting strategy exists, while baiting agreement guarantees that playing such a baiting strategy still leads to agreement. Lossfree reward guarantees that such a baiting strategy is a strong baiting strategy. Coming back to the rational generals example of Figure 4.2, baiting dominance states the existence of the reward for the rational general, baiting agreement guarantees that generals will still decide whether to attack or retreat after paying the reward to the rational general, and lossfree reward guarantees that only the slashed capital of the Byzantine generals will be used to pay the reward to the rational general.

Reward for baiting. Since the protocol is accountable, we add a *baiting reward* \mathfrak{R} for player i if i can prove to the rest of the players that a coalition of at least $t_\ell + 1$ players are trying to cause a disagreement, but before they succeed at causing the disagreement. If multiple players are eligible for the baiting reward, then only one is chosen at random to win the reward, and the rest are treated as fraudsters that did not bait. We select the winner at random in an additional *winner consensus* in which the winner is decided from among the candidates to win proposed by honest processes in this winner consensus. We explain further the winner consensus later in this section. Players can prove that a coalition is trying to cause a disagreement through PoFs which undeniably show two conflicting messages signed by the same set of players. The reward is only given to i if i exposes this coalition before the coalition causes the disagreement (i.e., before both partitions of honest players decide different decisions).

Funding the reward with deposits. we require all players to place a minimum *deposit* \mathfrak{L} . We also require such deposit to be big enough so that the deposit taken from the exposed coalition is enough to pay the reward, satisfying lossfree reward. Our goal is to set \mathfrak{R} and \mathfrak{L} so that we implement a baiting strategy for a set R of rational players in the coalition, such that if others in the coalition bait, then for all $i \in R$, player i is better off also trying to bait and getting the reward, while if the rest of the players in the coalition do not bait, then if i baits then i gets the greatest expected utility that it can in that information set. We analyze in Theorem 4.4 the required values for such deposit and reward necessary to incentivize at least $|R| = e > \frac{k+t-n}{2} + t_\ell$ rational players in a coalition to follow a baiting strategy, depending on the size $k + t$ of the coalition and on the maximum total gain from disagreeing \mathfrak{G} . For now, however, let us ignore the values of \mathfrak{L} and \mathfrak{R} and focus on the protocol that solves the rational agreement problem, by assuming that these values of \mathfrak{L} and \mathfrak{R} are enough to make $e > \frac{k+t-n}{2} + t_\ell$ rational

players bait the coalition, instead of terminating a disagreement. We will come back to specify proper values for \mathfrak{L} and \mathfrak{R} in Section 4.3.3. If these PoFs expose at least $t_\ell + 1$ players including the winner of the baiting reward \mathfrak{R} , then the t_ℓ (or more) remaining colluding players lose the deposit amount \mathfrak{L} .

Dominating disagreements. We explore here the possible runs, assuming that we already have such a baiting strategy, and what each of these runs means for the payoffs of a rational player i :

1. Rational players including i contribute to reaching agreement and follow the protocol $\vec{\sigma}$, getting some utility $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T) \geq \epsilon$ where $\epsilon > 0$.
2. Some rational players collude with i and deviate to disagree, playing strategy $\vec{\phi}$ with some Byzantine players T and other rational players K such that $|K \cup T| \geq n/3$, $K \cap T = \emptyset$, obtaining utility $u_i(\vec{\sigma}_{N-K-T}, \vec{\phi}_{K \cup T}) = g$.
3. Player i deviates to bait other rational players into colluding with some Byzantine players such that $|K \cup T| \geq n/3$, $K \cap T = \emptyset$, and this deviation consists of playing strategy $\vec{\eta}$ to expose the colluding players via PoFs, obtaining the baiting reward. As a result, player i obtains utility $u_i(\vec{\sigma}_{N-K-T}, \vec{\phi}_{K \cup T-R}, \vec{\eta}_R) = \rho(e)\mathfrak{R} - \bar{\rho}(e)\mathfrak{L}$, where R is the set of players of the coalition that bait, i.e., $i \in R$, with $|R| = e$. $\rho(e) = 1/e$ represents the probability of winning the reward, while $\bar{\rho}(e) = 1 - \rho(e) = (e - 1)/e$ the probability of not winning it after baiting.
4. Player i deviates to disagree only to suffer a trap baited by another rational player (or group of rational players), obtaining utility $u_i(\vec{\sigma}_{N-K-T}, \vec{\phi}_{K \cup T-R}, \vec{\eta}_R) \leq -\mathfrak{L}$.
5. In any run where the protocol does not terminate, player i obtains negative utility.
6. Player i contributes to reaching agreement but a coalition causes a disagreement. In this case, i is one of the victims of a disagreement (for example, a double-spending). Hence, i obtains negative utility.

Notice that runs 4, 5 and 6 are strictly dominated by run 1 (following the protocol). Our goal is to make runs represented by 3 runs that also implement agreement and that strictly dominate runs represented by 2.

4.3.2 Baiting component: the BFTCR protocol

In this section, we present an implementation of a baiting strategy for rational agreement. As such, we extend an accountable consensus protocol with a Byzantine Fault-Tolerant *commit-reveal* (BFTCR) phase in order to solve consensus even if there is a disagreement at consensus level, if at least e rational players decide to betray the coalition so as to try to win a reward. We show in Algorithm 6 the BFTCR phase. As such, we speak of a *predecision* for a decision of the accountable consensus protocol, whereas a *decision* now refers to the outcome of the BFTCR protocol. The BFTCR phase consists of 5 main parts:

1. a reliable broadcast, in which players share their encrypted commitment (line 11),

2. a second reliable broadcast, in which players share the first $(n - t_\ell)$ encrypted commitments that they delivered in the first reliable broadcast (line 15),
3. a regular broadcast, in which players share the key to reveal their commitment (line 19),
4. an additional consensus to select the winner of the reward, if some players reveal a list of PoFs (line 35), and
5. a slashing of the deposits from the fraudsters, payment of the reward to the winner and resolution of the disagreement on predecisions (line 36).

Commit and reveal. The purpose of the first group of reliable broadcasts is to reliably broadcast the encrypted PoFs, should a player own them, or an encrypted hash of a predecision otherwise. We say that the *commitment* is the encrypted content that each player decides to broadcast in this first reliable broadcast. In line 15 each player i then starts the second reliable broadcast by broadcasting a list of the first $(n - t_\ell)$ delivered commitments that i delivered in the first reliable broadcast. The purpose of the calls to broadcast in lines 19 and 21 is to deliver the keys to decrypt the encrypted messages. A player i thus *reveals* his commitment by broadcasting the key. A player i decrypts the commitment of player j in line 22. Then, player i adds this decrypted message to the list of decided hashes in lines 24 to 27, or to the list of PoFs received in lines 29 to 31.

Termination. The BFTCR phase of the TRAP protocol terminates in one of two ways:

- either there is no disagreement on predecisions, and then the protocol terminates when at least $(n - t_\ell)$ messages are decrypted with the same hash of the predecisions in line 27; or
- some players reveal a disagreement on predecisions through PoFs, and then the protocol terminates when at least $t_\ell + 1$ messages are decrypted (without counting players that are proven fraudsters through PoFs) with a reward to a chosen baiter and a punishment to the remaining players that are listed in the PoFs from lines 32 to 36.

Note that accountability does not guarantee that a baiter will gather enough PoFs before a disagreement takes place. We prove that baiters will gather enough PoFs before a disagreement takes place as part of the proof of Theorem 4.3. The idea is that e rational players will wait to receive enough PoFs to be able to commit to bait, where e is big enough to prevent termination of either of the partitions of honest players.

Valid candidates of the winner consensus. We define a *valid candidate* to win the reward as a member of a deviating coalition that committed to bait the coalition (by sending a commitment to a list of PoFs of the coalition in line 11) independently of whether other e players of the coalition also committed to bait, for $e > \frac{k+t-n}{2} + t_\ell$. The objective of the BFTCR protocol is to distinguish valid candidates from players who try to win the reward only after they learn that the disagreement will not succeed. An honest player i considers a baiter j as a valid candidate if i can see j 's commitment to bait in at least $t_\ell + 1$ messages from the second reliable broadcast. We refer to this $t_\ell + 1$ messages as a *proof-of-baiting* (PoB). The BFTCR protocol selects the

winner of the bait among the list of valid candidates by executing an additional consensus, in the call to `select_winner` in line 35, in which all participating players propose the PoFs they know about and the valid candidates, along with the PoBs. We detail further this call later in this section.

Note that a rational player i that commits to bait a coalition may deviate from Algorithm 6 in order to hinder other deviants from becoming valid candidates after i reveals its commitment. This is because this way i maximizes its chances of winning the reward (by minimizing the number of valid candidates for the reward). This is an expected deviation of a baiting rational player, which consists on waiting to deliver as many messages from the second reliable broadcast as possible from both partitions of honest players that suffered the disagreement on predecisions, and we show the correctness of this approach as part of the proof of Theorem 4.4.

Correctness and randomness of the winner consensus. We show in Theorem 4.5 that no deviating player can win the reward without being a valid candidate, i.e., no player can bait and win the reward after learning that other e (or more) players baited. Additionally, note that the winner consensus solves consensus for $n > 9/5(k + t)$ because at least $t_\ell + 1$ provably fraudulent players of the coalition will not participate in it, and we consider $n > 2(k + t)$. That is, at most $n' = n - (t_\ell + 1) < 2n/3$ players participate in the winner consensus. Since the maximum coalition size is $k + t < n/2$, then the remaining players of the coalition that could participate in the winner consensus are $t' = n/2 - (t_\ell + 1) < n/6$, and thus $t' < n'/3$ and the winner consensus solves consensus. Notice this is true even if honest players disagree on the initial $t_\ell + 1$ provably fraudulent players, since eventually they agree on the set of fraudulent players, as we detail in Chapter 5.

Furthermore, the winner consensus only terminates once at least $n - t'$ proposals have been decided, which can be optimized through a democratic consensus protocol [23, 47]. Finally, after $n - t'$ proposals are decided upon, the participants execute an iteration of a random beacon that tolerates $t' < n'/3$ Byzantine faults [36, 31, 32], in order to select the winner of the baiting reward randomly from among any of the valid candidates that were in any of the decided proposals. We present two random beacons that can be used for this purpose in Chapter 6.

Following the winner consensus, in line 36, fraudsters are punished and the baiter is rewarded, respectively. The call to `resolve(...)` resolves the two disagreeing predecisions by deterministically choosing one of them (i.e., lexicographical order) or, depending on the application, merging both to solve SBC (Definition 2.2.4).

Resolving a disagreement on consensus predecisions with BFTCR. It is clear that if there is no disagreement on the predecisions, the BFTCR phase will terminate and satisfy consensus. We consider here the output of the BFTCR phase in the case where there is a disagreement into two predecisions. We speak of a disagreement on predecisions being *finalized* if it becomes a disagreement on decisions (that is, on the output of the BFTCR phase). We will show in Theorem 4.3 that if $e > \frac{k+t-n}{2} + t_\ell$ rational players commit to bait instead of finalizing the disagreement on predecisions, then the TRAP protocol still satisfies consensus. For this purpose, we define $e(k, t) = \lfloor \frac{k+t-n}{2} + t_\ell \rfloor + 1$ (i.e., the smallest natural value that satisfies $e > \frac{k+t-n}{2} + t_\ell$). Then, we first show in Lemma 4.3 that if $e(k, t)$ rational players bait,

Algorithm 6 BFT commit-reveal protocol for (honest) player i .

```

1: State:
2:    $enc\_msgs$ , list of delivered encrypted messages from the first group reliable broadcasts, initially  $\emptyset$ 
3:    $list\_enc\_msgs$ , list of delivered encrypted messages from the first group of reliable broadcasts, initially  $\emptyset$ 
4:    $decrypted\_msgs$ , list of delivered decrypted messages from the first group of reliable broadcasts, initially  $\emptyset$ 
5:    $\{RB_j^1\}_{j=0}^n$ , the first group of reliable broadcasts where  $j$  is the source
6:    $\{RB_j^2\}_{j=0}^n$ , the second group of reliable broadcasts where  $j$  is the source
7:    $hashes$ , a dictionary, keys are hashes and values are integers, initially  $\{\}$ 
8:    $local\_hash$ , local hash of the predecided value, according to this player
9:    $POF\_received$ , Boolean, initially False
10:   $i, i\_msg, i\_key, i\_enc\_msg$ , player's id, message, key, and encrypted message

11:  $RB_i^1.start(i\_enc\_msg)$   $\triangleright$  start first group of reliable broadcasts

12: Upon RB-delivering  $enc\_msg$  from reliable broadcast  $RB_j^1$ :
13:    $enc\_msgs[j] \leftarrow enc\_msgs$ 
14:   if ( $size(enc\_msgs) \geq n - t_\ell$ ) then
15:      $RB_i^2.start(enc\_msgs)$   $\triangleright$  start second reliable broadcast sharing these delivered commitments

16: Upon RB-delivering  $enc\_msgs_j$  from reliable broadcast  $RB_j^2$ :
17:    $list\_enc\_msgs[j] \leftarrow enc\_msgs_j$ 
18:   if ( $size(list\_enc\_msgs) \geq n - t_\ell$  and  $size(enc\_msgs) \geq n - t_\ell$ ) then
19:      $broadcast(i\_key, i)$   $\triangleright$  reveal  $i$ 's commitment by broadcasting decryption key

20: Upon delivering  $key$  from  $j$  and RB-delivering from  $RB_j^1$  and  $RB_j^2$  :
21:    $broadcast(key, j)$ 
22:    $decrypted\_msgs[j] \leftarrow decrypt(enc\_msgs, key)$   $\triangleright$  decrypt  $j$ 's commitment
23:   if ( $decrypted\_msgs[j].type = HASH$ ) then  $\triangleright$  if it is the hash of a predecision
24:      $hash \leftarrow decrypted\_msgs[j].get\_hash()$ 
25:      $hashes[hash] += 1$   $\triangleright$  add to count
26:     if ( $hashes[hash] \geq n - t_\ell$  and  $local\_hash = hashes[hash]$ ) then
27:        $decide(hash)$   $\triangleright$  if count for this hash reaches threshold, then decide it
28:   else if ( $decrypted\_msgs[j].type = POFS$ ) then  $\triangleright$  if instead list of PoFs
29:      $PoFs \leftarrow decrypted\_msgs[j].get\_PoFs()$ 
30:     if ( $verify(PoFs)$ ) then  $list\_PoFs[j] \leftarrow PoFs$   $\triangleright$  verify PoFs are valid
31:      $POF\_received \leftarrow \mathbf{True}$ 
32:   if ( $POF\_received$ ) then
33:      $msgs\_filtered \leftarrow keys(decrypted\_msgs) \setminus keys(PoFs)$   $\triangleright$  count honest decryption keys received
34:     if ( $size(msgs\_filtered) \geq t_\ell + 1$ ) then  $\triangleright$  punish fraudsters, reward winner, and resolve the (pre-)disagreement
35:        $baiter, frauds, predec_1, predec_2 \leftarrow select\_winner(list\_enc\_msgs, lPoFs)$   $\triangleright$  winner consensus
36:        $punish(frauds); reward(baiter); resolve(predec_1, predec_2)$ 

```

then the only possible outcome is to resolve a disagreement on predecisions.

Lemma 4.3. Let n players play the associated game of the TRAP protocol $\vec{\sigma}$, out of which k can be rational and t Byzantine, with $n > 2(k + t)$. Suppose a run in which a coalition causes a disagreement on predecisions, and consider the start of the BFTCR phase. Then, if $e(k, t)$ rational players of the coalition commit to bait then the only possible outcome is to pay the reward and resolve the disagreement on predecisions.

Proof. First, we show that $e(k, t)$ deviating players committing to bait suffices to prevent the disagreement on predecisions to be finalized in a disagreement on decisions. This is analogous to the proof of Lemma 4.1. Then, we show that if $e(k, t)$ players commit to bait, then the BFTCR phase safely terminates resolving predecisions, with all honest players that start the winner consensus terminating it and agreeing. Finally, we show that deviating players cannot get the reward and also cause a disagreement, i.e., if one player terminates the winner consensus then all honest players start it.

Suppose two predecisions v_A, v_B that two partitions of players not in the coalition A and B predecided, such that $A \cap B = \emptyset$, and $|A| + |B| + k + t \leq n$. For A to decide v_A (resp. B to decide v_B), players in A (resp. B) must be able to decide without hearing from players in B (resp. A). Therefore, $|A| + k + t \geq n - t_\ell$ and also $|B| + k + t \geq n - t_\ell$ to finalize the disagreement. We consider now how many e rational players out of k must bait (i.e., must not contribute to finalizing the disagreement) for a disagreement to necessarily fail. This value must be such that $|A| + (k - e) + t < n - t_\ell$ and same for B 's partition, which solves to $e > \frac{k+t-n}{2} + t_\ell$ (analogously to Lemma 4.1).

Then, we recall that the BFTCR phase resolves predecisions, rewards and punishes players if at least $t_\ell + 1$ players have been exposed through PoFs. Thus, every non-deviating player can ignore messages received from a set containing at least $t_\ell + 1$ players. All non-deviating players eventually converge to the same set of detected fraudsters (see Chapter 5), as all honest players broadcast the PoFs they hear from and update their detected fraudsters accordingly. As such, let F represent the set of detected fraudsters, then for all $|F| \in [t_\ell + 1, k + t]$ it follows that $n'/3 > k + t - |F|$ for $n' = n - |F|$, and thus the winner consensus tolerates deviations from the rest of rational and Byzantine players not yet detected.

Finally, we show that if the reward is paid, then it is not possible to cause a disagreement at decision level. We have shown in the previous paragraph that all non-deviating players that execute the winner consensus terminate agreeing. We must thus prove that if an honest player terminates the winner consensus, then no honest player can terminate deciding a predecision without executing the winner consensus. Since $n'/3 > k + t - |F|$, the winner consensus terminates with the participation of just $2n'/3$ players, of which at least $2n'/3 - (k + t - |F|)$ are honest. Since there are $n - k - t$ honest players in total, if the winner consensus terminates for some honest player, then there are at most $c = n - k - t - (2n'/3 - (k + t - |F|)) = (n - |F|)/3$ honest players that have neither learned about the disagreement nor executed the winner consensus yet. Thus, for these remaining honest players to not be able to decide without executing the winner consensus, it is necessary that $c + t + k < n - t_\ell \iff t + k < n/2$.

Hence, as long as at least $e(k, t) = \lfloor \frac{k+t-n}{2} + t_\ell \rfloor + 1$ rational players play the baiting strategy,

the only possible outcome is for one of them to get the reward, and to resolve the disagreement on predecisions. \square

Theorem 4.3 (Baiting agreement). Let n players play the associated game of the TRAP protocol $\vec{\sigma}$, of which k can be rational and t Byzantine, with $n > 2(k + t)$. Suppose that $e(k, t) = \lfloor \frac{k+t-n}{2} + t_\ell \rfloor + 1$ rational players in the coalition play the baiting strategy committing to bait if they participate in a disagreement on predecisions. Then the TRAP protocol solves rational agreement.

Proof. The proof of t_ℓ -immunity follows from the fact that the consensus component is t_ℓ -immune and the fact that the additional BFTCR phase consists of two Byzantine fault-tolerant reliable broadcasts and one additional broadcast per player, terminating each of them if $n - t_\ell$ players follow the protocol.

For $\epsilon(k, t)$ -robustness, it is clear that if there is no disagreement on predecisions, then rational and honest players are more than $n - t_\ell$ and thus the protocol terminates and guarantees validity and agreement. If there is instead a disagreement on predecisions then, as long as $e(k, t)$ players commit to bait, by Lemma 4.3 the only outcome is to pay the reward and resolve the disagreement. \square

We show in Theorem 4.3 that, provided $e(k, t)$ rational players commit to bait if there is a disagreement on predecisions, the TRAP protocol solves the rational agreement problem. We only have left to prove for which values of \mathfrak{L} and \mathfrak{R} we can guarantee that the strategy to bait the coalition strictly dominates that of terminating a disagreement for at least $e(k, t)$ rational players in the coalition. We do this in Section 4.3.3.

4.3.3 Financial component: deposits & reward

In this section, we focus on the key idea of this chapter: what are the values required for a deposit per player and a reward to players for baiting the coalition that make a strong baiting strategy. In particular, and derived from the BFTCR algorithm of Section 4.3.1, we focus on a baiting strategy that at least $e(k, t)$ rational players will play in Theorem 4.4. Then, we prove that the proposed TRAP protocol implements rational agreement and is $\epsilon(k, t)$ -robust for $n > \frac{3}{2}k + 3t$ and $n > 2(k + t)$ in Theorem 4.5 and Corollary 4.2.

We show in Theorem 4.4 which values of \mathfrak{L} and \mathfrak{R} make the disagreeing strategy a strictly dominated strategy by the baiting strategy for at least $e(k, t)$ rational players (i.e., a dominated strategy even if player i already knows that $e(k, t) - 1$ other players are also baiting at the time that i has to decide whether to bait or not). In other words, we show in Theorem 4.4 under which values of \mathfrak{R} and \mathfrak{L} such strategy $\vec{\eta}$ is a strong $(k, t, e(k, t))$ -baiting strategy that satisfies baiting dominance and lossfree reward.

The result of Theorem 4.4 is the key part of the TRAP protocol for two reasons. First, because it shows that the first $e - 1$ baiters do not even prevent a disagreement from taking place, and thus if the rest of $t + k - (e - 1)$ colluding players want to finalize the disagreement, they can. Second, because it shows that if $e - 1$ players commit to bait, then the remaining

$t + k - (e - 1)$ must take the decision on whether to commit to bait or not independently of what the rest of them are doing. Thus, this is analogous to a reduction from the extensive-form game into a normal-form game for this case, played by the $t + k - (e - 1)$ remaining rational and Byzantine players, in which all rational players' dominating strategy is to bait the coalition, regardless of what the rest are doing. Without this proof, Byzantine players in the coalition could threat rational players to also bait if they see them baiting, creating a deterrent and changing the equilibrium of rational players into colluding to finalize the disagreement.

We first show in Lemma 4.4 that the TRAP protocol guarantees that no player can decide to join the baiting strategy $\vec{\eta}$ and become a valid candidate for the winner consensus after learning that another $e(k, t)$ players played $\vec{\eta}$: they must take that decision before they know whether $e(k, t)$ other players will play $\vec{\eta}$ or not.

Lemma 4.4. Let n players play the associated game of the TRAP protocol $\vec{\sigma}$, out of which k can be rational and t Byzantine, with $n > \frac{3}{2}k + 3t$ and $n > 2(k + t)$. Suppose a run in which a coalition causes a disagreement on predecisions and players start the BFTCR phase. Then, deviating player i in the coalition cannot become a valid candidate for the reward unless it commits to bait before it learns that $e(k, t)$ other players commit to bait.

Proof. We show that if $e(k, t)$ rational players in the coalition play the baiting strategy, becoming valid candidates to win the reward, then the remaining $k + t - e(k, t)$ cannot obtain valid PoBs to become candidates of the winner consensus after learning that $e(k, t)$ players become candidates. Given that the non-baiting members of the coalition are trying to finalize a disagreement, they will still split non-deviating players into two partitions A and B for the BFTCR protocol. Hence, we look at how many rational players must take part in both partitions of the BFTCR protocol. Notice that $|A| + |B| + t + k \leq n$, $|A| + k + t \geq n - t_\ell$ and $|B| + k + t \geq n - t_\ell$. Thus, analogous to how we calculate e in Lemma 4.1, we have that $c \geq (n - t_\ell) - \frac{n - t - k}{2}$ is the number of members of the coalition that must participate in a partition for it to terminate deciding a predecision, with $A \cap B = \emptyset$, as their predecisions differ. We are interested in calculating $c - t$, the minimum number of rational players out of these c members of the coalition, this is why we include as many Byzantine players as possible. Notice also that we want to see how many rational players must take part in both partitions, meaning that we are interested in $c - t - \frac{k}{2} = (n - t_\ell) - \frac{n + t}{2} \geq e(k, t)$ for $n > \frac{3}{2}k + 3t$.

Hence, both partitions will include at least $e(k, t)$ repeated rational players. What is left to prove is that if these $e(k, t)$ players commit to bait, then by the time they reveal their commitment, the remaining players cannot collude to try and obtain PoBs to become valid candidates of the winner consensus too. Since $|A| + k + t \geq n - t_\ell$ and $|B| + k + t \geq n - t_\ell$, there are $|D| \geq 2(n - t_\ell) - 2k - 2t$ honest players that delivered at least $e(k, t)$ commitments to bait, for $|D| \leq |A| + |B|$, if these $e(k, t)$ repeated rational players commit to bait. Notice that $|D| \geq t_\ell + 1$ for $n > 2(k + t)$. Then, each of the $e(k, t)$ players can wait for $n - t_\ell$ deliveries of the second reliable broadcast before revealing their commitment by broadcasting their key without compromising termination. Thus, we must calculate for which values of k and t the remaining players cannot obtain PoBs to become valid candidates, that is, for which values of k and t other players that did not bait yet cannot include the new commitment to bait in

$t_\ell + 1$ valid second reliable broadcasts. Since the remaining set of honest players C such that $|C| = n - t - k - |D|$ are $|C| \geq n - k - t - (2(n - t_\ell) - 2k - 2t)$, we calculate for which values of k and t we have $|C| + k + t - t_\ell \leq t_\ell$, which results in $n > 2(k + t)$. This means that the $e(k, t)$ baiters can be sure that no deviating player can commit to bait and win the reward without being a valid candidate for the winner consensus. \square

We use the result from Lemma 4.4 to prove lossfree reward and baiting dominance in Theorem 4.4.

Theorem 4.4 (lossfree reward and baiting dominance). Let $\vec{\sigma}$ be the TRAP protocol, executed by n players of which exactly k are rational and t Byzantine, for some values of k, t satisfying $n > \max(\frac{3}{2}k + 3t, 2(k + t))$. Let $\vec{\eta}$ be the strategy in which $e(k, t)$ rational players reveal PoFs of the coalition if there is a disagreement on predecisions. Then, $\vec{\eta}$ is a strong baiting strategy if:

1. each player is required to deposit $\mathfrak{L} = \mu \cdot \mathfrak{G}$, with $\mu > \frac{e(k, t)}{k(t_\ell - e(k, t) + 1)}$, and
2. the baiting reward \mathfrak{R} is such that $\mathfrak{R} = t_\ell \mathfrak{L}$.

Proof. Recall that the gain is split equally among all k rational players in the coalition $g = \mathfrak{G}/k$. To guarantee lossfree reward, the sum of losses from the coalition must always be equal or greater than the reward given for the coalition to always lose funds while failing to disagree, that is $t_\ell \mathfrak{L} \geq \mathfrak{R} \iff \mathfrak{L} \geq \frac{\mathfrak{R}}{t_\ell}$.

As a result, the baiting strategy $\vec{\eta}$ must strictly dominate the strategy to disagree for rational players, even if a rational player knows another $e - 1$ other rational players also play the same strategy $\vec{\eta}$ committing to bait. Since the probability of winning the bait between e players is uniformly distributed $\rho(e) = \frac{1}{e}$ we have that the utility for a player to play the baiting strategy knowing that another $e(k, t) - 1$ players are playing the same strategy is $\rho(e(k, t))\mathfrak{R} - \bar{\rho}(e(k, t))\mathfrak{L}$. If, instead, the player disagrees then the player's utility is $\frac{\mathfrak{G}}{k}$. As such, and since Lemma 4.4 shows that no rational player can become a valid candidate to win the reward after learning that $e(k, t)$ other players commit to bait, we obtain that $\vec{\eta}$ strictly dominates the disagreeing strategy if $\rho(e(k, t))\mathfrak{R} - \bar{\rho}(e(k, t))\mathfrak{L} > \frac{\mathfrak{G}}{k}$ and replacing \mathfrak{R} by $t_\ell \mathfrak{L}$, and \mathfrak{L} by $\mu \mathfrak{G}$ we obtain:

$$\mu > \left(k \left(t_\ell \rho(e(k, t)) - \bar{\rho}(e(k, t)) \right) \right)^{-1} \iff \mu > \frac{e(k, t)}{k(t_\ell - e(k, t) + 1)}.$$

As for the reward, $t_\ell \mathfrak{L} \geq \mathfrak{R}$ for the slashed deposits to always cover the reward, and thus we set $t_\ell \mathfrak{L} = \mu \mathfrak{G} t_\ell = \mathfrak{R}$.

Hence, $e(k, t)$ will play the baiting strategy (baiting dominance) of which one will be rewarded, and the reward will be paid with the deposits of the fraudsters (lossfree reward). \square

Notice that any two values \mathfrak{L} and \mathfrak{R} suffice if they satisfy $\rho(e(k, t))\mathfrak{R} - \bar{\rho}(e(k, t))\mathfrak{L} > \frac{\mathfrak{G}}{k}$ (4.1), so that rational players prefer to bait than to disagree, and $t_\ell \mathfrak{L} \geq \mathfrak{R}$ (4.2), so that the reward is always less than the slashed deposits.

The key to these two equations lies in the trade-off between \mathfrak{R} and \mathfrak{L} , that is: \mathfrak{R} must be sufficiently large compared to \mathfrak{L} so that players prefer to bait than to disagree (Equation 4.1),

but \mathfrak{R} must be sufficiently small compared to \mathfrak{L} so that the slashed deposits can always pay for the reward (Equation 4.2).

It is already possible to derive from Theorem 4.4 results for the number of Byzantine players tolerated for $\epsilon - (k - t, t)$ -robustness, given a deposit. That is, suppose that $\vec{\eta}$ only requires $e(k, t) = 1$ rational player to satisfy agreement, and let $\mathfrak{L} = \mu \cdot G$, then every coalition of size at least $t_\ell + 1$ players has at least $k \geq t_\ell + 1 - t$ rational players, and thus the maximum amount of Byzantine players tolerated for $\epsilon - (k - t, t)$ -robustness is $t < t_\ell + 1 - \frac{1}{t_\ell \mu}$. For example, let us set the deposit $\mathfrak{L} = \mu \mathfrak{G}$ to $\mu = \frac{1}{n}$, i.e., the total deposit is $\mathcal{D} = \mathfrak{L} \cdot n = \mathfrak{G}$, and $n = 100$, it follows that the TRAP protocol is $\epsilon - (k - t, t)$ -robust and $t \leq 30$. If instead $\mu = \frac{1}{3n}$, then $t \leq 24$.

Finally, we gather all results together in Theorem 4.5, and Corollary 4.2.

Theorem 4.5. Let $\vec{\sigma}$ be the TRAP protocol, executed by n players of which k are rational and t Byzantine, for all values k, t satisfying $n > \max(\frac{3}{2}k + 3t, 2(k + t))$. Let $\vec{\eta}$ be the strategy in which $e(k, t)$ rational players reveal PoFs of the coalition if there is a disagreement on predecisions. Then, $\vec{\eta}$ is a strong $(k, t, e(k, t))$ -baiting strategy if:

1. each player is required to deposit $\mathfrak{L} = \mu \cdot \mathfrak{G}$, where $\mu > \max_{(k,t)} \left(\frac{e(k,t)}{k(t_\ell - e(k,t) + 1)} \right)$, and
2. the baiting reward is $\mathfrak{R} = t_\ell \mathfrak{L}$.

Proof. Theorem 4.3 uses the proof of baiting agreement from Lemma 4.3 to show that if $e(k, t)$ play the baiting strategy in the event of a disagreement on predecisions, then the TRAP protocol solves the rational agreement problem. Theorem 4.4 shows that $e(k, t)$ will play the baiting strategy (baiting dominance) of which one will be rewarded, and the reward will be paid with the deposits of the fraudsters (lossfree reward).

Finally, we consider all possible values of k and t analogously to Theorem 4.4, deriving a value of μ that holds for all possible values of k and t : $\mu > \max_{(k,t)} \left(\frac{e(k,t)}{k(t_\ell - e(k,t) + 1)} \right)$. \square

Notice that the greater the size of the coalition, the greater μ must be in order for the protocol to be $\epsilon - (k, t)$ -robust. However, for $n > \frac{3}{2}k + 3t$ and $n > 2(k + t)$, since for every two rational players that join the coalition one Byzantine must leave, the coalition that maximizes the total deposit $\mathcal{D} = \mathfrak{L}n = \mu \mathfrak{G}n$ is a coalition of $k = 1$ rational player and $t = t_\ell$ Byzantine players, and that means $\mu > \frac{1}{\lceil \frac{n}{3} \rceil - 1}$. Corollary 4.2 shows such particular robustness.

Corollary 4.2. Let $\vec{\sigma}$ be the TRAP protocol. Then $\vec{\sigma}$ is $\epsilon - (k, t)$ -robust for the rational agreement problem for $n > \frac{3}{2}k + 3t$ and $n > 2(k + t)$ if the following predicates hold:

1. Each player is required to deposit $\mathfrak{L} = \mu \cdot \mathfrak{G} + \epsilon$, where $\mu = \frac{1}{\lceil \frac{n}{3} \rceil - 1}$ and $\epsilon > 0$, and
2. the baiting reward is $\mathfrak{R} = t_\ell \mathfrak{L}$.

Thus, there are two possible outcomes for the TRAP protocol:

- if the coalition is made by so many rational players that deviating does not compensate the risk of losing their deposits, then the TRAP protocol will provide agreement at predecision level without paying a reward \mathfrak{R} , or

- if the coalition has enough Byzantine players to make the deviation into two predecisions profitable, then enough $e(k, t)$ rational players in the coalition will bait so that the disagreement on predecisions can safely be resolved and decided, and one rational player among the baiters will receive a reward \mathfrak{R} , paid entirely by the deposits of the rest of the provably fraudulent players.

In both scenarios, the TRAP protocol implements rational agreement, being ϵ -(k, t)-robust for $n > \max\left(\frac{3}{2}k + 3t, 2(k + t)\right)$.

4.4 Bridging the gap: crash and rational as Byzantine players

In this section we bridge the gap between games that are robust against Byzantine players and games that are robust against crash players, showing that the TRAP protocol is not only (k, t) -robust for consensus for $n < \max(\frac{3}{2}k + 3t, 2(k + t))$ but also $(k + t, t)$ -crash-robust for the same values of k and t .

More generally, we show, on the one hand, that a (k, t) -robust consensus protocol becomes $(k + t, t)$ -robust in the crash model. On the other hand, we show that the existence of a (k, t) -robust consensus protocol in the crash model that does not make use of a baiting strategy implies the existence of a ϵ -($k - t, t$)-robust consensus protocol in the Byzantine model, with the help of cryptography.

4.4.1 From Byzantine to crash players

It is immediate that a protocol that tolerates up to t Byzantine faults also tolerates up to t crash faults, the question lies with the inclusion of rational players. We propose in Lemma 4.5 a first relation between Byzantine fault tolerance and crash-fault tolerance in the presence of rational players.

Lemma 4.5. Let $\vec{\sigma}$ be a protocol that implements consensus and is ϵ - s -immune. Then $\vec{\sigma}$ is also ϵ -(s, s)-crash-robust.

Proof. We prove this by contradiction. Let r be the minimum number of players that must participate in the protocol for it to terminate, it is clear that $r \leq N - s$ since the protocol is ϵ - s -immune. As such, let A and B be two disjoint sets of honest players. Since the protocol must also guarantee agreement, it follows by contradiction that if agreement was not satisfied then $|A| + s \geq r$ and $|B| + s \geq r$, but this is only possible if $r \leq \frac{n+s}{2}$. Therefore, we have $N - s \geq r > \frac{n+s}{2}$.

We define $s_c \leq s$ and $s_k \leq s$ as the maximum number of tolerated crash and rational faults, respectively. It is immediate that termination is guaranteed, since rational players will participate and $r \leq N - s \leq N - s_c$. For agreement, we have $|A| + |B| + s_c + s_k < n$, with $s_c \leq s$ and $s_k \leq s$. We consider that s_c crash players crash after having sent some messages only to players in $|A|$ and the s_k rational players, which is the best case for the deviating coalition (otherwise they crash sending the same message to the entire set of honest players and thus they do not contribute to disagreeing). For the s_k rational players to lead players in B to a different

decision than the decision of players in A plus the crash players s_c , both $s_k + s_c + |A| \geq r$ and $s_k + |B| \geq r$ must hold. This means that $|A| + |B| + 2s_k + s_c \geq 2r \iff n + s_k \geq 2r \iff \frac{n+s}{2} \geq r$, however, this is a contradiction: we already showed that $\frac{n+s}{2} < r$ for $\vec{\sigma}$ to be ϵ - s -immune. It follows that if a protocol that implements consensus is ϵ - s -immune then it is also ϵ -(s, s)-crash-robust. \square

Notice that the statement of Lemma 4.5 does not require to assume cryptography, and thus the same result takes place by considering $\epsilon = 0$, i.e., (k, t) -robustness. The same occurs with Theorem 4.6. Lemma 4.5 establishes a surprising yet meaningful relation between t -immunity and (k, t) -crash-robustness, further extended by Theorem 4.6: if a protocol is (k, t) -robust then it is also $(k + t, t)$ -crash-robust. This means that our TRAP protocol is $(k + t, t)$ -crash-robust for $n < \max(\frac{3}{2}k + 3t, 2(k + t))$, since we showed in Section 4.3 that it is (k, t) -robust for the same values of k and t . We omit the proofs of Theorems 4.6 and 4.7 as they are immediate from the proof of Lemma 4.5.

Theorem 4.6. Let $\vec{\sigma}$ be a protocol that implements consensus and is ϵ -(k, t)-robust. Then, $\vec{\sigma}$ is also ϵ -($k + t, t$)-crash-robust.

By Lemma 4.5 and Theorem 4.6, it is possible to take existing protocols, bounds and other results that apply to immunity and robustness and apply them directly to crash-immunity and crash-robustness. Moreover, Lemma 4.5 establishes a parametrizable hierarchy between crash faults and Byzantine faults: for every Byzantine fault tolerated by a protocol that solves consensus, the same protocol tolerates instead one crash fault and one rational player.

Interestingly, an analogous proof provides the same result for a protocol that tolerates instead crash and Byzantine players. We show this result in Theorem 4.7. However, we first define ϵ -(t', t)-immunity to combine tolerance to a number of crash and Byzantine players together:

Definition 4.4.1 (ϵ -(t', t)-immunity). A joint strategy $\vec{\sigma} \in \mathcal{S}$ is ϵ -(t, t')-immune if, for all sets T of Byzantine players such that $T \subseteq N$ with $|T| \leq t$, all sets T' of crash players such that $T' \subseteq N$, $T \cap T' = \emptyset$, all $\vec{\tau} \in \mathcal{S}_T$, all $\vec{\theta} \in \mathcal{S}_{T'}$, all strategies σ_s of the scheduler, and all $i \notin T \cup T'$, we have:

$$u_i(\vec{\sigma}_{-T \cup T'}, \vec{\tau}_T, \vec{\theta}_{T'}, \sigma_s) \geq u_i(\vec{\sigma}, \sigma_s) - \epsilon.$$

Theorem 4.7. Let $\vec{\sigma}$ be a protocol that implements consensus and is ϵ -(t', t)-immune. Then, $\vec{\sigma}$ is also ϵ -($t, t' + t$)-crash-robust.

4.4.2 From crash to Byzantine players

One may wonder if the same result listed in Lemma 4.5 is true in the opposite direction, that is, whether a protocol $\vec{\sigma}$ that is ϵ -(k, t)-crash-robust is also ϵ -fun(k, t)-immune where $\text{fun}(k, t) = s$ for some $s > 0$.

We prove in Lemma 4.6 that we can construct a protocol that implements consensus and is ϵ - s -immune based on a protocol that is ϵ -(k, t)-crash-robust for $s = \min(k, t)$, assuming cryptography and that the protocol does not implement a (k, t) -crash-baiting strategy.

Lemma 4.6. Let $\vec{\sigma}$ be an ϵ -(k, t)-crash-robust protocol that implements consensus without a (k, t) -crash-baiting strategy with respect to $\vec{\sigma}$. Then, assuming cryptography and a public-key infrastructure scheme, there is an ϵ -min(k, t)-immune protocol $\vec{\sigma}'$ that implements consensus.

Proof. We show how we create the protocol $\vec{\sigma}'$ from $\vec{\sigma}$ to tolerate the new deviations that Byzantine players can perform. We list the possible deviations of t Byzantine players in a protocol $\vec{\sigma}'$:

1. Byzantines players force disagreement by sending equivocating messages.
2. Byzantine players stop replying.
3. Byzantine players reply only to a subset of the honest players.
4. Byzantine players reply wrongly formatted messages.
5. Byzantine players force non-termination by sending equivocating messages.
6. Byzantine players force non-decision (empty decision) by sending equivocating messages.

Protocol $\vec{\sigma}$ already tolerates coalitions of up to $\min(k, t)$ players performing deviation 2, since t crash players can stop replying. For deviation 1, we show that if the k rational players are not enough to cause a disagreement, then k Byzantine players would also not be enough. Consider instead that k rational players can make a coalition big enough to cause a disagreement, then it is clear that they would cause a disagreement unless there is a (k, t) -crash-baiting strategy that prevents it, by definition. Therefore, since $\vec{\sigma}$ does not implement a (k, t) -crash-baiting strategy, if the protocol tolerates k rational players trying to deviate then it also tolerates deviation 1 from Byzantine players. We now show how to construct $\vec{\sigma}'$ to be robust against the rest of the deviations. To tolerate deviation 4, honest players in $\vec{\sigma}'$ ignore wrongly formatted messages, converting deviation 4 into the same deviation as 2. Now, we consider deviations 5 and 6. Since up to $\min(k, t)$ rational players cannot force a disagreement, these players would not even deviate to not terminate or to not decide (their expected utilities from playing such strategies is less than from following $\vec{\sigma}$), however, $\min(k, t)$ Byzantine players can have a greater expected utility from such deviations.

First, we require every player to broadcast any signed message newly delivered. This makes deviation 3 not a deviation anymore, since every honest player eventually verifies and delivers all messages. Also, in the event of an impasse between two partitions (that is, deviations 5 and 6), this makes it possible for honest players to gather enough evidence of which processes are responsible for such an event, in that they signed conflicting messages. If protocol $\vec{\sigma}$ decides an empty proposal in the absence of agreement, then we construct $\vec{\sigma}'$ so that it instead repeats the protocol in a new round. This way, we make deviations 5 and 6 the same deviation.

What is left to prove is that it is impossible for a coalition of up to $\min(k, t)$ Byzantine players to force non-termination by leading honest players into a next round sending equivocating messages. For this purpose, recall that every message sent to a non-empty subset of honest players eventually reaches all honest players thanks to transferable authentication [182], i.e. all

honest relay to the rest all signed messages they deliver. As such, honest players are eventually able to gather two conflicting, equivocating messages from each of the deviating players, identifying such set as responsible for the attempted equivocation. Thus, we describe the final modification of $\vec{\sigma}'$ with respect to $\vec{\sigma}$: once an honest player i identifies (via conflicting signed messages) a player j that sent equivocating messages, i ignores any message coming directly from j from that moment on. Notice that this modification thus converts deviations 5 and 6 into either deviation 3 or 2, and we already showed that $\vec{\sigma}'$ tolerates such deviations as long as the number of Byzantine players is at most $\min(k, t)$. Therefore, we have constructed $\vec{\sigma}'$ extending $\vec{\sigma}$ so that every above-shown deviation from up to $\min(k, t)$ Byzantine players converts into a deviation that $\vec{\sigma}$ already tolerates, meaning that $\vec{\sigma}'$ is ϵ - $\min(k, t)$ -immune. \square

Lemma 4.6 establishes a relation in the opposite direction from Lemma 4.5. We conjecture that this is possible to prove even without the help of cryptography. Nevertheless, we do need to restrict the protocol $\vec{\sigma}$ to be ϵ -(k, t)-crash-robust without the help of (k, t)-crash-baiting strategies. This is because we can only consider k rational players that behave as Byzantine faults in terms of equivocation, that is, that try to cause a disagreement. The existence of a (k, t)-crash-baiting strategy means that some rational players will not try to cause a disagreement, and thus we could not rule out deviation 2 as a deviation that Byzantine players can follow in order to break safety. Nevertheless, this is not surprising, since Byzantine players represent many more deviations by players whose utilities are not even defined, it is expected that more assumptions are required by the direction of Lemma 4.6 than by that of Lemma 4.5. Since the proof of Lemma 4.6 is constructive in providing $\vec{\sigma}'$, we refer to the resulted $\vec{\sigma}'$ as the *BFT-extension* of $\vec{\sigma}$.

We show in Theorem 4.8 the analogous result to Lemma 4.6 as Theorem 4.6 is to Lemma 4.5. The proofs of theorems 4.8 and 4.10 are analogous to that of Lemma 4.6.

Theorem 4.8. Let $\vec{\sigma}$ be an ϵ -(k, t)-crash-robust protocol that implements consensus without a (k, t)-crash-baiting strategy with respect to $\vec{\sigma}$, where $k \geq t$. Then, assuming cryptography and a public-key infrastructure scheme, there is an ϵ -($k - t, t$)-robust protocol $\vec{\sigma}'$ that implements consensus.

Theorem 4.8 excludes protocols that make use of a (k, t)-crash-baiting strategy $\vec{\eta}$ to be ϵ -(k, t)-crash-robust, we show in Theorem 4.9 that if $\vec{\eta}$ is both an effective (k, t)-crash-baiting strategy and an effective ($k - t, t$)-baiting strategy, where *effective* means that playing the baiting strategy still implements consensus, then there is a protocol that is ϵ -($k - t, t$)-robust.

Theorem 4.9. Let $\vec{\sigma}$ be an ϵ -(k, t)-crash-robust protocol that implements consensus such that there is an effective (k, t)-crash-baiting strategy $\vec{\eta}$ with respect to $\vec{\sigma}$, where $k \geq t$. Let $\vec{\sigma}'$ be the BFT-extension of $\vec{\sigma}$, assuming cryptography and a public-key infrastructure scheme. If there is $\vec{\eta}'$ such that $\vec{\eta}'$ is also an effective ($k - t, t$)-baiting strategy with respect to $\vec{\sigma}'$, then $\vec{\sigma}'$ is an ϵ -($k - t, t$)-robust protocol that implements consensus.

Proof. The proof is analogous to that of Lemma 4.6, with the addition that since $\vec{\eta}'$ is an effective ($k - t, t$)-baiting strategy with respect to $\vec{\sigma}'$, in every scenario where $\vec{\eta}$ is played in $\vec{\sigma}$, then $\vec{\eta}'$ is also played in $\vec{\sigma}'$, and since $\vec{\eta}'$ is effective, it implements consensus. \square

We make use of cryptography in Lemma 4.6 in order to offer a constructive proof that is useful for both Theorem 4.8 and Theorem 4.9. Notice however that it is trivial from Theorem 4.2 that $k + 2t < n$ and thus $\min(k, t) < n/3$.

Again, notice that the results from Theorems 4.8 and 4.9 assume $k \geq t$, if instead $t \geq k$, then we obtain the result from theorems 4.10, for which we reuse the definition of ϵ -(t, t')-immunity from Section 4.4.1.

Theorem 4.10. Let $\vec{\sigma}$ be an ϵ -(k, t)-crash-robust protocol that implements consensus without a (k, t) -crash-baiting strategy with respect to $\vec{\sigma}$, where $t \geq k$. Then, assuming cryptography and a public-key infrastructure scheme, there is an ϵ -($t - k, k$)-immune protocol $\vec{\sigma}'$ that implements consensus.

4.5 Summary

In this chapter, we first justified baiting strategies by showing that, both in the Byzantine and in the crash fault model, it is impossible to solve the consensus problem beyond their classical impossibility bounds by introducing rational players unless baiting strategies are implemented.

To this end, we proposed TRAP, a protocol that bypasses these bounds by implementing a baiting strategy to be (k, t) -robust and $(k + t, t)$ -crash-robust for $n > \max\left(\frac{3}{2}k + 3t, 2(k + t)\right)$. TRAP builds upon an accountable consensus protocol by extending it with a reward and deposit per player, and an additional novel BFTCR phase that we present in this chapter, ensuring the existence of a baiting strategy that solves consensus and in which the reward is paid with the slashed deposits from proven deviants.

These results are prosperous for applications like blockchains where players are often found to display rational behavior to maximize their profits while minimizing their risks. Moreover, our model only assumes that rational players deviate if their expected gain from causing a disagreement is greater than the gain from exposing the coalition, or that of following the protocol. These gains are easily calculable in blockchains by simply adding up the outputs of all transactions in a block.

Nevertheless, the results of this chapter can not be met if rational players are not properly bounded by this calculation of the expected gain of a disagreement. This problem becomes evident when considering repeated consensus: rational players can expose the coalition and get rewarded once, after which the coalition cannot try to perform an attack, or they can misbehave with the coalition for as many iterations of consensus as they can.

For this reason, we take our model one step further in Chapter 5 presenting a new type of fault that models any player that is interested in causing a disagreement, but not in preventing termination, in what we call the Byzantine-deceitful-benign (BDB) failure model. This model offers guarantees even when our rational model does not properly asses the benefits of a disagreement for rational players. Interestingly, this model will also allow for a consensus protocol that, even if it does not implement a baiting strategy, exposes a flexibility to different faults that enriches its tolerance to faults compared to classical resilient-optimal, Byzantine fault-tolerant consensus protocols.

Additionally, we take a first step in the solution to repeated consensus for the rational model with Byzantine faults in Chapter 6 by proposing a novel random beacon protocol that produces random outputs in the presence of a coalition of t_s processes trying to control the randomness of these outputs.

Chapter 5

ZLB, a Blockchain Tolerating Colluding Majorities

In Chapter 4, we showed TRAP, a protocol for consensus that overcomes the classical bound of BFT consensus protocols by exploiting the rationality of players in blockchains.

TRAP ensures that a disagreement on predecisions does not materialize into a disagreement on decisions (the output of the new BFTCR phase) as long as at most t_ℓ decide to disagree on decisions, and achieves so by incentivizing rational players not to participate with Byzantine players.

However, rational players might be motivated by more complex utility functions against which the properties guaranteed by the TRAP protocol may break, because if $t_\ell + 1$ or more processes decide to disagree on decisions then there will be a disagreement on the output of the BFTCR protocol. In fact, rational players can even be bribed [120]. As noted in Section 4.5, this is a notorious problem when extending the analysis from a single-shot consensus to a repeated consensus problem, the natural step to extend the results from the TRAP consensus protocol for blockchains.

For this reason, in this chapter we explore protocols and show results even in the presence of rational players that prefer to cause a disagreement and have no interest in preventing termination, but whose exact utility functions are unknown otherwise.

Our result. To this end, we present the *Byzantine-deceitful-benign* (BDB) failure model, introducing two new types of processes, characterized by the faults they commit. First, a *deceitful* process is a process that sends some conflicting messages (messages that contribute to a violation of agreement). Second, a *benign* process is a faulty process that never sends any conflicting messages, contributing to non-termination. For example, a benign process can crash or send stale messages, or even equivocate as long as its messages have no effect on the agreement property. These two faults lie at the core of the consensus problem.

We present a new lower bound on the solvability of the Byzantine consensus problem by precisely exploring these two additional types of faults (that either prevent termination or agreement when $t \geq n/3$). Our lower bound states that there is no protocol solving consensus in the partially synchronous model [13] if $n \leq 3t + d + 2q$ with t Byzantine processes, d deceitful

processes, and q benign processes.

Furthermore, we show that this lower bound is tight, in that we present the Basilic¹ class of protocols that solves consensus with $n > 3t + d + 2q$. Like TRAP, Basilic builds upon accountability, but unlike previous accountable solutions, processes executing Basilic exclude detected fraudsters immediately after detection. Recipients also cross-check the messages they received with other recipients. Thanks to this exclusion, Basilic satisfies a new property, *active accountability*, which guarantees that deceitful processes can not prevent termination.

Basilic is a class of consensus protocols, each parameterized by a different *voting threshold* or the number of distinct processes from which a process receives messages in order to progress. For a voting threshold of $h \in (n/2, n]$, Basilic satisfies termination if $h \leq n - q - t$, and agreement if $h > \frac{d+t+n}{2}$. This means that for just one threshold, say $h = 2n/3$, Basilic tolerates multiple combinations of faulty processes: it can tolerate $t < n/3$, $q = 0$ and $d = 0$; but also $t = 0$, $q < n/3$ and $d < n/3$; or even $t < n/6$, $q < n/6$ and $d < n/6$. This voting threshold can be modified by an application in order to tolerate any combination of t Byzantine, d deceitful and q benign processes satisfying $n > 3t + d + 2q$.

The generalization of Basilic to any voting threshold h thus allows us to pick the best suited protocol depending on the application requirements. If, on the one hand, the application runs in a closed network (e.g., data center) dominated by benign processes, then the threshold will be lowered to ensure termination. If, on the other hand, the application runs in an open network (e.g., blockchain) dominated by deceitful processes, then the threshold will be raised to ensure agreement.

After presenting Basilic, we propose the *Zero-Loss Blockchain* (or *ZLB* for short), a blockchain that simultaneously solves consensus in the presence of a Byzantine adversary controlling a third of the processes and eventually solves consensus in the presence of an adversary controlling more than half of the processes with the purpose of causing a disagreement, both without assuming synchrony. The key breakthrough of ZLB is that it falls back to eventual consensus only for a finite amount of time, after which consensus can be solved again until the adversary changes the processes it corrupts. We call this property convergence. More specifically, ZLB solves consensus for $t + d < n/3$ while also falling back to the eventual consensus problem only in a bounded amount of unlucky cases where $n/3 \leq t + d < 2n/3$.

To demonstrate the efficiency of ZLB, we implement it with Bitcoin transactions, and compare its performance to modern blockchain systems. We show that, on 90 machines spread across distinct continents, ZLB outperforms by 5.6 times the HotStuff [79] state machine replication that inspired Facebook’s Libra [183], and obtains comparable performance to the recent Red Belly Blockchain [28]. Our empirical results also show an interesting phenomenon in that the impact of the attacks decreases rapidly as the system size increases, due to the increased message delays.

Furthermore, we also develop a Zero-Loss Payment application on top of ZLB, in which we guarantee that the financial losses from potential disagreements caused by attackers are canceled

¹The name “Basilic” is inspired from the Basilic cannon that Ottomans used to break through the walls of Constantinople. Much like the cannon, our Basilic protocol provides a tool to break through the classical bounds of Byzantine fault tolerance.

out by the deposit taken from the same attackers to fund the effects of the disagreement.

Summary. In summary, in this chapter:

- i) We present the novel BDB failure model, compare it with previous models and justify it.
- ii) We extend the classical impossibility bounds of BFT consensus to the BDB model.
- iii) We introduce the Basilic class of consensus protocols that we prove resilient-optimal in both the BFT and the BDB model.
- iv) We show that protocols of the Basilic class have optimal communication complexity.
- v) We introduce the Longlasting Blockchain (LLB) problem designed to solve the blockchain problem in situations where the adversary can cause a disagreement.
- vi) We present the Zero-Loss Blockchain (ZLB), a blockchain that solves LLB using the Basilic class.
- vii) We build ZLB and compare its performance with the state of the art, showing that it is faster than Facebook’s Libra blockchain, and competitive with the recent Red-Belly blockchain that is not accountable.
- viii) We build a zero-loss payment application on top of ZLB in which no honest process or user loses any funds resulting from disagreement attacks.

Chapter outline. In Section 5.1 we introduce the BDB model. Section 5.2 shows the new impossibility bounds of consensus in the BDB model. In Section 5.3 we present the Basilic class of protocols, prove its correctness and complexities, and that it also solves \diamond -consensus. Section 5.4 presents the LLB problem and ZLB, shows ZLB’s correctness and proofs and its experimental evaluation with previous works. Section 5.5 shows the zero-loss payment application in which no honest process or user loses any fund resulting from temporary disagreements. We finally conclude in Section 5.6.

5.1 Byzantine-deceitful-benign fault model

We introduce in this section formal definitions needed for our novel Byzantine-deceitful-benign (BDB) fault model.

Conflicting messages. Basilic detects and removes faulty processes that try to cause a disagreement, even if they do not succeed at causing the disagreement. For this reason, Basilic must be able to detect processes that send distinct messages to different processes where they were expected to broadcast the same message to different processes [158], we refer to these messages as conflicting. Given a protocol σ , we say that a message, or set of messages, msg sent by process p *conforms* to an execution σ_E of the protocol σ , if σ_E belongs to the set of all possible executions where p sent m and p is an honest process. Also, a faulty process p sending two messages msg, msg' *contributes* to a disagreement if there is an execution σ_E of σ such

that (i) sufficiently many faulty processes sending msg, msg' (and possibly more messages) to a disjoint subset of honest processes, one to each, leads to a disagreement, and (ii) σ_E does not lead to a disagreement without p sending msg, msg' . Two messages msg, msg' are *conflicting* with respect to σ if:

1. msg, msg' individually conform to algorithm σ for some execution $\sigma_E, \sigma_{E'}$, respectively, $\sigma_E \neq \sigma_{E'}$,
2. there is no execution $\sigma_{E''}$ of σ such that both messages together conform to $\sigma_{E''}$, and
3. if p sending msg, msg' to a disjoint subset of honest processes, one to each, contributes to a disagreement.

When combined in one message and signed by the sender, conflicting messages constitute a PoF, as we explained in Section 2.2.6.2. An example of two conflicting messages is a faulty process sending two different proposals for the same round (the proposer should only propose one value per round).

Our definition of conflicting messages differs from previous similar concepts in that conflicting messages allow for any process p to verify if two messages are conflicting: an honest process can always construct a PoF from two conflicting messages alone, but it cannot do so with all mutant messages [184], as p would need to also learn the entire execution, or with messages sent from an equivocating process [182], as these do not necessarily contribute to disagreeing.

Fault model. There are three mutually exclusive classes of faulty processes: Byzantine, deceitful and benign, in what we refer to as the *Byzantine-deceitful-benign* (BDB) failure model. Each faulty process belongs to only one of these classes. Byzantine, deceitful and benign processes are characterized by the faults they can commit. A fault is *deceitful* if it contributes to breaking agreement, in that it sends conflicting messages violating the protocol in order to lead two or more partitions of processes to a disagreement. We allow deceitful processes to constantly keep sending conflicting messages, even if they do not succeed at causing a disagreement, but instead their deceitful behavior prevents termination. As deceitful processes model processes that try to break agreement, we assume also that a deceitful fault does not send conflicting messages for rounds or phases of the protocol that it has already terminated at the time that it sends the messages. Deceitful processes can alternate between sending conflicting messages and following the protocol, but cannot deviate in any other way. A *benign* fault is any fault that does not ever send conflicting messages. Hence, benign faults cover only faults that can break termination, e.g. by crashing, sending stale messages, etc.

As usual, Byzantine processes can act arbitrarily. Thus, Byzantine processes can commit benign or deceitful faults, but they can also commit faults that are neither deceitful nor benign. A fault that sends conflicting messages and crashes afterwards is, by these definitions, neither benign nor deceitful. We denote t , d , and q as the number of Byzantine, deceitful, and benign processes, respectively. We assume that the adversary is static, in that the adversary can choose up to t Byzantine, d deceitful and q benign processes at the start of the protocol, known only to the adversary. The total number of faulty processes is thus $f = t + d + q$.

In order to distinguish benign (resp. deceitful) processes from Byzantine processes that commit a benign (resp. deceitful) fault during a particular execution of a protocol, we formalize fault tolerance in the BDB model. Let $E_\sigma(t, d, q)$ denote the set of all possible executions of a protocol σ given that there are up to t Byzantine, d deceitful and q benign processes. We say that a protocol σ for a particular problem P is (t, d, q) -*fault-tolerant* if σ solves P for all executions $\sigma_E \in E_\sigma(t, d, q)$. We abuse notation by speaking of a (t, d, q) -fault-tolerant protocol σ as a protocol that tolerates t , d and q Byzantine, deceitful and benign processes, respectively.

Note that, given a protocol σ , then $E_\sigma(0, d + k, q) \subset E_\sigma(k, d, q)$ by definition. Thus, if σ is (k, d, q) -fault-tolerant then σ is $(0, d + k, q)$ -fault-tolerant, and also $(0, d, q + k)$ -fault-tolerant. However, the contrary is not necessarily true: a protocol σ that is $(0, d + k, q)$ -fault-tolerant is not necessarily (k, d, q) -fault tolerant, as $E_\sigma(k, d, q) \not\subseteq E_\sigma(0, d + k, q)$, because Byzantine participants can commit more faults than deceitful or benign. Finally, honest processes follow the protocol, as mentioned in Section 2.2.1.

Compared to commission and omission faults, notice that not all commission faults contribute to causing disagreements. For example, some commission faults broadcast an invalid message that can be discarded. In our BDB model, this type of fault would categorize as benign, and not deceitful, since invalid messages never contribute to a disagreement, but can instead prevent termination (by only sending invalid messages that are discarded). All omission faults are however benign faults, while the contrary is also not true (as per the same aforementioned example). Compared to the alive-but-corrupt failure model [73], deceitful faults are not restricted to only contribute to a disagreement if they know the disagreement will succeed, but instead we let them try forever, even if they do not succeed. Hence, while a protocol might tolerate $d < n/3$ abc faults along with $q < n/3$ benign faults, it would not necessarily tolerate $d < n/3$ deceitful faults along with $q < n/3$ benign faults. The contrary direction always holds.

We believe thus the BDB model to be better suited for consensus, as it establishes a clear difference in the types of faults depending on the type of property that the fault jeopardizes (agreement for deceitful, termination for benign), without restricting the behavior of these faults to the cases where they are certain that they will cause a disagreement. We restate that the property of validity is defined only to rule out trivial solutions of consensus in which all processes decide a constant, and this property can be locally checked for correctness.

5.2 Impossibility of consensus in the BDB model

In this section, we extend Dwork et al.'s impossibility results [13] on the number of honest processes necessary to solve the Byzantine consensus problem in partial synchrony by adding deceitful and benign processes. First, we prove in Section 5.2.1 lower bounds on the size of the committee of any consensus protocol. Then, we prove in Section 5.2.2 lower bounds depending on the voting threshold of that protocol, which we define in the same section.

5.2.1 Impossibility bounds

First, we consider the case where $t = 0$, i.e., there are only deceitful and benign processes. In particular, we show in Lemma 5.1 that if a protocol solves consensus then it tolerates at most $d < n - 2q$ deceitful processes and $q < n/2$ benign processes. The intuition for the proof is analogous to the classical impossibility proof of consensus in partial synchrony in the presence of $t_\ell + 1$ Byzantine processes. Lemma 5.1 extends to the BDB model the classical lower bound for the BFT model [13], by tolerating a stronger adversary than the classical bound (e.g. an adversary causing $d = t_\ell$ deceitful faults and $q = t_\ell$ benign faults). By contradiction, we show that in the presence of a greater number of faulty processes than bounded by Lemma 5.1, in some executions all processes would either not terminate, or not satisfy agreement, if maintaining validity.

Lemma 5.1. Let a protocol σ solve consensus for all executions $\sigma_E \in E_\sigma(t, d, q)$ for some $t, d, q > 0$. Then, $d + t < n - 2(q + t)$.

Proof. First, we show $q < n/2$ by contradiction, as done by previous work for omission faults [13]. Suppose $q \geq n/2$, $d = 0$, $t = 0$ and consider processes are divided into a disjoint partition P, Q such that P contains between 1 and q processes and Q contains $n - |P|$. First, consider scenario A: all processes in P are benign and the rest honest, and all processes in Q propose value 0. Then, by validity all processes in Q decide 0. Then, consider scenario B: all processes in Q are benign and the rest honest, and all processes in P propose value 1. Then, by validity all processes in P decide 1. Now consider scenario C: no process is benign, and processes in P propose all 1 while processes in Q propose all 0. For processes in P scenario C is indistinguishable from scenario B, while for processes in Q scenario C is indistinguishable from scenario A. This yields a contradiction.

It follows that $q < n/2$. Hence, for $n = 2$, and since $q < 1$, it is immediate that for $d + t \geq 2$ it is impossible to solve consensus. As such, we have left to consider $d + t \geq n - 2(q + t)$ with $n \geq 3$. We will prove this by contradiction.

Consider processes are divided into three disjoint partitions P, Q, R , such that P and Q contain between 1 and $q + t$ processes each, and R contains between 1 and $d + t$. First consider the following scenario A: processes in P and R are honest and propose value 0, and processes in Q are benign. It follows that $P \cup R$ must decide value 0 at some time T_A , for if they decided 1 there would be a scenario in which processes in Q are honest and also propose 0, but messages sent from processes in Q are delivered at a time greater than T_A , having processes in $P \cup R$ already decided 1. This would break the validity property. Also, they must decide some value to satisfy termination tolerating $q + t$ benign faults.

Consider now scenario B: processes in P are benign, and processes in R and Q are honest and propose value 1. By the same approach, $R \cup Q$ decide 1 at a time T_B .

Now consider scenario C: processes in P and Q are honest, and processes in R are deceitful, the messages sent from processes in Q are delivered by processes in P at a time greater than $\max(T_A, T_B)$, and the same for messages sent from processes in P to processes in Q . Processes in P propose 0, processes in Q propose 1, and processes in R propose 0 to those in P and 1 to

those in Q . Then, for processes in P this scenario is identical to scenario A, deciding 0, while for processes in Q this is identical to scenario B, deciding 1, which leads to a disagreement. This yields a contradiction. \square

Corollary 5.1 (Impossibility of consensus with $t = 0$). It is impossible for a consensus protocol σ to tolerate d deceitful and q benign processes if $d \geq n - 2q$ or $q \geq n/2$.

Proof. This is immediate from Lemma 5.1 since σ is $(0, d, q)$ -fault-tolerant if σ solves P for all executions $\sigma_E \in E_\sigma(0, d, q)$. \square

We prove the impossibility result of Theorem 5.1 by extending the result of Corollary 5.1: it is impossible to solve consensus in the presence of t Byzantine, q benign and d deceitful processes unless $n > 3t + d + 2q$.

Theorem 5.1 (Impossibility of consensus). It is impossible for a consensus protocol to tolerate t Byzantine, d deceitful and q benign processes if $n \leq 3t + d + 2q$.

Proof. This is immediate from Lemma 5.1 since σ is (t, d, q) -fault-tolerant if σ solves P for all executions $\sigma_E \in E_\sigma(t, d, q)$. \square

5.2.2 Impossibility bounds per voting threshold

The proofs for the impossibility results of Section 5.2.1 (and for the classical impossibility results [13]) derive a trade-off between agreement and termination. In some scenarios, processes must be able to terminate without delivering messages from a number of processes that may commit benign faults. In other scenarios, processes must be able to deliver messages from enough processes before terminating in order to make sure that no disagreement caused by deceitful faults is possible. We prove in this section the impossibility results depending on this trade-off.

A protocol that satisfies both agreement and termination in partial synchrony must thus state a threshold that represents the number of processes from which to deliver messages in order to be able to terminate without compromising agreement. If this threshold is either too small to satisfy agreement, or too large to satisfy termination, then the protocol does not solve consensus. We refer to this threshold as the *voting threshold*, and denote it with h . Typically, this threshold is $h = n - t_\ell = \lceil \frac{2n}{3} \rceil$ to tolerate $t_\ell = \lceil \frac{n}{3} \rceil - 1$ Byzantine faults [23, 47, 167, 79]. We prove however in Lemma 5.2 and Corollary 5.2 that $h > \frac{d+t+n}{2}$ with $h \in (n/2, n]$ for safety.

Lemma 5.2 (Impossibility of Agreement ($t = 0$)). Let σ be a protocol with voting threshold $h \in (n/2, n]$ that satisfies agreement. Then σ tolerates at most $d < 2h - n$ deceitful processes.

Proof. The bound $h \in (n/2, n]$ derives trivially: if $h \leq n/2$ then two subsets without any faulty processes can reach the threshold for different values (Lemma 5.1). We calculate for which cases it is possible to cause a disagreement. Hence, we have two disjoint partitions of honest processes such that $|A| + |B| \leq n - d$. Suppose that processes in A and in B decide each a different decision $v_A, v_B, v_A \neq v_B$. This means that both $|A| + d \geq h$ and $|B| + d \geq h$ must hold. Adding them up, we have $|A| + |B| + 2d \geq 2h$ and since $|A| + |B| \leq n - d$ we have

$n + d \geq 2h$ for a disagreement to occur. This means that if $h > \frac{n+d}{2}$ then it is impossible for d deceitful processes to cause a disagreement. \square

The proof of Lemma 5.2 can be straightforwardly extended to include Byzantine processes, resulting in Corollary 5.2.

Corollary 5.2. Let σ be a protocol with voting threshold $h \in (n/2, n]$ that satisfies agreement. Then σ tolerates at most $d + t < 2h - n$ deceitful and Byzantine processes.

Next, in Lemma 5.3 and Corollary 5.3 we show the analogous results for the termination property. That is, we show that if a protocol solves termination while $t = 0$, then it tolerates at most $q \leq n - h$ benign processes, or $q + t \leq n - h$ benign and Byzantine processes.

Lemma 5.3 (Impossibility of Termination ($t = 0$)). Let σ be a protocol with voting threshold h that satisfies termination. Then σ tolerates at most $q \leq n - h$ benign processes.

Proof. If $n - q < h$, then termination is not guaranteed, since in this case termination would require the votes from some benign processes. This is impossible if $h \leq n - q$, as it guarantees that the threshold is lower than all processes minus the q benign processes. \square

Corollary 5.3. Let σ be a protocol with voting threshold h that satisfies termination. Then, σ tolerates at most $q + t \leq n - h$ benign and Byzantine processes.

Combining the results of corollaries 5.2 and 5.3, one can derive an impossibility bound for a consensus protocol given its voting threshold. We show this result in Corollary 5.4.

Corollary 5.4. Let σ be a protocol that solves the consensus problem with voting threshold $h \in (n/2, n]$. Then, σ tolerates at most $d + t < 2h - n$ and $q + t \leq n - h$ Byzantine, deceitful and benign processes.

We show in Figure 5.7 the threshold h to tolerate a number d of deceitful and q of benign processes. For example, for a threshold $h = \lceil \frac{5n}{9} \rceil - 1$, then $d < \frac{n}{9}$ for safety and $q < \frac{4n}{9}$ for liveness, with $t = 0$. The maximum number of Byzantine processes tolerated with $d = q = 0$ is the minimum of both bounds, being for example $t < \frac{n}{9}$ for $h = \lceil \frac{5n}{9} \rceil - 1$. In the remainder of this chapter, we assume the adversary satisfies the resilient-optimal bounds of $h \leq n - q - t$ and $h > \frac{d+t+n}{2}$, given a particular voting threshold h . The result of Theorem 5.1 holds regardless of the voting threshold. Thus, a protocol that satisfies both $h \leq n - q - t$ and $h > \frac{d+t+n}{2}$ can set its voting threshold $h \in (n/2, n]$ in order to solve consensus for any combination of t Byzantine, q benign and d deceitful processes, as long as $n > 3t + d + 2q$ holds.

5.3 Basilic, resilient-optimal consensus in the BDB model

In this section, we introduce the Basilic class of protocols, a class of resilient-optimal protocols that solve, for different voting thresholds, the actively accountable consensus problem in the BDB model. In particular, all protocols within the Basilic class tolerate t Byzantine, d deceitful and q benign processes satisfying $n > 3t + d + 2q$, and, given a particular protocol $\sigma(h)$ of the

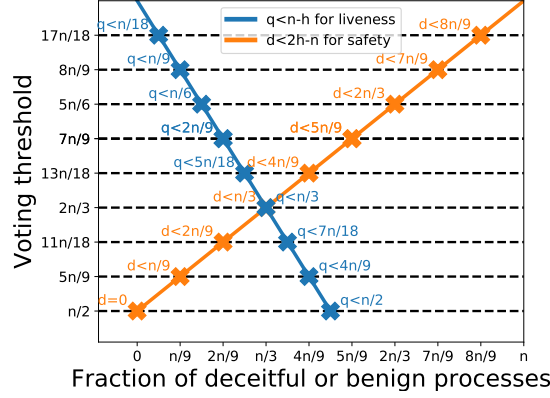


Figure 5.1: Number of deceitful processes d and benign processes q tolerated for safety and liveness, respectively, per voting threshold h and with $t = 0$ Byzantine processes.

class uniquely defined by a voting threshold $h \in (n/2, n]$, then $\sigma(h)$ tolerates a number n of processes satisfying $d + t < 2h - n$ and $q + t \leq n - h$. In this section, we first need to introduce few definitions in Section 5.3.1. Second, we present the overview of the Basilic class and show its components in Section 5.3.2.

5.3.1 Actively accountable consensus problem

The accountable consensus problem [47] includes the property of accountability in order to provide guarantees in the event that deceitful and Byzantine processes manage to cause a disagreement. This property is however insufficient for the purpose of Basilic. We need an additional property that identifies and removes all deceitful behavior that prevents termination. Faulty processes can break agreement in a finite number of conflicting messages, but once they send a pair of these conflicting messages, they leave a trace that can result in their exclusion from the system. Our goal is to exploit this trace to make sure that deceitful processes cannot contribute to breaking liveness. As a result, we include the property of active accountability, stating that deceitful faults do not prevent termination of the protocol.

Definition 5.3.1 (Actively accountable consensus problem). A protocol σ with voting threshold h solves the actively accountable consensus problem if the following properties are satisfied:

- **Termination.** Every honest process eventually decides on a value.
- **Validity.** If all honest processes propose the same value, no other value can be decided.
- **Agreement.** If $d + t < 2h - n$ then no two honest processes decide on different values.
- **Accountability.** If two honest processes output disagreeing decision values, then all honest processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.
- **Active accountability.** Deceitful behavior does not prevent liveness.

We generalize the previous definition of accountability [47] by including the voting threshold h . That is, the previous definition of accountability is the one we present in this work for the standard voting threshold of $h = 2n/3$.

5.3.2 Basilic Internals

Basilic is a class of consensus protocols, all these protocols follow the same pseudocode (Algorithms 7–10) but differ by their voting threshold $h \in (n/2, n]$. The structures of these protocols follow the classic reduction [102] from the consensus problem, which accepts any ordered set of input values, to the binary consensus problem, which accepts binary input values.

5.3.2.1 Basilic Overview

More specifically, Basilic has at its core the binary consensus protocol called *actively accountable binary consensus* or AABC for short (Algorithm 8–9) and presented in Section 5.3.2.3. We show in Figure 5.2 an example execution with $n = 4$ processes in the committee. First, each process p_i selects their input value v_i , which they share with everyone executing an instance of a reliable broadcast protocol called *actively accountable reliable broadcast* or AARB for short (Algorithm 10). Then, processes execute one instance $AABC_i$ of the binary consensus protocol for each process p_i to decide whether to select the associated input value from process p_i . Finally, processes locally process the minimum input value from the values whose associated AABC instance output 1.

This Basilic binary consensus protocol shares similarities with Polygraph [47], as it also detects guilty processes, but goes further, by excluding these detected processes and adjusting its voting threshold at runtime to solve consensus even in cases where Polygraph cannot ($n/3 \leq t + q + d$). We will summarize the comparison of Basilic with the state of the art in Table 5.2 (Section 5.3.5.2). Similarly to Polygraph, Basilic can perform the superblock optimization [23, 28] to solve SBC (Definition 2.2.4) simply by deciding the union of both v_0 and v_2 in the example, instead of the minimum. This provides a better normalized communication complexity of the protocol (per decision). Finally, the rest of the reduction is depicted in Algorithm 7 and invokes n actively accountable reliable broadcast instances or AARB (Algorithm 10) described in Section 5.3.2.4, followed by n of the aforementioned AABC instances.

Certificates and transferable authentication. Basilic uses certificates in order to validate or discard a message, and also to detect deceitful processes by cross-checking certificates. A certificate is a list of previously delivered and signed messages that justifies the content of the message on which the certificate is piggybacked. Thus, honest processes perform transferable authentication [182]. That is, process p_i can deliver msg from p_j by verifying the signature of msg , even if msg was received from p_k , for $k \neq i \neq j$.

Detected deceitful processes. A key novelty of Basilic is to remove detected deceitful processes from the committee at runtime. For this reason, we refer to d_r as the number of detected deceitful processes, and define a voting threshold $h(d_r)$ that varies with the number of detected deceitful processes. Therefore, processes start Basilic with an initial voting threshold

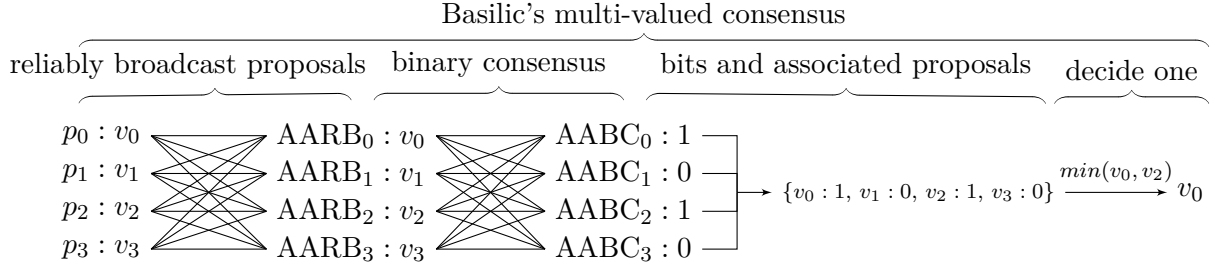


Figure 5.2: Basilic execution example for a committee of $n = 4$ processes. First, each process p_i selects their input value v_i , which they share with everyone executing their respective instance $AARB_i$ of $AARB$. Then, processes execute one instance $AABC_i$ of the binary consensus protocol to decide whether to select the associated input value from process p_i . Finally, processes locally process the minimum input value from the values whose associated $AABC$ instance output 1.

$h(d_r = 0) = h_0$, e.g., $h_0 = \lceil \frac{2n}{3} \rceil$, but then update the threshold by removing detected deceitful processes, i.e. $h(d_r) = h_0 - d_r$. This way, detected deceitful processes break neither liveness nor safety, as we will show. Certificates must always contain $h(d_r)$ signatures from distinct processes justifying the message (after filtering out up to d_r signatures from detected deceitful processes), or else they will be discarded. Recall that the adversary is thus constrained to the bounds from Corollary 5.4 depending on the voting threshold. As Basilic uses a threshold that updates at runtime starting from an initial threshold $h(d_r) = h_0 - d_r$, we restate these bounds applied to the initial threshold $h_0 \leq n - q - t$ and $h_0 > \frac{d+t+n}{2}$, or to the updated threshold $h(d_r) \leq n - q - t - d_r$ and $h(d_r) > \frac{d+t+n}{2} - d_r$.

5.3.2.2 The general Basilic protocol

We bring together the n instances of the $AABC$ protocol with the n instances of the $AARB$ protocol in Algorithm 7, where we show the general Basilic protocol. The protocol derives from Polygraph's general protocol [47], which in turn derives from DBFT's multi-valued consensus protocol [23]. Note that, as is the case for Polygraph's and DBFT's pseudocode, Algorithms 7,8 assume that each call to $AABC\text{-prop}_i$ is concurrent (non-blocking). We omit the specification of access to shared resources between concurrent executions and note instead that the shown algorithms may not have strictly sequential semantics.

Honest processes first start their respective $AARB$ instance (for which they each are the source) by proposing a value in line 2. Delivered proposals are stored in a message msg with the index corresponding to the source of the proposal. A binary consensus at index k is started with input value 1 for each index k where a proposal has been recorded (line 6). Notice that we can guarantee to decide 1 on at most $h(d_r)$ proposals (line 7), where d_r can be up to d and is set by update-committee in Algorithm 9, meaning that, for the standard threshold $h(d_r) = \lceil \frac{2n}{3} \rceil - d_r$, the minimum number of decided proposals is $\lceil \frac{n}{3} \rceil$, since $d_r < \frac{n}{3}$. Once honest processes decide 1 on at least $h(d_r)$ $AABC$ instances, honest processes start the remaining $AABC$ instances with input value 0 (line 9), without having to wait to $AARB$ -deliver their respective values.

Finally, once all AABC instances have terminated (line 10), honest processes can output a decision. As such, processes take as input a list of AARB-delivered values and their associated index and output a decision selecting the AARB-delivered value with the lowest associated index whose binary consensus with the same index output 1 (line 13).

Algorithm 7 The general Basilic with initial threshold h_0 .

```

1: Basilic-gen-propose $_{h_0}(v_i)$ :
2:    $msgs \leftarrow \text{AARB-broadcast}(\text{EST}, \langle v_i, i \rangle)$   $\triangleright$  deliver 'EST' messages, Algorithm 10
3:   repeat:
4:     if  $(\exists v, k : (\text{EST}, \langle v, k \rangle) \in msgs)$  then  $\triangleright$  proposal AARB-delivered
5:       if  $(\text{BIN-CONSENSUS}[k] \text{ not yet invoked})$  then  $\triangleright$  Algorithm 8
6:          $\text{bin-decisions}[k] \leftarrow \text{BIN-CONSENSUS}[k].\text{AABC-prop}(1)$   $\triangleright$  Non-blocking concurrent call in dedicated process
7:       until  $|\text{bin-decisions}[k]| = 1 \geq h(d_r)$   $\triangleright$  decide 1 on at least  $h(d_r)$ 
8:       for all  $k$  such that  $\text{BIN-CONSENSUS}[k]$  not yet invoked do
9:          $\text{bin-decisions}[k] \leftarrow \text{BIN-CONSENSUS}[k].\text{AABC-prop}(0)$ 
10:      wait until for all  $k$ ,  $\text{bin-decisions}[k] \neq \perp$ 
11:       $j \leftarrow \min\{k : \text{bin-decisions}[k] = 1\}$   $\triangleright$  select minimum value (or union for SBC)
12:      wait until  $\exists v : (\text{EST}, \langle v, j \rangle) \in msgs$ 
13:      decide  $v$ 

```

5.3.2.3 Actively accountable binary consensus

We show in Algorithm 8 the Basilic *actively accountable binary consensus* (AABC) protocol with initial threshold $h_0 \in (n/2, n]$, along with some additional components and functions in Algorithm 9. First, note that all delivered messages are correctly signed (as wrongly signed messages are discarded) and stored in sig_msgs , along with all sent messages (as we detail in Rule 3 of Algorithm 8).

The Basilic's AABC protocol is divided in two phases, after which a decision is taken. A key difference with Polygraph is that when a timer for one of the two phases reaches its timeout, if a process cannot terminate that phase yet, then it broadcasts its set of signed messages for that phase and resets the timer, as detailed in Rule 4. This allows Basilic to prevent deceitful processes from breaking termination by trying to cause a disagreement and never succeeding. For example, for $n = 4$ and $h_0 = \lceil 2n/3 \rceil = 3$, if $q = 1$ and $d = 1$, the deceitful process could prevent the 2 honest processes from terminating by constantly sending them conflicting messages, even if none of these honest processes will reach the threshold for the disagreeing values. Thus, once the timer is reached, processes exchange their known set messages and can update the committee removing processes that sent conflicting messages. It is important that processes wait for this timer before taking a decision for the phase, or before exchanging signed messages, since only waiting for that increasing timer guarantees that all sent messages will be received before the timer reaches its timeout, after GST (instead the timer can be left unchanged assuming synchronization of processes' internal clocks). Each process maintains an estimate (line 15), initially given as input, and then proceeds in rounds executing the following phases:

Algorithm 8 Basilic's AABC with initial threshold h_0 for p_i .

```

14: AABC-proph0( $v_i$ ):
15:    $est \leftarrow v_i$ 
16:    $r \leftarrow 0$ 
17:    $timeout \leftarrow 1$ 
18:    $cert[0] \leftarrow \emptyset$ 
19:    $bin\_vals \leftarrow \emptyset$ 
20:   repeat:
21:      $r \leftarrow r + 1$ 
22:      $timeout \leftarrow timeout \cdot 2\Delta$   $\triangleright$  set timer, for termination after GST
23:      $coord \leftarrow ((r - 1) \bmod n) + 1$   $\triangleright$  rotate coordinator
24:     ► Phase 1:
25:      $timer \leftarrow \text{start-timer}(timeout)$   $\triangleright$  start timer
26:      $\text{abv-broadcast}(\text{EST}[r], est, cert[r - 1], i, bin\_vals)$ 
27:     if ( $i = coord$ ) then
28:       wait until  $bin\_vals[r] = \{w\}$ 
29:        $\text{broadcast}(\text{COORD}[r], w)$ 
30:       wait until  $bin\_vals[r] \neq \emptyset \wedge timer$  expired
31:     ► Phase 2:
32:      $timer \leftarrow timeout$   $\triangleright$  reset timer
33:     if ( $((\text{COORD}[r], w) \in sig\_msgs \wedge w \in bin\_vals[r])$ ) then
34:        $aux \leftarrow \{w\}$   $\triangleright$  prioritize coordinator's value
35:     else  $aux \leftarrow bin\_vals[r]$   $\triangleright$  else use any received value
36:      $\text{broadcast}(\text{ECHO}[r], aux)$   $\triangleright$  broadcast signed ECHO message
37:     wait until ( $vals = \text{comp-vals}(sig\_msgs, bin\_vals, aux) \neq \emptyset \wedge timer$  expired)
38:     ► Decision phase:
39:     if ( $|vals| = 1$ ) then  $est \leftarrow vals[0]$   $\triangleright$  if only one, adopt as estimate
40:     if ( $(est = (r \bmod 2) \wedge p_i$  not decided before) then
41:        $\text{decide}(est); \text{return } est$   $\triangleright$  if parity matches, decide the estimate
42:     else  $est \leftarrow (r \bmod 2)$   $\triangleright$  otherwise, the estimate is the round's parity bit
43:      $cert[r] \leftarrow \text{compute-cert}(vals, est, r, bin\_vals, sig\_msgs)$ 
44:   Upon receiving a signed message  $s\_msg$ :
45:      $pofs \leftarrow \text{check-conflicts}(\{s\_msg\}, sig\_msgs)$   $\triangleright$  returns  $\emptyset$  or PoFs
46:      $\text{update-committee}(pofs)$   $\triangleright$  remove fraudsters
47:   Upon receiving a certificate  $cert\_msg$ :
48:      $pofs \leftarrow \text{check-conflicts}(cert\_msg, sig\_msgs)$   $\triangleright$  returns  $\emptyset$  or PoFs
49:      $\text{update-committee}(pofs)$   $\triangleright$  remove fraudsters
50:   Upon receiving a list of PoFs  $pofs\_msg$ :
51:     if ( $\text{verify-pofs}(pofs\_msg)$ ) then  $\triangleright$  if proofs are valid then
52:        $\text{update-committee}(pofs\_msg)$   $\triangleright$  remove fraudsters from committee
53:   Rules:

```

1. Every message that is not properly signed by the sender is discarded.
 2. Every message that is sent by **abv-broadcast** without a valid certificate after Round 1, except for messages with value 1 in Round 2, are discarded.
 3. Every signed message received is stored in sig_msgs , including messages within certificates.
 4. Every time the timer reaches the timeout for a phase, and if that phase cannot be terminated, processes broadcast their current delivered signed messages for that phase (and all messages received for future phases and rounds) and reset the timer for that phase. These messages are added to the local set of messages and cross-checked for PoFs on arrival.
-

1. In the first phase, each process broadcasts its estimate (given as input) via an accountable binary value reliable broadcast (ABV-broadcast) (line 26), which we present in Algorithm 9, lines 67–82 and discuss in Section 5.3.2.3. Decision and **abv-broadcast** messages are discarded unless they come with a certificate justifying them.

The protocol also uses a rotating coordinator (line 23) per round which carries a special **COORD** message (lines 27–29). All processes wait until they deliver at least one message from the call to **abv-broadcast** and until the timer, initially set to Δ , expires (line 30). (Note that the bound on the message delays remains unknown due to the unknown GST.) If a process delivers a message from the coordinator (line 33), then it broadcasts an **ECHO** message with the coordinator’s value and signature in the second phase (line 36). Otherwise, it echoes all the values delivered in phase 1 as part of the call to **abv-broadcast** (line 35).

2. In the second phase, processes wait till they receive $h(d_r)$ **ECHO** messages, as shown in the call to **comp-vals** (line 37), which returns the set of values that contain these $h(d_r)$ signed **ECHO** messages. Function **comp-vals** is depicted in Algorithm 9 (lines 83–92). Processes then try to come to a decision in lines 39–43. As it was the case for phase 1, when the timer expires in phase 2, all processes broadcast their current set of **ECHO** messages. Then, they update their committee if they detect deceitful processes through PoFs (lines 44–52) and recheck if they reach the updated $h(d_r)$ threshold, after which they reset the timer.

3. During the decision phase, if there is just one value returned by **comp-vals** and that value’s parity matches with the round’s parity, process p_i decides it (line 41) and broadcasts the associated certificate in the call to **compute-cert**. If the parity does not match then process p_i simply adopts the value as the estimate for the next round (line 39). If instead there is more than one value returned by **comp-vals** then p_i adopts the round’s parity as next round’s estimate (line 42). Adopting the parity as next round’s estimate helps with convergence in the next round, in this case where processes are hesitating between two values. The call to **compute-cert** (depicted at lines 93–102 of Algorithm 9) gathers the signatures justifying the current estimate and broadcasts the certificate if the estimate was decided in this round.

Detecting and removing deceitful processes. Upon receiving a signed message, honest processes check if the received message conflicts with some previously delivered message in storage in *sig_msgs* by calling **check-conflicts** (line 45). This function returns $pofs = \emptyset$ if there are no conflicting messages, or a list *pofs* of PoFs otherwise. Then, at line 46, honest processes call **update-committee** (depicted at lines 54–66 of Algorithm 9) to remove the $|pofs|$ detected deceitful processes at runtime. In the call to **update-committee**, process p_i removes all processes that are proven deceitful via new PoFs, and updates the committee N , its size n , and the voting threshold $h(d_r)$. After that, p_i rechecks all delivered messages in that phase in case it can now terminate the phase with the new threshold $h(d_r)$ (and after filtering out messages delivered by the d_r removed deceitful processes) by calling **recheck-certs-termination()** in line 65 of Algorithm 9. Finally, it resets the timer for the current phase by calling **reset-current-timer()** in line 66 of Algorithm 9.

Algorithm 9 Helper components.

```

54: update-committee(new_pofs): ▷ function that removes fraudsters
55:   if (new_pofs ≠ ∅ ∧ new_pofs ⊈ local_pofs) then
56:     new_pofs ← new_pofs \ local_pofs ▷ consider only new PoFs
57:     local_pofs ← local_pofs ∪ new_pofs ▷ store new PoFs
58:     broadcast(POF, new_pofs) ▷ broadcast new PoFs
59:     new_deceitful ← new_pofs.get_processes() ▷ get deceitful from PoFs
60:     new_deceitful ← new_deceitful \ local_deceitful
61:     local_deceitful ← local_deceitful ∪ new_deceitful
62:     N ← N \ {new_deceitful}; n ← |N| ▷ remove new deceitful
63:     dr ← |local_deceitful| ▷ update number of detected deceitful
64:     h(dr) ← recalculate-threshold(N, dr)
65:     recheck-certs-termination() ▷ check termination of current phase
66:     reset-current-timer() ▷ reset timer of current phase

67: abv-broadcast(MSG, val, cert, i, bin_vals):
68:   broadcast(BVECHO, ⟨val, cert, i⟩) ▷ broadcast message
69:   if (r = 3 or (r = 2 and val = 1)) then discard all messages received without a valid certificate
70:   Upon receipt of (BVECHO, ⟨v, ·, j⟩)
71:     if ((BVECHO, ⟨v, ·, ·⟩) received from  $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$  processes and BVECHO, ⟨v, ·, i⟩ not broadcast) then
72:       Let cert be any valid certificate cert received in these messages
73:       broadcast(BVECHO, ⟨v, cert, i⟩)
74:     if ((BVECHO, ⟨v, ·, ·⟩) received from h(dr) processes and (BVREADY, ⟨v, ·, ·⟩) not yet broadcast) then
75:       Let cert be any valid certificate cert received in these messages
76:       Construct bv_cert a certificate with h(dr) signed BVECHO
77:       bin_vals ← bin_vals.add(BVREADY, ⟨v, cert, j, bv_cert⟩)
78:       broadcast(BVREADY, ⟨v, cert, j, bv_cert⟩)
79:     if ((BVREADY, ⟨v, cert, j, bv_cert⟩) received from 1 process) then
80:       bin_vals ← bin_vals.add(BVREADY, ⟨v, cert, j, bv_cert⟩)
81:     if ((BVREADY, ⟨v, cert, j, bv_cert⟩) not yet broadcast) then
82:       broadcast(BVREADY, ⟨val, cert, i, bv_cert⟩)

83: comp-vals(msgs, b_set, aux_set): ▷ check for termination of phase 2
84:   If ∃ S ⊆ msgs where the following conditions hold:
85:     (i) |S| contains h(dr) distinct ECHO[r] messages
86:     (ii) aux_set is equal to the set of values in S ▷ h(dr) with same est
87:   then return(aux_set)
88:   Else If ∃ S ⊆ msgs where the following conditions hold:
89:     (i) |S| contains h(dr) distinct ECHO[r] messages
90:     (ii) Every value in S is in b_set ▷ h(dr) messages with different est
91:   then return(V = the set of values in S)
92:   Else return(∅) ▷ else not ready to terminate

93: compute-cert(vals, est, r, bin_vals, msgs): ▷ compute and send cert
94:   if (est = (r mod 2)) then
95:     if (r > 1) then
96:       to_return ← (cert : (EST[r], ⟨v, cert, ·⟩) ∈ bin_vals)
97:     else to_return ← (∅)
98:   else to_return ← (h(dr) signed msgs containing only est)
99:   if (vals = {(r mod 2)} ∧ no previous decision by pi) then
100:     cert[r] ← h(dr) signed messages containing only r mod 2
101:     broadcast(est, r, i, cert[r]) ▷ broadcast decision
102:   return(to_return)

```

Termination and agreement of Basilic’s AABC. We show the detailed proofs of agreement and termination in Lemmas 5.17 and 5.19. The idea is that removing deceitful processes has no effect on agreement, while it facilitates termination, since the threshold $h(d_r) = h_0 - d_r$ decreases the initial threshold h_0 with the number of removed deceitful processes. Also, since all honest processes broadcast their delivered PoFs and thanks to the property of accountability, eventually all honest processes agree on the same set of removed deceitful processes.

Then, if a process p_i terminates broadcasting certificate $cert_i$ while another process p_j already removed newly detected deceitful processes new_d_r present in $cert_i$, then $|cert_i| - new_d_r \geq h(d_r + new_d_r)$ by construction. As such, either an honest process terminates and then all subsequent honest processes can terminate, even after removing more deceitful processes, or honest processes eventually reach a scenario where all deceitful processes are detected $d_r = d$ and removed, after which honest processes terminate.

Note that removing processes at runtime can result in rounds whose coordinator is already removed. For the sake of correctness, we do not change the coordinator for that round even if it has already been removed. This guarantees that all honest processes eventually reach a round in which they all agree on the same coordinator, which is an honest process. If this round is the first after GST and after all deceitful processes have been removed from the committee, then honest processes will reach agreement.

Accountable binary value broadcast. The ABV-broadcast that we present in Algorithm 9 is inspired from the E protocol presented by Malkhi et al. [185] and the binary broadcast presented in Polygraph [47]. If honest processes add a value v to bin_vals (lines 77 and 80) as a result of the ABV-broadcast, we say that they *ABV-deliver* v . Processes exchange BVECHO and BVREADY messages during ABV-broadcast. BVECHO messages are signed and must come with a valid certificate $cert_i$ justifying the value, as shown in lines 68 and 73. BVREADY messages carry the same information as BVECHO messages plus an additional certificate bv_cert containing $h(d_r)$ BVECHO messages justifying the BVREADY message, constructed in line 76. This way, as soon as a process receives a BVREADY message with a value (line 79), it already obtains $h(d_r)$ BVECHO messages too, meaning it can ABV-deliver that value adding it to bin_vals (lines 77 and 80). Honest processes broadcast signed BVECHO messages for their estimate (line 68) and for all values for which they receive at least $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ signed BVECHO messages from distinct processes. Waiting for this many BVECHO messages for a value v guarantees that all honest processes ABV-deliver v , as we show in Section 5.3.4.

In particular, we show that our ABV-broadcast satisfies the following properties: (i) ABV-Termination, in that every honest process eventually adds at least one value to bin_vals ; (ii) ABV-Uniformity, in that honest processes eventually add the same values to bin_vals ; (iii) ABV-Obligation, in that if $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ honest processes ABV-broadcast a value v , then all honest processes ABV-deliver v ; (iv) ABV-Justification, in that if an honest process ABV-delivers a value v then v was ABV-broadcast by an honest process; and (v) ABV-Accountability, in that every ABV-delivered value contains a valid certificate from the previous round.

We show in Lemma 5.4 that Basilic’s AABC satisfies AABC-active accountability, but we defer the rest of the proofs of actively accountable binary consensus to Section 5.3.4.

Lemma 5.4 (AABC-Active accountability). Basilic’s AABC satisfies active accountability.

Proof. We show that if a faulty process p_i sends two conflicting messages to two subsets $A, B \subseteq N$, each containing at least one honest process, then eventually all honest processes terminate, or instead they receive a PoF for p_i and remove it from the committee, after which they all terminate. The proof is analogous if there are instead more than two conflicting messages and subsets containing at least one honest process.

First, we observe that no process gets stuck in some round. Process p_i cannot get stuck in phase 1 since, by ABV-Termination (Lemma 5.5), every honest process eventually ABV-delivers a value.

A process also does not get stuck waiting on phase 2. First, notice that every value that is included in an ECHO message from an honest process is eventually delivered to bin_vals . Then, note that all honest processes eventually deliver $h(d_r)$ ECHO messages, or instead, when the timer expires, processes will exchange their ECHO messages and be able to construct PoFs and remove d_r deceitful processes that are preventing termination. In the latter case, after removing all deceitful processes from the committee and updating the threshold, they will deliver enough ECHO messages to terminate phase 2, since $h(d_r) \leq n - q - t - d$ for $d_r = d$. Note that there is no need for any assumption other than partial synchrony for this to be guaranteed (i.e. all messages eventually get delivered).

Then, we show that all honest processes always hold a valid certificate to broadcast a proper message, which could otherwise prevent termination during the ABV-broadcast in phase 1. For an estimate whose parity is the same as that of the finished round $r - 1$, process p_i must have received a valid certificate for the round (otherwise it would not have terminated such round). If the parity matches, then it can always construct a valid certificate from the delivered estimates in round $r - 1$.

As a result, all processes always progress infinitely in every round. Consider the first round r after GST where (i) the coordinator is honest and (ii) all deceitful processes have been detected and removed by all honest processes. In this case, every honest process will prioritize the coordinator’s value, adopting it as their ECHO message adding only that value. Hence, every process adopts the same value, and decides either in round r or round $r + 1$ (by Lemma 5.16). \square

5.3.2.4 Actively accountable reliable broadcast

Algorithm 10 shows Basilic’s *actively accountable reliable broadcast* (AARB). The protocol is analogous to the secure broadcast presented in previous work [185], with the difference that we also introduce a timer that honest processes use to periodically broadcast their set of delivered ECHO messages, in order to detect deceitful processes. The protocol starts when the source broadcasts an INIT message with its proposed value v (line 104). Upon delivering that message, all honest processes also broadcast a signed ECHO message with v (line 106). Then, once a process p_i delivers $h(d_r)$ distinct signed ECHO messages for the same value v , p_i first broadcasts a READY message (line 109) with a certificate containing the $h(d_r)$ ECHO messages justifying v (constructed in line 108), and then AARB-delivers the value (line 110). The same occurs if

instead a process delivers just one valid READY message containing a valid certificate justifying it in lines 111-115.

As it occurs with Basilic's AABC protocol presented in Algorithms 8 and 9, upon cross-checking newly received signed messages with previously delivered ones (lines 117 and 120), honest processes can detect deceitful faults and update the committee (lines 118 and 121), removing them at runtime, by calling `update-committee`. This can also occur when receiving a list of PoFs (line 122). Note that this is the same call to the same function as in the AABC protocol shown in Algorithm 8, because honest processes update the committee across the entire Basilic protocol, and not just for that particular instance of AARB or AABC where the deceitful process was detected. We show in Section 5.3.4 that Basilic's AARB protocol satisfies the following properties of actively accountable reliable broadcast:

- **AARB-Unicity.** honest processes AARB-deliver at most one value.
- **AARB-Validity.** honest processes AARB-deliver a value if it was previously AARB-broadcast by the source.
- **AARB-Send.** If the source is honest and AARB-broadcasts v , then honest processes AARB-deliver v .
- **AARB-Receive.** If an honest process AARB-delivers v , then all honest processes AARB-deliver v .
- **AARB-Accountability.** If two honest processes AARB-deliver distinct values, then all honest processes receive PoFs of the deceitful behavior of at least $2h(d_r) - n$ processes including the source.
- **AARB-Active accountability.** Deceitful behavior does not prevent liveness.

5.3.3 Basilic's fault tolerance in the BDB model

We show in Figure 5.3a the combinations of Byzantine, deceitful and benign processes that Basilic tolerates, depending on the initial threshold h_0 . The solid lines represent the variation in tolerance to benign and deceitful processes as the number of Byzantine processes varies for a particular threshold. For example, for $h_0 = \frac{2n}{3}$, if $t = 0$ then $d < \frac{n}{3}$ and $q < \frac{n}{3}$. As t increases, for example to $t = \lceil \frac{n}{6} \rceil - 1$, then $d < \frac{n}{6}$ and $q < \frac{n}{6}$.

We compare our Basilic's fault tolerance with that of previous works in Figure 5.3b. In particular, we represent multiple values of the initial threshold $h_0 \in \{5n/9, 2n/3, 3n/4, 5n/6\}$ for Basilic. First, we show that classical Byzantine fault-tolerant (BFT) protocols tolerate only the case $t < n/3$ with a triangle dot (\blacktriangle) in the figure. This is the case of most partially synchronous BFT consensus protocols [23, 47, 167, 79]. Second, we represent Flexible BFT [73] in their greatest fault tolerance setting in partial synchrony. As we can see, such setting overlaps with Basilic's initial threshold of $h_0 = 2n/3$. However, the difference lies in that while Basilic tolerates all the cases in the solid line $h_0 = 2n/3$, Flexible BFT only tolerates a particular dot of the line, set at the discretion of each user. That is, Flexible BFT's users must decide, for

Algorithm 10 Basilic's AARB with initial threshold h_0 .

```

103: AARB-broadcast $h_0$ ( $v_i$ ):                                ▷ executed by the source
104:   broadcast(INIT,  $v_i$ )                                     ▷ broadcast to all
105:   Upon receiving (INIT,  $v_i$ ) from  $p_j$  and not having sent ECHO:
106:     broadcast(ECHO,  $v, j$ )                                  ▷ echo value to all
107:   Upon receiving  $h(d_r)$  (ECHO,  $v, j$ ) and not having sent a READY:
108:     Construct  $cert_i$  containing at least  $h(d_r)$  signed msgs (ECHO,  $v, j$ )
109:     broadcast(READY,  $v, cert_i, j$ )                          ▷ broadcast certificate
110:     AARB-deliver( $v, j$ )                                       ▷ AARB-deliver value
111:   Upon receiving (READY,  $v, cert, j$ ), and not having sent a READY:
112:     if (verify( $cert$ ) = False) then continue
113:     Set  $cert_i$  to be one of the valid certs received (READY,  $v, cert, j$ )
114:     broadcast(READY,  $v, cert_i, j$ )                          ▷ broadcast certificate
115:     AARB-deliver( $v, j$ )                                       ▷ AARB-deliver value
116:   Upon receiving a signed message  $s\_msg$ :
117:      $pofs \leftarrow$  check-conflicts( $\{s\_msg\}, sig\_msgs$ )      ▷ returns  $\emptyset$  or PoFs
118:     update-committee( $pofs$ )                                   ▷ remove fraudsters
119:   Upon receiving a certificate  $cert\_msg$ :
120:      $pofs \leftarrow$  check-conflicts( $cert\_msg, sig\_msgs$ )      ▷ returns  $\emptyset$  or PoFs
121:     update-committee( $pofs$ )                                   ▷ remove fraudsters
122:   Upon receiving a list of PoFs  $pofs\_msg$ :
123:     if (verify-pofs( $pofs\_msg$ )) then                        ▷ if proofs are valid then
124:       update-committee( $pofs\_msg$ )                            ▷ exclude from committee
125: Rules:
    1. Processes broadcast their current delivered signed INIT and ECHO messages once a timer timer, initially
       set to  $\Delta$ , reaches 0, and reset the timer to  $\Delta$ .

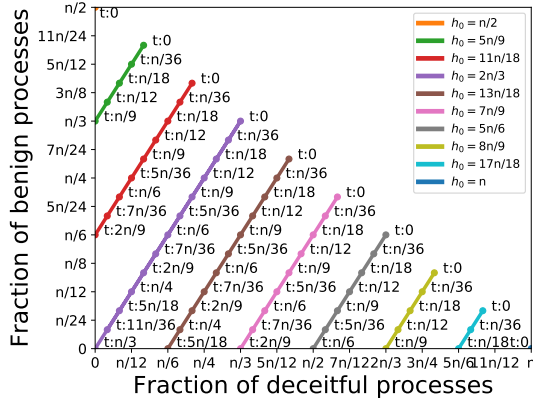
```

example, whether they tolerate either $\lceil 2n/3 \rceil - 1$ total faults, being none of them Byzantine, or instead tolerate $\lceil n/3 \rceil - 1$ Byzantine faults, not tolerating any additional fault. Basilic can however tolerate any range satisfying both $h_0 > \frac{n+d+t}{2}$ for safety and $h_0 \leq n - q - t$ for liveness, which allows our users and processes to tolerate significantly more combinations of faults for one particular threshold $h_0 \in (n/2, n]$. For this reason, we represent the line of Flexible BFT as a dashed line, whereas Basilic's lines are solid. For each initial voting threshold h_0 , the maximum number of Byzantine processes Basilic tolerates is $t < \min(2h_0 - n, n - h_0)$, which is obtained by setting $q = d = 0$ and resolving both bounds for safety and liveness.

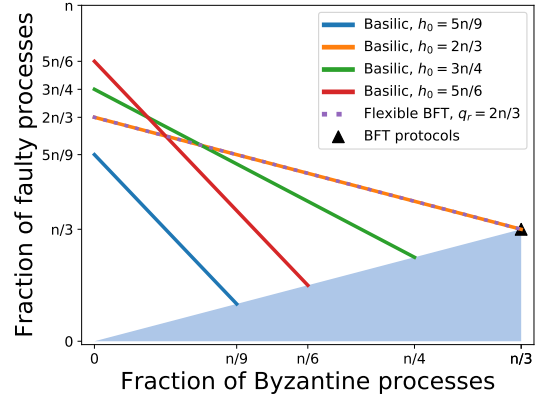
5.3.4 Basilic's correctness

In this section, we prove the properties of Basilic, including its ABV-broadcast, AABC and AARB protocols. We summarize all proofs in the result shown in Theorem 5.2 to show that Basilic protocol with initial threshold h_0 solves consensus if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$. This result translates in the Basilic class of protocols solving consensus if $n > 3t + d + 2q$, as we show in Corollary 5.5.

Theorem 5.2 (Consensus per threshold). The Basilic protocol with initial threshold h_0 solves the actively accountable consensus problem if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$.



(a) Combinations of benign, deceitful and Byzantine processes that Basilic tolerates, for an initial threshold h_0 .



(b) Fraction of faulty processes, compared with fraction of Byzantine processes, for a particular initial threshold h_0 of the general Basilic protocol, compared with other works.

Figure 5.3

Corollary 5.5 (Consensus). The Basilic class of protocols solves actively accountable consensus if $n > 3t + d + 2q$.

5.3.4.1 Accountable binary value broadcast

We first start with the properties that ABV-broadcast satisfies. We say process p_i ABV-broadcasts value v to refer to p_i sending a BVECHO message containing v and a valid certificate justifying v . We prove ABV-termination in Lemma 5.5, ABV-uniformity in Lemma 5.6, ABV-obligation in Lemma 5.7, ABV-justification in Lemma 5.8, and ABV-accountability in Lemma 5.9.

Lemma 5.5 (ABV-Termination). Every non-faulty process eventually adds at least one value to bin_vals .

Proof. Note that all non-faulty processes broadcast a BVECHO message with value v when they receive $\lfloor \frac{n-q}{2} \rfloor - d_r + 1$ BVECHO messages with v . First, let us consider that $t = d = 0$, in that case, non-faulty processes broadcast a BVECHO message with v if they receive $\lfloor \frac{n-q-t}{2} \rfloor + 1$ BVECHO messages with v . Also recall that $v \in \{0, 1\}$. As such, let us consider a partition of non-faulty processes $A, B \subseteq N$ such that $A \cap B = \emptyset$, and let us consider that processes in A initially sent a BVECHO message with $v = 0$ while processes in B sent a BVECHO message with $v = 1$. It is clear that $|A| + |B| \geq n - q - t$ and thus either $|A| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$ or $|B| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$. W.l.o.g. let us assume that $|A| \geq \lfloor \frac{n-q-t}{2} \rfloor + 1$, then processes in B eventually receive enough BVECHO messages with value $v = 0$ to also broadcast a BVECHO message with $v = 0$. Thus, since $h(d_r) \leq n - q - t - d_r$, eventually all non-faulty processes receive enough BVECHO messages to add at least the value 0 to bin_vals .

Suppose instead that $d > 0$ and $t = 0$. Then, if the $d_r \leq d$ deceitful processes that behave deceitful at a particular phase are enough to prevent termination, this means that d_r processes

have sent at least two conflicting messages to at least two non-faulty processes. As such, when the timer expires and non-faulty processes broadcast their received signed BVECHO messages, all non-faulty processes will eventually receive enough BVECHO messages to send a BVECHO message. Thus, the case $d > 0$ is analogous to the case $d = 0$ since BVECHO messages are relayed when the timer expires, and we have proven in the previous paragraph that termination is guaranteed in that case. The same analogy takes place if $t > 0$.

Note additionally that if d_r detected deceitful processes have been removed, then the thresholds decrease by the same factor d_r , preserving termination. □

Lemma 5.6 (ABV-Uniformity). If a non-faulty process p_i adds value v to the set bin_vals , then all other non-faulty processes also eventually add v to their local set bin_vals .

Proof. This proof is straightforward: p_i adds v to the set bin_vals if it holds $h(d_r)$ signed BVECHO messages with v . In that case, it also constructs a certificate bv_cert with these messages and broadcasts bv_cert as part of the BVREADY with v before adding v to bin_vals . Therefore, all other non-faulty processes will eventually receive p_i 's BVREADY message along with bv_cert containing enough BVECHO messages to also add v to their local bin_vals . Finally, recall that all non-faulty processes broadcast their BVREADY message before adding v to bin_vals , which solves the case that p_i is faulty and sends BVREADY only to a subset of the non-faulty processes. □

Lemma 5.7 (ABV-Obligation). If $\lfloor \frac{n-q-t}{2} \rfloor - d_r + 1$ non-faulty processes ABV-broadcast a value v , then all non-faulty processes ABV-deliver v .

Proof. This proof is analogous to that of Lemma 5.5. □

Lemma 5.8 (ABV-Justification). If process p_i is non-faulty and ABV-delivers v , then v has been ABV-broadcast by some non-faulty process.

Proof. Assume first $t = 0$ and suppose the contrary: p_i ABV-delivers v and all non-faulty processes ABV-broadcast v' , $v \neq v'$. Since benign processes may either send v' to a subset of the non-faulty processes or nothing at all, this means that $d - d_r > \lfloor \frac{n-q}{2} \rfloor - d_r + 1$ for deceitful alone to be able to make p_i ABV-deliver v . But using the bound $d - d_r < n - h(d_r)$ we obtain that $q \geq 2h(d_r) - n$, which contradicts our assumption on the number of benign faults (i.e. the bound $q < 2h(d_r) - n$). As a result, it follows that at least some non-faulty process must have ABV-broadcast v . The prove is analogous if $t > 0$. □

Lemma 5.9 (ABV-Accountability). If process p_i adds value v to bin_vals then associated with v is a valid certificate $cert$ from the previous round.

Proof. Since every BVECHO and BVREADY message without a valid certificate is discarded, it follows immediately that when a value v is added to bin_vals then p_i has access to a valid certificate. □

5.3.4.2 Actively accountable reliable broadcast

In this section, we prove the properties of Basilic's reliable broadcast, AARB. We prove AARB-unicity in Lemma 5.10, AARB-validity in Lemma 5.11, AARB-send in Lemma 5.12, AARB-receive in Lemma 5.13, AARB-accountability in Lemma 5.14 and AARB-active accountability in Lemma 5.15.

Lemma 5.10 (AARB-Unicity). Non-faulty processes AARB-deliver at most one value.

Proof. By construction all non-faulty processes AARB-deliver at most one value. \square

Lemma 5.11 (AARB-Validity). If non-faulty process p_i AARB-delivers v , then v was AARB-broadcast by p_s .

Proof. Process p_i AARB-delivers v if it receives $h(d_r)$ messages $\langle \text{ECHO}, v, \cdot, \cdot \rangle$. Non-faulty processes only send an ECHO message for v if they receive $\langle \text{INIT}, v \rangle$. Thus, since $d + t < h(d_r)$, p_s AARB-broadcast v to at least one non-faulty process. \square

Lemma 5.12 (AARB-Send). If p_s is non-faulty and AARB-broadcasts v , then all non-faulty processes eventually AARB-deliver v .

Proof. Deceitful processes either broadcast v or multicast v' to a partition A and v to a partition B . In the first case (in which all deceitful behave like non-faulty processes), since the number of benign and Byzantine processes is $q + t \leq n - h(d_r)$ it follows that at least $h(d_r)$ non-faulty processes will echo v , being that enough for all processes to eventually AARB-deliver it.

Consider instead some $d_r \leq d + t$ deceitful processes behave deceitful echoing different messages to two different partitions each containing at least one non-faulty process. Then when the timer expires and non-faulty processes exchange their delivered ECHO messages, all processes will update their committee removing the d_r detected deceitful. Thus, since processes also recalculate the thresholds and recheck them after updating the committee, this case becomes the aforementioned case where no deceitful process behaves deceitful. The same occurs if one of the partitions AARB-delivers a value while the other does not and reaches the timer (Lemma 5.13). \square

Lemma 5.13 (AARB-Receive). If a non-faulty process AARB-delivers v from p_s , then all non-faulty processes eventually AARB-deliver v from p_s .

Proof. First, since $d + t < 2h(d_r) - n$ it follows that deceitful and Byzantine processes can not cause two non-faulty processes to AARB-deliver different values (analogously to Lemma 5.2). Then, before a process p_i AARB-delivers a value v , it broadcasts a READY message containing the certificate that justifies delivering v . Thus, when p_j receives that READY message, it also AARB-delivers v . \square

Lemma 5.14 (AARB-Accountability). If two non-faulty processes p_i and p_j AARB-deliver v and v' , respectively, such that $v \neq v'$, then all non-faulty processes eventually receive PoFs of the deceitful behavior of at least $2h(d_r) - n$ processes (including p_s).

Proof. Non-faulty processes broadcast the certificates of the values they AARB-deliver, containing $h(d_r)$ signed ECHO messages from distinct processes. Therefore, analogous to Lemma 5.2, at least $2h(d_r) - n$ processes must have sent conflicting ECHO messages, and they will be caught upon cross-checking the conflicting certificates. Also, some non-faulty processes must have received conflicting signed INIT messages from p_s in order to reach the threshold $h(d_r)$ to AARB-deliver conflicting messages, meaning that p_s is also faulty. \square

Lemma 5.15 (AARB-Active accountability). The Basilic's AARB protocol satisfies active accountability.

Proof. We prove here that if a number of faulty processes send conflicting messages to two subsets $A, B \subseteq N$, each containing at least one non-faulty process, then:

- eventually all non-faulty processes terminate without removing the faulty processes, or
- eventually all non-faulty processes receive a PoF for these faulty processes and remove them from the committee, after which, if the source is non-faulty, they terminate.

The proof is analogous if there are instead more than two conflicting messages and subsets containing at least one honest process. W.l.o.g. we consider just $p_A \in A$ and $p_B \in B$. If they both terminate despite the conflicting messages, we are finished. Suppose instead a situation in which only one of them, for example p_A , terminated AARB-delivering a value v . Then p_A broadcast a READY message with enough $h(d_r)$ ECHO messages in the certificate $cert$ for p_B to also AARB-deliver v and terminate. Let us consider w.l.o.g. only one faulty process p_i . If a signature from p_i in $cert$ conflicts with a local signature from p_i stored by p_B , then p_B constructs and broadcasts a PoF for p_i , and then updates the committee and the threshold. Then, it rechecks the certificate filtering out the signature by p_i , which would cause p_B to also AARB-deliver v (since the threshold also decreased accordingly).

Suppose neither p_A nor p_B has terminated yet. Then, when the timer is reached and they both broadcast the INIT and ECHO messages they delivered, they will both be able to construct a PoF for p_i , after which they update the committee and the threshold. Then, if the source was non-faulty, non-faulty processes can terminate analogously to the previous case. \square

5.3.4.3 Basilic binary consensus

We focus in this section on the properties of Basilic's binary consensus, AABC. We first prove that if all non-faulty processes start a round r with the same estimate v , then all non-faulty processes decide v in round r or $r + 1$ in Lemma 5.16. Then, we prove AABC-agreement in Lemma 5.17, AABC-strong validity in Lemma 5.18 and AABC-validity as Corollary 5.6 of Lemma 5.18, AABC-active accountability in Lemma 5.4, AABC-termination in Lemma 5.19, and AABC-accountability in Lemma 5.20. This thus makes AABC the first actively accountable binary consensus protocol, as we show in Theorem 5.3.

Lemma 5.16. Assume that each non-faulty process begins round r with the estimate v . Then, every non-faulty process decides v either at the end of round r or round $r + 1$.

Proof. By Lemma 5.7, v is eventually delivered to every non-faulty process. By Lemma 5.8, v is the only value delivered to each non-faulty process. As such, v is the only value in bin_vals and the only value echoed by non-faulty processes, since deceitful processes that prevent termination are removed from the committee when the timer expires (and the threshold is updated). This means that v will be the only value in vals . If $v = r \bmod 2$ then all non-faulty processes decide v . Otherwise, by the same argument every non-faulty process decides v in round $r + 1$. \square

Lemma 5.17 (AABC-Agreement). If $d+t \leq 2h-n$, no two non-faulty processes decide different values.

Proof. W.l.o.g. assume that the non-faulty process p_i decides v in round r . This means that p_i received $h(d_r)$ ECHO messages in round r , and that $\text{vals} = \{v\}$. Consider the ECHO messages received by non-faulty process p_j in the same round. If v is in p_j 's vals then p_j adopts estimate v because $v = r \bmod 2$. If instead p_j 's $\text{vals} = \{w\}$, $w \neq v$, then p_j received $h(d_r)$ ECHO messages containing only w .

Analogously to Lemma 5.2, it is impossible for p_j and for p_i to receive $h(d_r)$ ECHO messages for v and for w , respectively. We then conclude, by Lemma 5.16, that every non-faulty process decides value v in either round $r + 1$ or round $r + 2$. \square

Lemma 5.18 (AABC-Strong Validity). If a non-faulty process decides v , then some non-faulty process proposed v .

Proof. This proof is identical to Polygraph's proof of strong validity [47]. \square

Corollary 5.6 (AABC-Validity). If all processes are non-faulty and begin with the same value, then that is the only decision value.

Lemma 5.19 (AABC-Termination). Every non-faulty process eventually decides on a value.

Proof. This proof derives directly from Lemma 5.4. \square

Lemma 5.20 (AABC-Accountability). If two non-faulty processes output disagreeing decision values, then all non-faulty processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.

Proof. This proof is identical to Polygraph's proof of accountability [47], with the a generalization to any threshold $h(d_r)$ analogous to the one we make in Lemma 5.14. \square

Theorem 5.3. Basilic's AABC solves the actively accountable binary consensus problem.

Proof. Corollary 5.6 and Lemmas 5.17, 5.4, 5.19, and 5.20 prove AABC-validity, AABC-agreement, AABC-active accountability, AABC-termination and AABC-accountability, respectively. \square

5.3.4.4 General Basilic protocol

We gather all the results together in this section, showing the proofs for the general Basilic protocol. We prove active accountability in Lemma 5.21, validity in Lemma 5.22, termination in Corollary 5.7, agreement in Lemma 5.23, and accountability in Lemma 5.21. Finally, we prove that Basilic solves the actively accountable consensus problem in Theorem 5.4.

Lemma 5.21 (Active accountability). Basilic satisfies active accountability.

Proof. We show w.l.o.g. that if a faulty process p_i sends two conflicting messages to two subsets $A, B \subseteq N$, each containing at least one honest process, then eventually all honest processes terminate, or instead they receive a PoF for p_i and remove it from the committee, after which they all terminate. The proof is analogous if there are instead more than two conflicting messages and subsets containing at least one honest process.

First, analogously to Lemma 5.4, all conflicting messages that can be sent in Basilic are messages of Basilic's AARB or AABC, that already satisfy active accountability (see Lemmas 5.4 and 5.15). This means that if $d_r > 0$, then honest processes eventually update the committee and threshold, after which they recheck if they hold enough signed messages to terminate. Next, we prove termination. By the AARB-Send property (Lemma 5.12), all honest processes will eventually deliver the proposals from honest processes. Eventually all honest processes propose 1 in all binary consensus whose index corresponds to an honest proposer, and by AABC-Validity decide 1. Since eventually $h(d_r) \leq n - q - d - t$ if enough d_r prevent termination and are thus detected and removed, we can conclude that at least $h(d_r)$ binary consensus instances will terminate deciding 1.

Once honest processes decide 1 on at least $h(d_r)$ proposals, they propose 0 to the rest, and by AABC-Termination (Lemma 5.19) all remaining binary consensus instances will terminate. Next, we show that for every binary consensus upon which we decided 1, at least one honest process AARB-delivered its associated proposal. For the sake of contradiction, if no honest process had AARB-delivered its associated proposal, then all honest processes would have proposed 0, meaning by AABC-Validity that the final decision of the binary consensus would have been 0, not 1. As a result, by the AARB-Receive property (Lemma 5.13), eventually all honest processes will deliver the proposal for all binary consensus that they decided 1 upon. Finally, processes decide the value proposed by the proposer with the lower index. \square

Corollary 5.7 (Termination). The Basilic protocol satisfies termination.

Proof. Trivial from Lemma 5.21. \square

Lemma 5.22 (Validity). Basilic satisfies validity.

Proof. This is trivial by Corollary 5.6 and the proofs of AARB. Suppose all processes begin Basilic with value v . If all processes are non-faulty then every proposal AARB-delivered was AARB-sent by a non-faulty process, and since all processes AARB-send v , only v is AARB-delivered.

Since initially processes only start an AABC instance for which they can propose 1, this means that eventually all processes start one AABC instance proposing 1. By Corollary 5.6, this instance will terminate with all processes deciding 1. Since the rest of the AABC instances will eventually terminate by Lemma 5.19, this means that processes will terminate at least one instance of AABC outputting 1. Upon calculating the minimum of all values (which are all v) whose associated bit is set to 1, all processes will decide v . \square

Lemma 5.23 (Agreement). The Basilic protocol satisfies agreement.

Proof. The proof is immediate from Lemmas 5.17 and 5.13. \square

Lemma 5.24 (Accountability). If two non-faulty processes output disagreeing decision values, then all non-faulty processes eventually identify at least $2h - n$ faulty processes responsible for that disagreement.

Proof. The proof is immediate from Lemmas 5.4 and 5.15. \square

Theorem 5.4 (Theorem 5.2). The Basilic protocol with initial threshold $h_0 \in (n/2, n]$ solves the actively accountable consensus problem if $d + t < 2h_0 - n$ and $q + t \leq n - h_0$.

Proof. Corollary 5.7 and Lemmas 5.21, 5.22, 5.23, and 5.24 satisfy termination, active accountability, validity, agreement, and accountability, respectively. \square

Corollary 5.8 (Corollary 5.5). The Basilic class of protocols solves the actively accountable consensus problem if $n > 3t + d + 2q$.

Proof. The proof is immediate from Theorem 5.2 after removing h_0 from the system of two inequations defined by $d + t < 2h_0 - n$ and $q + t \leq n - h_0$. \square

5.3.5 Basilic's complexities

In this section, we show the complexities of Basilic. We execute one instance of Basilic's AARB reliable broadcast and of Basilic's AABC binary consensus per process.

5.3.5.1 Naive Basilic

We summarize the complexities of the three protocols without optimizations in Table 5.1, that we prove in Lemmas 5.25 and 5.26, and Theorem 5.5.

Complexity	AARB	AABC	Basilic
Time	$\mathcal{O}(1)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Message	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
Bit	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(\lambda n^5)$

Table 5.1: Complexities of naive implementations of Basilic protocols.

Lemma 5.25 (Basilic’s AARB Complexity). After GST and if the source is non-faulty, Basilic’s AARB protocol has time complexity $\mathcal{O}(1)$, message complexity $\mathcal{O}(n^2)$ and bit complexity $\mathcal{O}(\lambda \cdot n^3)$.

Proof. After GST, all non-faulty processes will have received a message from each non-faulty process and from each deceitful processes by the time the timer reaches 0. Thus, either non-faulty processes can terminate, or they broadcast their current list of ECHO and INIT messages, after which they remove the detected deceitful processes, and they can terminate too. Thus, the time complexity is $\mathcal{O}(1)$. Then, the message complexity is $\mathcal{O}(n^2)$, as each non-faulty process broadcasts at least one ECHO and READY message, and, in some executions, a list of ECHO messages that they delivered by the time the timer reaches 0. Since both this list and READY messages contain $\mathcal{O}(n)$ signatures, or $\mathcal{O}(\lambda n)$ bits, the bit complexity of Basilic’s AARB is $\mathcal{O}(\lambda n^3)$. \square

Lemma 5.26 (Basilic’s AABC Complexity). After GST, Basilic’s AABC protocol has time complexity $\mathcal{O}(n)$, message complexity $\mathcal{O}(n^3)$ and bit complexity $\mathcal{O}(\lambda \cdot n^4)$.

Proof. After GST, Basilic’s AABC protocol terminates in the first round (i) whose leader is a non-faulty process and (ii) after having removed enough deceitful faults so that they cannot prevent termination. Since $t + d + q < n$, we have that (i) holds in $\mathcal{O}(n)$. As for every added round in which deceitful faults prevent termination, a non-zero number of deceitful faults are removed, we have that (ii) holds in $\mathcal{O}(n)$ as well. This means that Basilic terminates in $\mathcal{O}(n)$ rounds after GST. In each round during phase 1 of AABC, non-faulty processes execute an ABV-broadcast of $\mathcal{O}(n^2)$, obtaining $\mathcal{O}(n^3)$ messages. The bit complexity is $\mathcal{O}(\lambda n^4)$ as each message may contain up to two ledgers of $\mathcal{O}(n)$ signatures, or $\mathcal{O}(\lambda n)$ bits. The complexities of phase 2 are equivalent and obtained analogously to those of phase 1, as non-faulty processes may broadcast $\mathcal{O}(n)$ signatures if deceitful faults prevent termination of phase 2, or a certificate if they decide in this round. \square

Theorem 5.5. The Basilic protocol has time complexity $\mathcal{O}(n)$, message complexity $\mathcal{O}(n^3)$ and bit complexity $\mathcal{O}(\lambda \cdot n^5)$.

Proof. The proof is immediate from Lemma 5.26 and Lemma 5.25 since Basilic executes n instances of AARB followed by n instances of AABC. \square

5.3.5.2 Optimized Basilic

The complexities of Basilic after GST share the same asymptotic complexity of other recent works that are not actively accountable [47, 132], some of them not being accountable either [166], as we show in Table 5.2. This is because the adversary cannot prevent termination of any phase. Thus, after GST, all processes can continue to the next phase or terminate the protocol by the time the timer for that phase expires, resulting in an execution equivalent to that of Polygraph (apart from one additional message broadcast in ABV-broadcast). In this table, naive Basilic represents the protocol we show in Algorithm 7, whereas the following row, multi-valued Basilic, shows the analogous optimizations shown in Polygraph and applicable to

the Basilic protocol as well [47]. The rows containing 'superblock' refer to the result of applying the additional superblock optimization [23, 47, 27, 80, 81, 28], which consists on deciding on the union of all $h(d_r)$ ($\mathcal{O}(n)$) proposals whose associated AABC instance output 1, solving SBC (Definition 2.2.4) instead of just consensus. This optimization is only available to democratic protocols in which all processes provide an input [23, 47, 27, 80, 81, 28] (i.e. DBFT, Polygraph and Basilic in Table 5.2), as noted in Section 2.2.9. After these optimizations, the resulting normalized bit complexity (i.e. per decision) of Basilic is as low as those of other works that are only accountable and not actively accountable, such as BFT Forensics [132] or Polygraph [47]. Furthermore, since this is the minimum complexity for accountability [47], this means that this is also optimal in the bit complexity. Note that other optimizations present in other works, such as the possibility to obtain an amortized complexity of $\mathcal{O}(\lambda n^2)$ in BFT Forensics per decision after n iterations of the protocol [186], are orthogonal to our optimizations, and thus they also apply to Basilic.

An additional advantage of Basilic, as well as of other democratic protocols [47, 23], compared to non-democratic protocols [132, 186], is that the distribution of proposals scatters the bits throughout multiple channels of the network, instead of bloating channels that have the leader as sender or recipient. That is, while BFT Forensics' normalized an amortized per route complexity (Section 2.2.9) is $\Omega(\lambda n)$, as this is the number of bits that must be sent through the $\Theta(n)$ channels to and from the leader, Basilic's is instead $\mathcal{O}(\lambda)$, which are instead sent through each of the $\Theta(n^2)$ pairwise channels of the network.

Finally, not only are the rest of the protocols in Table 5.2 not actively accountable, but also this means that they only solve consensus tolerating at most $t < n/3$ faults in the BDB model, whereas Basilic with initial threshold $h_0 = 2n/3$ solves consensus where $d + t < n/3$ and $q + t \leq n/3$ faults, hence tolerating the strongest adversary among these works.

5.3.6 Solving eventual consensus with Basilic

In this section, we adapt Basilic to solve eventual consensus in the BDB model, and then prove that the Basilic protocol is resilient optimal. We detail thus \diamond -Basilic (*BEC*), an adaptation of Basilic for the \diamond -consensus problem (see Section 2.2.6.4). Process p_i executes \diamond -Basilic with the following steps:

1. BEC first executes **Basilic-gen-propose** $_{h_0}(v_i)$, whose output is returned by p_i as BEC's output of **proposeEC** $_0$.
2. If p_i finds no disagreement between operations k and k' , then for all operations **proposeEC** $_j$, $k' > j \geq k$, the output is that of **proposeEC** $_{j-1}$.
3. If p_i finds a new disagreement at operation j for some index $r \in [0, n - 1]$, then:
 - (a) If the disagreement is between AARB-delivered values, BEC resolves it as follows: let $(\text{EST}, \langle u, r \rangle)$ be the value that differs with the locally AARB-delivered value $(\text{EST}, \langle v, r \rangle)$, then, for **proposeEC** $_j$, p_i applies $y = \min(v, u)$ to the disagreeing value. Next, if the output of **proposeEC** $_{j-1}$ was v , p_i replaces the AARB-delivered value with y , and outputs y instead for **proposeEC** $_j$.

Table 5.2: Complexities of Basilic compared to other works.

Algorithm	Msgs	Bits	Accountable	Actively accountable
PBFT [166]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	\times	\times
Tendermint [187]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	\times	\times
HotStuff [186]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^2)$	\times	\times
DBFT superblock [23]	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	\times	\times
BFT Forensics [132]	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^3)$	\checkmark	\times
Polygraph's binary [47]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	\checkmark	\times
Naive Polygraph [47]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	\checkmark	\times
Polygraph Multi-v. [47]	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	\checkmark	\times
Polygraph superblock [47]	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	\checkmark	\times
Basilic's AABC	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$	\checkmark	\checkmark
Naive Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^5)$	\checkmark	\checkmark
Multi-valued Basilic	$\mathcal{O}(n^4)$	$\mathcal{O}(\lambda n^4)$	\checkmark	\checkmark
Basilic superblock	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^3)$	\checkmark	\checkmark

(b) If the disagreement is between values 1 and 0 decided at AABC's protocol, then p_i sets $\text{bin-decisions}[r]$ to 1. Then, p_i recalculates if the minimum decided value changed after adding this binary decision (i.e., p_i re-executes lines 11-13 of Algorithm 7), and outputs this decision for proposeEC_j .

(c) Finally, p_i broadcasts the values (and certificates) of all the disagreements that p_i has not yet broadcast.

We show in Theorem 5.6 that \diamond -Basilic with initial threshold h_0 solves the \diamond -consensus problem if $d + t < h_0$ and $q + t \leq n - h_0$, where t , d and q are the numbers of Byzantine, deceitful and benign processes, respectively, and h_0 the initial threshold. This means that the \diamond -Basilic class of protocols solves \diamond -consensus for any combination of t , d and q Byzantine, deceitful and benign processes, respectively, such that $2t + d + q < n$, as we show in Corollary 5.9.

Theorem 5.6 (\diamond -Consensus per threshold). The \diamond -Basilic protocol with initial threshold h_0 solves the \diamond -consensus problem if $d + t < h_0$ and $q + t \leq n - h_0$.

Proof. \diamond -Integrity is trivial. The bound $q + t \leq n - h_0$ is proven in Corollary 5.3: \diamond -Basilic starts by executing Basilic, which does not terminate unless $q + t \leq n - h_0$, satisfying \diamond -Termination. \diamond -Validity derives immediately from Basilic's proof of validity (Lemma 5.22).

We only have left to prove \diamond -Agreement. If $d + t < h_0$ then all valid certificates contain at least one honest process. This means that the number of disagreements is finite. Then, since honest processes broadcast all disagreements they find (and their corresponding valid certificates), all honest processes will eventually find all disagreements. Also, all honest processes will find all disagreements of Basilic by its accountability property (Lemma 5.21). Let us consider that all honest processes, except p_i , have already found and treated all disagreements

(as specified by the \diamond -Basilic protocol). Suppose that p_i finds the last disagreement at the start of operation proposeEC_{k-1} for some $k > 0$. Then, for all $j \geq k$, no two honest processes return different values to proposeEC_k , satisfying \diamond -Agreement. \square

Corollary 5.9 (\diamond -Consensus). The Basilic class of protocols solves \diamond -consensus if $n > 2t + d + q$.

Proof. The proof is immediate from Theorem 5.6 after removing h_0 from the system of two inequations defined by $d + t < h_0$ and $q + t \leq n - h_0$. \square

5.4 The Zero-Loss Blockchain

Having shown our actively accountable Basilic class of protocols, we now present our Zero-loss Blockchain (ZLB). ZLB is the first blockchain that tolerates an adversary controlling up to t_s processes trying to cause a disagreement, while also tolerating instead up to t_l Byzantine processes. ZLB achieves this high level of tolerance by resolving temporary disagreements and replacing provably fraudulent processes. For this purpose, we first detail the Longlasting Blockchain problem and an additional assumption of the adversary fitting for the long-lasting nature of ZLB.

In particular, solving the longlasting blockchain problem is to solve consensus when possible ($n > 3t + d + 2q$, Corollary 5.5), and to recover from a situation where consensus is violated ($n \leq 3t + d + 2q$) by excluding faulty processes, resolving this violation, and preventing future ones ($n > 3t' + d' + 2q$).

5.4.1 Longlasting Blockchain

A Longlasting Blockchain (LLB) is a Byzantine fault-tolerant blockchain that allows for some consensus instances to reach a disagreement before fixing the disagreement by merging the branches of the resulting fork and deciding the union of all the past decisions using SBC (Definition 2.2.4). As a result, we consider that a consensus instance Φ_i outputs a set of enumerable decisions $\text{out}(\Phi_i) = s_i$, $|s_i| \in \mathbb{N}$ that all n processes replicate. We refer to the state of the blockchain at the i -th consensus instance Φ_i as all decisions of all instances up to the i -th consensus instance.

More formally, a blockchain is an LLB if it ensures termination, agreement and convergence:

Definition 5.4.1 (Longlasting Blockchain Problem). A blockchain is an LLB if all the following properties are satisfied:

1. **Termination:** For all $k > 0$, consensus instance Φ_k solves eventual consensus.
2. **Agreement:** If $d + t < 2h - n$ when Φ_k starts, then honest processes executing Φ_k reach agreement.
3. **Convergence:** There is a finite number of consensus instances that solve eventual consensus, after which all consensus instances solve consensus.

Termination does not imply agreement from among honest processes in the first output of the same consensus instance, but it implies that all instances terminate with an output that may change to eventually reach agreement, whereas agreement is the classic property of consensus. Convergence guarantees that there is a limited number of disagreements before reaching agreement.

5.4.2 Slowly-adaptive adversary

Considering a blockchain SMR rather than single-shot consensus requires to cope with attacks in which the corrupted processes that the adversary chooses change over time. As a result, we consider that the adversary that controls these faulty processes is adaptive in that f can change over time. However, we assume that the adversary is *slowly-adaptive* [100], as in previous blockchain systems with dynamic membership [100], in that the adversary experiences *static periods* during which Byzantine, deceitful, benign and honest processes remain so. We assume that these static periods are long enough for honest processes to discover and replace the faulty processes, for the sake of convergence. In particular, each static period t is assigned a consensus instance Φ_k and t starts when Φ_k starts. To cope with pipelined consensus instances, t may not end exactly when Φ_k ends, as there can be Φ_ℓ, \dots, Φ_m instances running at this time, in which case t ends (and static period $t + 1$ starts) as soon as Φ_ℓ, \dots, Φ_m and Φ_k have all ended with agreement.

5.4.3 Pool of process candidates

As our system will perform a membership change that excludes some processes and includes new ones, we model all users that can join as processes in the system by assuming that there exists a large pool of m users among which at least $2n/3$ are honest users (m can be much greater than n) and the rest are deceitful. This pool simply intends to represent the weakest requirement that from the entire world of users that will ever be proposed to be included as processes, at least $2n/3$ honest ones will eventually be proposed by honest processes. Notice this is a significantly weaker assumption than assuming, for example, that honest processes always propose honest users to be included. For simplicity and w.l.o.g., we assume that no user from this pool is proposed twice if it has been a process before, within the same static period of the adversary.

5.4.4 The Zero-Loss Blockchain

We detail our system in this section. Its two main ideas are (i) to replace deceitful processes undeniably responsible for a fork by new processes to converge towards a state where consensus can be reached, and (ii) to refund conflicting transactions that were wrongly decided. We will show that ZLB solves the Longlasting Blockchain problem. As depicted in Figure 5.4, we present below the components of our ZLB system, namely the Accountable SMR (ASMR) (Section 5.4.4.1) and the Blockchain Manager (BM) (Section 5.4.4.2) but we defer the zero loss payment application (Section 5.5).

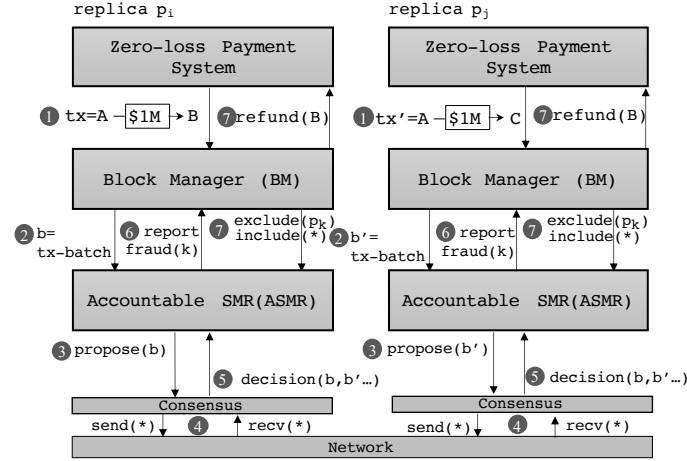


Figure 5.4: The distributed architecture of our ZLB system relies on Accountable SMR (ASMR), BM and the payment system. ② Each process batches some payment requests illustrated with ① a transfer tx (resp. tx') of \$1M from Alice's account (A) to Bob's (B) (resp. Carol's (C)). Consider that Alice has \$1M initially and attempts to double spend by modifying the code of a process p_k under her control so as to execute a coalition attack. ③–⑤ The ASMR component detects the deceitful process p_k that tried to double spend, the associated transactions tx and tx' and account A with insufficient funds. It uses A's balance to fund transaction tx , ⑥ notifies BM that ⑦ excludes or replaces process p_k and ⑦ funds tx' with p_k 's slashed deposit.

As long as new requests are submitted by a user to a process, the payment system component of the process converts them into payments that are passed to the BM component. As depicted in Figure 5.4, when sufficiently many payment requests have been received, the BM issues a batch of requests to the Accountable SMR (ASMR) that, in turn, proposes it to the consensus component. The consensus component exchanges messages through the network for honest processes to agree. If a disagreement is detected, then the account of the deceitful process is slashed. Consider that Alice (A) attempts to double spend by (i) spending her \$1M with both Bob (B) and Carol (C) in tx and tx' , respectively, and (ii) hacking the code of process p_k that commits deceitful faults to produce a disagreement. Once the ASMR detects the disagreement, BM is notified, process p_k is replaced and tx' is funded with p_k 's slashed deposit.

5.4.4.1 Accountable SMR (ASMR)

In order to detect faulty processes, we now present an accountable state machine replication, ASMR. ASMR is divided into five phases. For each index, ASMR first executes the accountable consensus in phase ① to try to agree on a set of transactions, to the run up to four additional subsequent phases ②–⑤ to tolerate disagreements: ② a confirmation that aims at confirming that no disagreement took place, ③–④ a membership change that aims at replacing deceitful processes responsible for a disagreement by new processes, and ⑤ a reconciliation phase that combines all the decisions of the disagreement, as depicted in Figure 5.5.

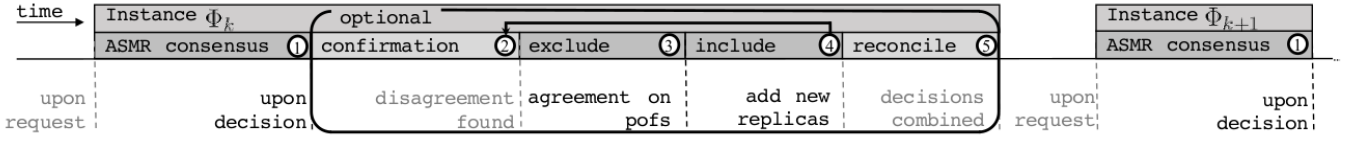


Figure 5.5: If there are enqueued requests that wait to be served, then a process starts a new instance Φ_k by participating in an ASMR consensus phase ①; a series of phases may follow: ② the process tries to confirm this decision to make sure no other honest process disagrees, ③ it invokes an exclusion protocol if faulty processes caused a disagreement, ④ it then includes new processes to compensate for the exclusion, and ⑤ merges the two batches of decided transactions. Some of these phases complete upon consensus termination (in black) whereas other phases terminate upon simple notification reception (in gray). The process starts a new instance Φ_{k+1} without waiting for phases ②-⑤ to terminate, as this is not always guaranteed.

① **ASMR consensus:** Honest processes propose a set of transactions, which they received from users, to an instance of the Basilic class of actively accountable consensus protocols, with initial threshold $h_0 = 2n/3$, in the hope to reach agreement. When the consensus terminates, all honest processes agree on the same decision or some honest processes disagree: they decide distinct sets of transactions.

Our ASMR's consensus is Basilic, which already removes processes at runtime. However, it is important to note that honest processes do not permanently remove processes removed within a consensus instance of Basilic (in that once the consensus instance terminates, these processes are re-added at the start of the next consensus instance). This ensures that a slowly-adaptive adversary cannot increase the percentage of the committee it controls as the committee decreases in size. Honest processes store however the PoFs, and will eventually trigger a membership change that will permanently replace faulty processes by new processes.

② **Confirmation:** As honest processes could be temporarily unaware of a disagreement if the adversary controls $d + t \geq n/3$ deceitful and Byzantine processes, they enter a confirmation phase waiting for messages coming from more distinct processes than what BFT consensus requires. If faulty processes caused a disagreement, then the confirmation terminates and leads honest processes to detect disagreements, i.e., honest processes receive certificates supporting distinct decisions. Otherwise, this phase may not terminate, as an honest process needs to deliver messages from more than $(\delta + 1/3) \cdot n$ processes, where δ is the ratio of potential deceitful faults $\delta = (d + t)/n$, so as to guarantee that no disagreement was possible by a *deceitful ratio* δ . In particular, with $q = 0$ (i.e. $f = t + d$), honest processes need to receive agreeing messages from $n - x$ processes solving $\lfloor (n - x)/(f - x + 1) \rfloor = 1$, which translates to at least $8n/9$ processes for $f = \lceil 5n/9 - 1 \rceil$, or all processes for $f = \lceil 2n/3 \rceil$, as we show in Theorem 5.7. More specifically, we speak of a decision v being α -confirmed, $\alpha \in [0, 2/3]$, at Φ_k if only an adversary with a deceitful ratio $\delta \geq \alpha$ could have caused an honest process to decide $v' \neq v$ at Φ_k .

However, Φ_k always terminates, as it proceeds in parallel with the confirmation without waiting for its termination. If the confirmation phase terminates, it either confirms that a block

is irrevocably final (no process disagreed), or a membership change starts. Similarly, if there is a disagreement, then the confirmation phase always terminates.

③-④ **Membership change:** Our membership change (Algorithm 11) consists of two consecutive consensus algorithms: one that excludes deceitful processes (line 22), and another that adds newly joined processes (line 40). We separate inclusion and exclusion in two consensus instances to avoid deciding to exclude and include processes proposed by the same process. Process p_i maintains a series of variables: the current consensus instance Φ_k , the *deceitful* processes among the whole set N of current process ids, a set N' of process ids that is updated at runtime for the exclusion protocol, the pool of process candidates *pool*, a set of certificates *certificates*, a set of PoFs *pofs* and of new PoFs *new_pofs*, a local threshold f_d of detected deceitful processes, a set *cons-exclude* of decided PoFs and a set *cons-include* of decided new processes.

③ **Exclusion protocol:** If honest processes detect at least $f_d = 2h_0 - n$ (i.e. $f_d = n/3$ for $h_0 = 2n/3$) deceitful processes (via distinct PoFs), they stop their pending ASMR consensus (line 19) before restarting it with the new set of processes (line 47). Then, honest processes start the membership change ignoring messages from these f_d processes by using instead an updated committee N' that excludes these processes (lines 20-22). We fix $f_d = 2h_0 - n$ for the remaining of this dissertation. Honest processes propose in line 22 their set of PoFs at the start of the exclusion protocol **ex-propose** by invoking the Basilic actively accountable consensus protocol. We will use $h'(d'_r) = h'_0 - d'_r$ to refer to the voting threshold of the exclusion protocol, and N' to refer to the updated committee of the exclusion protocol. We will discuss specific values of h'_0 later in this dissertation.

The key novelty of our exclusion protocol is for processes to exclude other processes, and thus update their committee N' , at runtime upon reception of new valid PoFs (lines 23-24). Note that Basilic already removes detected faulty processes at runtime, but starting with $d'_r = f_d$ removed processes gives an advantage to honest processes from the start. Hence, upon delivering a certificate (line 29), honest processes verify that the certificate contains a threshold $h'(d'_r)$ of signatures from processes that have not been detected faulty (line 33) and decide the proposals that the certificate justifies at line 34. Upon updating their committee, honest processes re-check all their certificates (line 26) and re-broadcast their PoFs (line 25). As our exclusion protocol solves the SBC problem (Definition 2.2.4), it maximizes the number of excluded processes by deciding at least $h'(d'_r)$ proposals at once.

Note that instead of waiting for f_d PoFs (line 17), processes could start Algorithm 11 as soon as they detect one deceitful process, however, waiting for at least f_d PoFs guarantees that a membership change is necessary, as there are enough attackers to cause a disagreement, and will help remove many deceitful processes from the same coalition at once.

④ **Inclusion protocol:** To compensate for the excluded processes, an inclusion protocol **inc-propose** (line 40) adds new candidate processes taken from the pool of process candidates (Section 5.4.2) in line 39. This inclusion protocol is also an instance of Basilic with the same voting threshold $h'(d'_r) = h'_0 - d'_r$ as the exclusion protocol, but it differs in the format and verification of the proposals: each proposal contains as many new processes as the number of

Algorithm 11 Membership change at process p_i , consensus Φ_k

```

1: State:
2:    $\Phi_k, k^{th}$  instance of ASMR consensus
3:    $N$ , set of processes forming the committee
4:    $N'$ , updated set of processes, initially  $N' = N$ 
5:   certificates, received certificates during exclusion, initially  $\emptyset$ 
6:   pofs, the set of proofs of fraud (PoFs), initially  $\emptyset$ 
7:   new_pofs, set of newly delivered PoFs, initially  $\emptyset$ 
8:   cons-exclude, the set of PoFs output by consensus, initially  $\emptyset$ 
9:   cons-include, the set of new processes output by consensus, initially  $\emptyset$ 
10:  pool, the pool of process candidates from which to propose new processes
11:  deceitful  $\in I$ , the identity of an agreed deceitful process, initially  $\emptyset$ 
12:   $f_d$ , the threshold of proofs of fraud to recover,  $\lceil n/3 \rceil$  by default



---


13: Upon receiving a list of proofs of fraud _pofs:
14:   if (verify(_pofs)) then ▷ if PoFs are correctly signed
15:     new_pofs  $\leftarrow$  _pofs \ pofs
16:     pofs.add(new_pofs) ▷ add PoFs on distinct processes
17:     if (ex-propose not started) then
18:       if (size(pofs)  $\geq f_d$ ) then ▷ enough to change members
19:         if ( $\Phi_k$  started and not finished) then  $\Phi_k.stop()$ 
20:          $N' \leftarrow N' \setminus pofs.processes()$ 
21:         ex-propose.update_committee(pofs) ▷ update committee
22:         ex-propose.start(pofs) ▷ exclusion consensus
23:       else if (new_pofs  $\neq \emptyset$  and ex-propose not finished) then  $N' \leftarrow N' \setminus new\_pofs.processes()$ 
24:         ex-propose.update_committee(new_pofs) ▷ update committee
25:         broadcast(new_pofs) ▷ broadcast new PoFs
26:         ex-propose.check_certificates(certificates) ▷ recheck certificates



---


27: Upon receiving a certificate ex-cert of the exclusion protocol:
28:   if (ex-cert  $\notin$  certificates and verify_certificate(ex-cert)) then certificates.add(ex-cert)
29:   ex-propose.check_certificates( $\{ex-cert\}$ ) ▷ check certificate with current  $N'$ 



---


30: function ex-propose.check_certificates(certs):
31:   for all cert  $\in$  certs do
32:     if (verify_certificate(cert)) then
33:       if ( $|cert.processes() \cap N'| \geq \frac{2|N'|}{3}$ ) then ▷ current threshold
34:         ex-propose.cert_decide(cert) ▷ decide certificate's decision



---


35: Upon deciding a list of proofs of fraud cons-exclude in ex-propose:
36:   detected-fraud(cons-exclude.get_processes()) ▷ application punishment
37:   pofs  $\leftarrow$  pofs \ cons-exclude.get_pofs() ▷ discard the treated pofs
38:    $N \leftarrow N \setminus cons\_exclude.get\_deceitfuls()$  ▷ exclude deceitful
39:   inc-prop  $\leftarrow pool.take(|cons-exclude|)$  ▷ take processes from the pool
40:   inc-propose.start(inc-prop) ▷ inclusion cons.



---


41: Upon deciding a list of processes to include cons-include in inc-propose:
42:   new_processes  $\leftarrow choose(|cons-exclude|, cons-include)$  ▷ deterministic
43:   for all new_process  $\in$  new_processes do ▷ for all new to inc.
44:     set-up-connection(new_process) ▷ new process joins
45:     send-catchup(new_process) ▷ get latest state
46:    $N \leftarrow N \cup new\_processes$ 
47:   if ( $\Phi_k$  stopped) then goto ① of Figure 5.5 ▷ restart cons.

```

processes excluded (lines 39-40). By contrast with the exclusion protocol, the inclusion protocol uses the updated committee (N from line 38 onward), resulting from taking the committee from the start of the membership change and excluding from it the decided processes to exclude at the end of the exclusion consensus (line 38). Since the union of the $h'(d'_r)$ decided proposals contains more than enough processes to include, honest processes apply a deterministic function **choose** (line 42) to the union of all decided proposals. This function restores the original committee size to n by selecting the processes evenly from all decided proposals. This guarantees (i) a fair distribution of inclusions across all decisions, and (ii) that the deceitful ratio does not increase even if all included processes are deceitful. At the end, the excluded processes are punished by the application layer (line 36) and the new processes are included (lines 42-47).

Honest processes from different partitions might find themselves at different consensus instances at the moment they execute the membership change. For this reason, even after the membership change terminates, there is a transient period where honest processes may receive blocks with certificates containing excluded processes, that were decided and broadcast by other honest processes in a different partition before they executed the membership change. Note, however, that all certificates contain at least 1 honest process by construction as long as $f < h$, and thus all honest processes eventually update their committee and stop generating new certificates with excluded processes.

⑤ **Reconciliation:** Upon delivering a conflicting block with an associated valid certificate, the reconciliation starts by combining all transactions that were decided by distinct honest processes in the disagreement. These transactions are ordered through a deterministic function, whose simple example is a lexicographical order but can be made fair by rotating over the indices of the instances.

Once the current instance Φ_k terminates, another instance Φ_{k+1} can start, even if it runs concurrently with a confirmation or a reconciliation at index k or at a lower index.

5.4.4.2 Blockchain Manager (BM)

We now present the Blockchain Manager (BM) that builds upon ASMR to merge the blocks from multiple branches of a blockchain when forks are detected. Once a fork is identified, the conflicting blocks are not discarded as it would be the case in classic blockchains when a double-spending occurs, but they are merged. Upon merging blocks, BM also copes with conflicting transactions, as the ones of a payment system, by taking the funds of excluded processes to fund conflicting transactions. We defer to Section 5.5 the details of the amount processes must have on a deposit to guarantee this funding.

Similarly to Bitcoin [1], BM accepts transaction requests from a permissionless set of users. In particular, this allows users to use different devices or wallets to issue distinct transactions withdrawing from the same account—a feature that is not offered in payment systems without consensus [188]. In contrast with Bitcoin, but similarly to recent blockchains [26, 23], our system does not incentivize all users to take part in trying to decide upon every block, instead a restricted set of permissioned processes have this responsibility for a given block. This is why

ZLB offers what is often called an open permissioned blockchain [23]. Nevertheless, ASMR can offer a permissionless blockchain with committee sortition [26] without substantial modifications. We discuss a committee sortition protocol for ZLB in Chapter 6.

5.4.4.2.1 Guaranteeing consistency across processes

By building upon the underlying ASMR that resolves disagreements, BM features a block merge to resolve forks, along with excluding detected faulty processes and including new processes. A consensus instance may reach a disagreement, resulting in the creation of multiple branches or blockchain forks (Theorem 5.8). BM builds upon the membership change of ASMR in order to recover from forks. In particular, the fact that ASMR excludes f_d deceitful processes each time a disagreement occurs guarantees that the ratio of deceitful processes δ converges to a state where consensus is guaranteed (Theorem 5.12). The maximum number of branches that can result from forks depends on the number q of benign faults, the number d of deceitful faults and the number t of Byzantine faults, as well as on the voting threshold h , as was already shown for histories of SMRs [46], and as we restate in Theorem 5.8.

5.4.4.2.2 In memory transactions

ZLB is a blockchain that inherits the same *Unspent Transaction Output (UTXO)* model of Bitcoin [1]; the balance of each account in the system is stored in the form of a UTXO table. In contrast with Bitcoin, the number of maintained UTXOs is kept to a minimum in order to allow in-memory optimizations. Each entry in this table is a UTXO that indicates some amount of coins that a particular account has. When a transaction transferring from source accounts s_1, \dots, s_x to recipient accounts r_1, \dots, r_y executes, it checks the UTXOs of accounts s_1, \dots, s_x . If the UTXO amounts for these accounts are sufficient, then this execution consumes as many UTXOs as possible and produces another series of UTXOs now outputting the transferred amounts to r_1, \dots, r_y as well as what is potentially left to the source accounts s_1, \dots, s_x . Maximizing the number of UTXOs to consume helps keeping the table compact. Each process can typically access the UTXO table directly in memory for faster execution of transactions.

5.4.4.2.3 Protocol to merge blocks

As depicted in Algorithm 12, the state of the blockchain Ω consists of a set of inputs *inputs-deposit* (line 4), a set of account addresses *punished-acts* (line 5) that have been used by deceitful processes, a *deposit* (line 3), that is used by the protocol, a set *txs* of transactions and a list *utxos* of UTXOs. The algorithm propagates blocks by broadcasting them on the network. As such, the algorithm starts upon reception of a valid block that conflicts with a known block of the blockchain Ω by trying to merge all transactions of the received block with those of the blockchain Ω (line 11). This is done by invoking the function **CommitTxMerge** (lines 17–23) where the inputs get appended to the UTXO table and conflicting inputs are funded with the deposit (line 22) of excluded processes. We explain in Section 5.5 how to build a payment system with a sufficient deposit to remedy successful disagreements.

Algorithm 12 Block merge at process p_i

```

1: State:
2:    $\Omega$ , a blockchain record with fields:
3:     deposit, an integer, initially 0
4:     inputs-deposit, a set of deposit inputs, initially in the first deposit
5:     punished-acts, a set of punished account addresses, initially  $\emptyset$ 
6:     txs, a set of UTXO transaction records, initially in the genesis block
7:     utxos, a list of unspent outputs, initially in the genesis block

8: Upon receiving conflicting block block:                                ▷ merge block
9:   for tx in block do                                                    ▷ go through all txs
10:    if (tx not in  $\Omega.txs$ ) then                                          ▷ check inclusion
11:      CommitTxMerge(tx)                                                  ▷ merge tx, go to line 17
12:      for out in tx.outputs do                                          ▷ go through all outputs
13:        if (out.account in  $\Omega.punished-acts$ ) then                    ▷ if punished
14:          PunishAccount(out.account)                                     ▷ punish also this new output
15:      RefundInputs()                                                       ▷ refill deposit, go to line 24
16:      StoreBlock(block)                                                  ▷ write block in blockchain

17: CommitTxMerge(tx):
18:   toFund  $\leftarrow 0$ 
19:   for input in tx.inputs do                                              ▷ go through all inputs
20:     if (input not in  $\Omega.utxos$ ) then                                     ▷ not spendable, need to use deposit
21:        $\Omega.inputs-deposit.add(input)$                                      ▷ use deposit to refund
22:        $\Omega.deposit \leftarrow \Omega.deposit - input.value$                ▷ deposit decreases in value
23:     else  $\Omega.consumeUTXO(input)$                                        ▷ spendable, normal case

24: RefundInputs():
25:   for input in  $\Omega.inputs-deposit$  do                                   ▷ go through inputs that used deposit
26:     if (input in  $\Omega.utxos$ ) then                                       ▷ if they are now spendable
27:        $\Omega.consumeUTXO(input)$                                            ▷ consume them
28:        $\Omega.deposit \leftarrow \Omega.deposit + input.value$                ▷ and refill deposit

```

5.4.4.2.4 Cryptographic techniques

To provide authentication and integrity, transactions are signed using the Elliptic Curves Digital Signature Algorithm (ECDSA) with parameters **secp256k1**, as in Bitcoin [1]. Each honest process assigns a strictly monotonically increasing sequence number to its transactions. The network communications use gRPC between users and processes and raw TCP sockets between processes, but all communication channels are encrypted through SSL. Finally, the exclusion protocol (Algorithm 11) uses ECDSA for authenticating the sender of messages responsible for disagreements (i.e., for PoFs). Unlike ECDSA, threshold encryption cannot be used to trace back the faulty users as they are encoded in less bits than what is needed to differentiate users, and message authentication codes (MACs) are insufficient to provide this transferable authentication [182].

5.4.5 The Zero-Loss Blockchain proofs

In this section, we prove the properties of ZLB to solve LLB depending on the voting threshold h' used by the exclusion and inclusion consensus. We also generalize results to the voting threshold h of ASMR consensus. Following, we discuss three options for h' , analyze their advantages and disadvantages, and discuss an additional desirable property, which we call awareness.

5.4.5.1 α -Confirmation

We show in Theorem 5.7 that if a process delivers $c > n - h + \alpha n$ distinct certificates, then either it confirms that no coalition of size αn could have caused a disagreement, or it finds a disagreement.

Theorem 5.7. Let σ be an accountable consensus protocol with voting threshold h , and let honest process p_i decide v in an iteration of σ . If honest process p_i delivers certificates from $c > n - h + \alpha n$ distinct processes, $\alpha \in [0, 2/3]$, then p_i either detects a disagreement or α -confirms v .

Proof. p_i delivers certificates from $c > n - h + \alpha n$ processes, meaning that $c - \alpha n > n - h$ are certificates delivered from honest processes. As the total number of honest processes is $n - \alpha n$, then $x = n - \alpha n - (c - \alpha n) = n - c$ are the number of honest processes from which p_i has not delivered a certificate. For some of these x processes to have decided $v' \neq v$ then $x + \alpha n \geq h \iff c \leq n - h + \alpha n$. Thus, if $c > n - h + \alpha n$, either p_i has already received a certificate for v' , or else all honest processes decided on v , for $\delta \leq \alpha$. In the latter, this means that v is α -confirmed. \square

In contrast with Theorem 5.7, we show in Theorem 5.8 the maximum number of disagreements (or fork branches) a that an adversary of size $d + t$ can cause in one consensus instance.

Theorem 5.8 (number of branches). Let σ be a consensus protocol with voting threshold h . Suppose $d + t < h$ faulty processes cause a disagreement, and let a be the number of disagreeing decisions. Then, $a \leq \frac{n-(d+t)}{h-(d+t)}$ for $d + t \geq \frac{ah-n}{a-1}$.

Proof. For $d + t$ faults to be able to create a branches, then they must reach the voting threshold h with each of a different disjoint partitions of the honest processes. As these partitions are disjoint, each contains $(n - (d + t))/a$ processes (assume n divisible by a w.l.o.g.). This means that $\frac{n-(d+t)}{a} + d + t \geq h \iff d + t \geq \frac{ah-n}{a-1}$ for the attackers to be able to generate a branches. The equivalent equation in terms of the number of branches is $a \leq \frac{n-(d+t)}{h-(d+t)}$. \square

5.4.5.2 Exclusion and inclusion protocols

Theorem 5.9 (Consensus of exclusion/inclusion protocol). Let ZLB execute with voting threshold h , being $f_d = 2h_0 - n$ the minimum number of detected processes to start the membership change. Then the exclusion and inclusion protocols of the membership change solves consensus if their initial voting threshold h'_0 satisfies $h'_0 > \frac{d+t+n}{2}$ for safety and $h'_0 \leq n - f + f_d$ for liveness.

Proof. Honest processes start the exclusion protocol by locally excluding f_d processes that they detected as faulty through accountability. By Basilic's accountability (Lemma 5.24), these are at least f_d detected faulty processes. Let us thus w.l.o.g. assume that exactly f_d processes are detected and excluded (if it was more, i.e. $d_r \geq f_d$, then consensus is even easier thanks to active accountability). As such, we define $d' + t' = d + t - f_d$, and $n' = n - f_d$. The exclusion protocol executes an instance of Basilic with voting threshold h'_0 , but it actually starts with $h'(f_d) = h'_0 - f_d$, since it starts when f_d faulty processes are detected. Thus, this instance of Basilic solves consensus for $h'(f_d) > \frac{d' + t' + n'}{2}$ for safety and $h'(f_d) \leq n' - q - t'$ for liveness (Theorem 5.2).

We thus consider first the safety bound. $h'(f_d) > \frac{d' + t' + n'}{2} \iff h'(f_d) > \frac{d + t + n - 2f_d}{2} \iff h'_0 > \frac{d + t + n}{2}$, since the membership change starts with the advantage of having detected $f_d = 2h_0 - n$ faulty processes before starting. For the liveness bound, notice that the minimum number of Byzantine processes that will be detected at the start of the membership change is $t - t' \geq f_d - d$. Thus, $h'(f_d) \leq n' - q - t' \iff h'(f_d) \leq n - f_d - q - t - d + f_d \iff h'(f_d) \leq n - f \iff h'_0 \leq n - f + f_d$. \square

Theorem 5.10 (\diamond -Consensus of exclusion/inclusion protocol). Let ZLB execute with initial voting threshold h_0 , being $f_d = 2h_0 - n$ the minimum number of detected processes to start the membership change. Then the exclusion and inclusion protocols of the membership change solve \diamond -consensus if their voting threshold h_0 satisfies $h'_0 > d + t$ for safety and $h'_0 \leq n - f + f_d$ for liveness.

Proof. Honest processes start the exclusion protocol by locally excluding f_d processes that they detected as faulty through accountability. By Basilic's accountability (Lemma 5.24), these are at least f_d detected faulty processes. The exclusion protocol executes an instance of Basilic with voting threshold h' , which solves \diamond -consensus for $d' + t' < h'(f_d)$ for safety and $h'(f_d) \leq n' - q - t'$ for liveness (Theorem 5.6), with $n' = n - f_d$ and $d' + t' = d + t - f_d$.

We thus consider first the safety bound. $d' + t' < h'(f_d) \iff h'(f_d) > d + t - f_d \iff h'_0 > d + t$ by replacing $d' + t'$ by $d + t - f_d$ and because the membership change starts with the advantage of having detected $f_d = 2h_0 - n$ faulty processes before starting. For the liveness bound, notice that the minimum number of Byzantine processes that will be detected at the start of the membership change is $t - t' \geq f_d - d$. Thus, $h'(f_d) \leq n' - q - t' \iff h'(f_d) \leq n - f_d - q + f_d - t - d \iff h'(f_d) \leq n - f \iff h'_0 \leq n - f + f_d$. \square

For the standard voting threshold of the ASMR consensus of $h = 2n/3$, this means that there are two different optimal voting thresholds h' for both the exclusion and inclusion protocols, depending on whether we choose the membership change to solve consensus or eventual consensus. These thresholds are $h'_0 = 7n/9 = 2n'/3$ for consensus and $h'_0 = 2n/3 = n'/2$ for eventual consensus. We discuss now these two options, their advantages and drawbacks. We also propose an additional threshold that is resilient-optimal for an additional property, known as awareness.

5.4.5.2.1 Membership change solving eventual consensus

The bound $h'_0 = 2n/3 = n'/2$ means that the exclusion and inclusion protocols solve eventual consensus, as shown by Theorem 5.10. The advantage of this bound is that the deceitful ratio δ is optimal at $\delta < 2/3$. This is the optimal value because for $\delta = 2/3 = h/n$ faulty processes can cause a disagreement without even communicating with honest processes, meaning that they can cause infinite disagreements (i.e. one per user), not satisfying convergence. As such, for this voting threshold h' the total number of tolerated faults is $f < 2n/3$ with $q+t \leq n/3$, $d+t < 2n/3$. Unfortunately, since the exclusion and inclusion protocols solve only eventual consensus and not consensus, this means that some processes may temporarily disagree on the processes to exclude and to include. All processes will however eventually agree on the same set to include and to exclude, as shown in Section 5.3.6. Furthermore, a disagreement on the exclusion protocol is detrimental to the adversary, because honest processes will eventually exclude even more faulty processes. Even a disagreement of the inclusion protocol is detrimental to the adversary, since the disagreement on the included processes requires the adversary to expose even more faulty processes. These attackers could instead wait to expose themselves as faulty during a disagreement on the ASMR consensus, which is more beneficial to attackers. Nevertheless, we show now a different voting threshold h' that solves this vulnerability of the membership change.

5.4.5.2.2 Membership change solving consensus

Setting a voting threshold $h'_0 = 7n/9 = 2n'/3$ allows the exclusion and inclusion protocols to solve consensus, as shown in Theorem 5.9. Compared to the previous scenario, this voting threshold allows honest processes to be sure that they agree on the decisions of the exclusion and inclusion protocols. This means that if the inclusion protocol includes only honest processes, then by the end of the membership change the adversary cannot cause any more disagreements, and agreement is guaranteed from then on, provided all honest processes have started the membership change.

The disadvantage of such an approach is that the total number of tolerated faults for this threshold is $f < 5n/9$ with $q+t \leq n/3$, $d+t < 5n/9$. Moreover, this voting threshold does not suffice to guarantee that no disagreement is possible once the membership change terminates, because some honest processes may not even be aware yet of the existence of a membership change, and thus may still be using the outdated committee with $f < 5n/9$ faulty processes. For this reason, we define a new property, which we call awareness.

Definition 5.4.2 (Awareness). Suppose that the inclusion protocol only includes honest processes. Suppose a membership change starts. Then, ZLB satisfies awareness if all honest processes can fix $k > 0$ such that Φ_l will solve consensus $\forall l \geq k$ during the static period of the adversary.

The definition of awareness is strictly stronger than that of convergence in that it does not suffice for honest processes to know that eventually they will solve consensus, but they must be aware of when they stop just solving eventual consensus and start solving consensus (provided that the inclusion protocol does not restate the deceitful ratio back to where it was prior to the

membership change). Awareness is also strictly stronger than α -confirmation of the membership change, because awareness also guarantees that the remaining honest processes that have not yet even heard of the membership change, and are thus still deciding blocks with the outdated committee, cannot decide with the outdated committee once any honest process terminates the membership change.

Theorem 5.11. ZLB solves awareness if $d + t < h_0 + h'_0 - n$.

Proof. Let O be the set of honest processes that have not yet heard of a membership change. If $|O| + d + t \geq h_0$ then processes in O are enough to terminate consensus instance Φ_k with decision v , $k > 0$. Let O' be the set of honest processes that have started the membership change. Then if $|O'| + d + t \geq h'_0$ the membership change can terminate, and since $h'_0 \geq h_0$ by construction, once processes in O' can terminate the membership change, they can also terminate consensus instance Φ_k with decision v' , $v' \neq v$. Thus, we calculate for which values of f it is impossible for both $|O| + d + t \geq h_0$ and $|O'| + d + t \geq h'_0$ to be met. By solving the system of equations, for both to be possible then $d + t \geq h_0 + h'_0 - n$, which means that for $d + t < h_0 + h'_0 - n$ either processes in O can terminate Φ_k deciding v , or processes in O' can terminate deciding v' , but not both. \square

By Theorem 5.11, a voting threshold $h'_0 = 7n/9 = 2n'/3$, while solving consensus of the exclusion and inclusion protocols for $f < 5n/9$, only satisfies awareness for $f < 4n/9$. Instead, setting a voting threshold $h'_0 = 5n/6 = 5n'/9$ solves consensus of the exclusion and inclusion protocols for $f < n/2$ with $q + t \leq n/3$, $d + t < n/2$, and awareness for the same adversary. We discuss these three starting settings of ZLB and compare them with the state of the art in Section 5.4.6.

5.4.5.3 ZLB proofs of LLB

In this section, we show that ZLB solves LLB, regardless of the three possible starting parameters that we showed in the previous section.

Lemma 5.27. If $d + t < \min(h_0, h'_0)$ and $h'_0 \leq n - f + f_d$, then every disagreement in ZLB leads to a membership change whose inclusion and exclusion protocols eventually solves consensus.

Proof. If $d + t \geq h_0$ then faulty processes can cause disagreements without communicating with honest processes, meaning that disagreements are not detected and the membership change does not start. Thus, it follows that $d + t < h_0$. Theorem 5.10 shows that $d + t < h'_0$ and $h'_0 \leq n - f + f_d$ for the membership change to solve eventual consensus. \square

Theorem 5.12. ZLB satisfies convergence for $f = d + q + t$ total faults if $q + t \leq n - h_0$, $d + t < \min(h_0, h'_0)$ and $h'_0 \leq n - f + f_d$.

Proof. By Lemma 5.27 every membership change solves eventual consensus. The remaining bound $q + t \leq n - h_0$ follows from the fact that the ASMR consensus must at least solve eventual consensus (Lemma 5.21). If $d + t < 2h_0 - n$ from the start then there is no disagreements (Theorem 5.2) and thus convergence is guaranteed. Instead for $2h_0 - n \leq d + t < \min(h'_0, h_0)$

and if $f \leq n - h'_0 + f_d$, by Lemma 5.27 every disagreement leads to a membership change that solves eventual consensus. The inclusion protocol does not increase the deceitful ratio, since the inclusion protocol does not include more processes than the number of excluded processes by the exclusion protocol (thanks to the deterministic function `choose`) and all excluded processes are faulty.

As the inclusion consensus decides at least $h'(d_r) = h'_0 - d_r$ proposals and $d + t < h'_0$ (because we implement Basilic to solve SBC by deciding the union of all proposals with associated bit decided to 1), it follows that some proposals from honest processes will be decided. As the pool of joining candidates is finite and no process is included more than once, then in the worst case all faulty processes from the pool have been included at least once, and from then on all honest processes propose to include only honest processes from the pool. At this point, the deceitful ratio will decrease in every new membership change, within a static period of the slowly-adaptive adversary.

Some inclusion consensus will thus eventually lead to a deceitful ratio $\delta n < 2h_0 - n$ and consensus is satisfied from then on. Let Φ_k be the first ASMR consensus such that $\delta n < 2h_0 - n$. All previous iterations $k' < k$ solve eventual consensus because $d + t < h_0$ and $q + t \leq n - h_0$ (Theorem 5.6). \square

Corollary 5.10. ZLB solves Longlasting Blockchain with h_0 for $q + t \leq n - h_0$ and $d + t < h_0$ for any h'_0 satisfying $d + t < h'_0$ and $h'_0 \leq n - f + f_d$.

Proof. For $d + t < h_0$, $q + t \leq n - h_0$, by Theorem 5.6 ASMR consensus solves eventual consensus, satisfying termination. If $d + t < 2h_0 - n$, then by Theorem 5.2 ASMR consensus solves consensus, satisfying agreement. Finally, convergence is shown in Theorem 5.12. \square

5.4.6 Comparative table

We show in Table 5.3 a comparison of ZLB with the aforementioned three voting thresholds of the membership changed. Notice that Basilic with initial voting threshold $h_0 = \frac{2n}{3}$ is the only one to solve \diamond -consensus against more than a supermajority of faults, thanks to characterizing them in the BDB model. We represent three settings for ZLB depending on the initial voting threshold h'_0 of the membership change, but with the same initial voting threshold of ASMR consensus set to $h_0 = \frac{2n}{3}$. These are the three settings that we discussed in Section 5.4.5.2. In any of the three cases, notice however that Basilic and ZLB are the only to both solve consensus for a resilient-optimal number of faults of $3t < n$ in the BFT model, and also solve eventual consensus for a greater number of faults than $3t < n$. Furthermore, only the three settings of ZLB solve \diamond -consensus for a total number of faults $3f \geq n$ of which up to $t = t_\ell$ are Byzantine (as noted in the table's footnotes) while simultaneously solving consensus for the resilient-optimal bound of $3t < n$ in partial synchrony. The differences of the three settings of ZLB lie in the \diamond -consensus and awareness columns, and in the number of faults tolerated, as discussed in Section 5.4.5.2. Some works are represented in multiple rows [74, 73] because their tolerance to faults and assumptions varies depending on their starting configuration.

Blockchain	N.	Consensus		\Diamond -Consensus		LLB	Awareness	Acc.	Slashing	Zero loss	Act.
		Byz.	Total	Byz.	Total						
[1, 189]	S.	0	0	$2t < n$	$2f < n$	\times	\times	\checkmark [189]	\checkmark [189]	\times	\times
[183]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\times	\times	\times	\times
[74] (P)	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\times	\times	\times	\times
[74] (S)	S.	$2t < n$	$2f < n$	$2t < n$	$2f < n$	\times	\times	\times	\times	\times	\times
[73] (1)	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\times	\times	\times	\times
[73] (2)	P.	0	$f < \frac{2n}{3}$	0	$f < \frac{2n}{3}$	\times	\times	\times	\times	\times	\times
[23, 190].	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\times	\checkmark	\times	\times
[127, 132, 135]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\checkmark	\checkmark	\times	\times
[47]	P.	$3t < n$	$3f < n$	$3t < n$	$3f < n$	\times	\times	\checkmark	\times	\times	\times
[78]	P.	0	0	$2t < n$	$2f < n$	\times	\times	\checkmark	\times	\times	\times
Basilic ($h_0 = \frac{2n}{3}$)	P.	$3t < n$	$f < \frac{2n}{3}^\dagger$	$3t < n$	$f < n^\ddagger$	\times	\times	\checkmark	\times	\times	\checkmark
ZLB ($h'_0 = \frac{7n}{9}$)	P.	$3t < n$	$f < \frac{2n}{3}^\dagger$	$3(q+t) < n, d+t < \frac{5n}{9}^\S$		\checkmark^\P	$d+t < \frac{4n}{9}$	\checkmark	\checkmark	\checkmark	\checkmark
ZLB ($h'_0 = \frac{2n}{3}$)	P.	$3t < n$	$f < \frac{2n}{3}^\dagger$	$3(q+t) < n, d+t < \frac{2n}{3}^\S$		\checkmark	\times	\checkmark	\checkmark	\checkmark	\checkmark
ZLB ($h'_0 = \frac{5n}{6}$)	P.	$3t < n$	$f < \frac{2n}{3}^\dagger$	$3(q+t) < n, 2(d+t) < n^\S$		\checkmark^\P	$2(d+t) < n$	\checkmark	\checkmark	\checkmark	\checkmark

† Actually, $3(d+t) < n$ and $3(q+t) < n$, meaning that if $f = \lceil \frac{2n}{3} \rceil - 2$ then $t = 0$ (Figure 5.3b)

‡ Actually, $d+t < \frac{2n}{3}$ and $3(q+t) < n$, meaning that if $f = n - 2$ then $t = 0$ (Theorem 5.6)

§ In addition to ‡ as it implements on top of Basilic

¶ Membership change solves consensus not just \Diamond -consensus

Table 5.3: Comparative table of ZLB with previous work, where N. means the network assumption (S. for synchrony and P. for partial synchrony), Byz. means Byzantine faults tolerated, Acc. means accountability, and Act. active accountability.

5.4.7 Experimental evaluation

This section answers the following: Does ZLB offer practical performance in a geo-distributed environment? When $f < n/3$, how does ASMR perform compared to the HotStuff state machine replication that inspired Facebook Libra [183] and the recent fast Red Belly Blockchain [28]? What is the impact of large scale coalition attacks on the recovery of ASMR? We defer the evaluation of a zero-loss payment application to Section 5.5.

Selecting the right blockchains for comparison. As we offer a solution for open networks, we cannot rely on the synchrony assumption made by other blockchains [26]. As we need to reach consensus, we have to assume an unknown bound on the delay of messages [13], and do not compare against randomized blockchains [139, 164, 191, 160] whose termination is yet to be proven [186]. This is why we focus our evaluation on partially synchronous blockchains. We thus evaluated Facebook Libra [183], however, its performance was limited to 11 transactions per second, seemingly due to its Move VM overhead. Hence, we omit these results here and focus on its raw state machine replication (SMR) algorithm, HotStuff and its available C++ code that was previously shown to lower communication complexity of traditional BFT SMRs [79] (we use the unchanged original implementation in its default configuration [192]). We also evaluate the recent scalable Red Belly Blockchain [28] (RBB), and the Polygraph protocol [47] as it is, as far as we know, the only implemented accountable consensus protocol. Nevertheless, this protocol does not tolerate more than $n/3$ failures as it cannot recover after detection.

Geo-distributed experimental settings. We deploy the four systems in two distributed

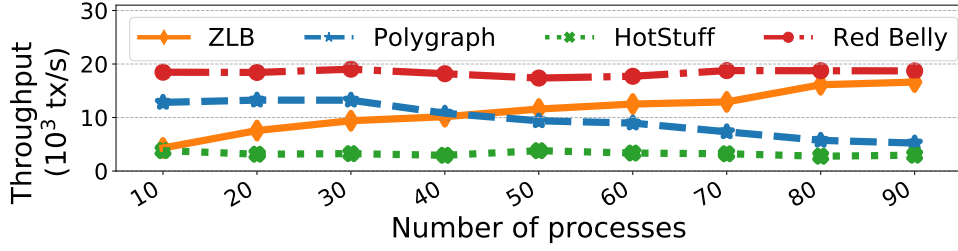


Figure 5.6: Throughput of ZLB compared to that of Polygraph [47], HotStuff [79] and Red Belly Blockchain [23].

settings of c4.xlarge Amazon Web Services (AWS) instances equipped with 4 vCPU and 7.5 GiB of memory: (i) a LAN with up to 100 machines and (ii) a WAN with up to 90 machines. We evaluate ZLB with a number of failures f up to $\lceil \frac{2n}{3} \rceil - 1$, however, when not specified we fix $f = d = \lceil 5n/9 \rceil - 1$ and $q = 0$. Since the impact of selecting a different $h'_0 \in [2n/3, n]$ is negligible in terms of throughput, we fix for this section $h'_0 = 7n/9$. Notice that for this threshold we can actually tolerate $d < 2n/3$ deceitful faults, provided that they are all detected at the start of the membership change, i.e. $d_r = 2n/3$. This can happen if all attackers collude together to maximize the number of branches that they can cause a disagreement for, as we do in our attack. All error bars represent the 95% confidence intervals and the plotted values are averaged over 3 to 5 runs. All transactions are ~ 400 -byte Bitcoin transactions with ECDSA signatures [1].

5.4.7.1 ZLB vs. HotStuff, Red Belly and Polygraph

Figure 5.6 compares the performance of ZLB, RBB, Libra and Polygraph deployed over 5 availability zones of 2 continents: California, Oregon, Ohio, Frankfurt and Ireland (exactly like the Polygraph experiments [47]). For ZLB, we only represent the decision throughput that reaches 16,626 tx/sec at $n = 90$ as the confirmation throughput is similar (16,492 tx/sec). As only ZLB tolerates $f \geq n/3$, we fix $f = 0$ for this comparison.

First, Red Belly Blockchain offers the highest throughput. As expected, it outperforms ZLB due to its lack of accountability: it does not require messages to piggyback certificates to detect PoFs. Both solutions solve SBC so that they decide more transactions (txs) as the number of proposals enlarges and use the same batch size of 10,000 txs per proposal. As a result ASMR scales pretty well: the cost of tolerating $f \geq n/3$ failures even appears negligible at 90 processes.

Second, HotStuff offers the lowest throughput even if it does not verify transactions. Note that HotStuff is benchmarked with its dedicated clients in their default configuration, they transmit the proposal to all servers to save bandwidth by having servers exchanging only a digest of each transaction. The performance is explained by the fact that HotStuff decides one proposal per consensus instance (i.e. one batch of 10,000 txs), regardless of the number of submitted transactions, which is confirmed by previous observations [81]. By contrast, ZLB becomes faster as n increases to outperform HotStuff by $5.6\times$ at $n = 90$, thanks to the superblock optimization that allows ZLB to decide multiple proposals at once per instance of its multi-valued consensus [23].

Finally, Polygraph is faster at small scale than ZLB, because Polygraph's distributed veri-

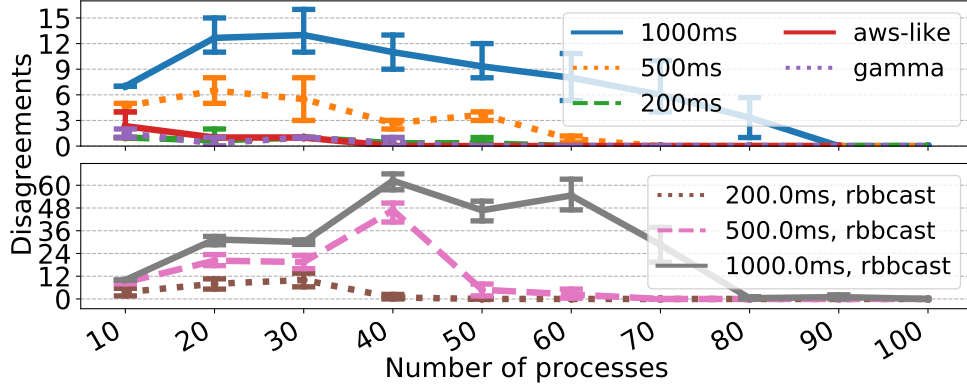


Figure 5.7: Disagreeing decisions for various uniform delays and for delays generated from a Gamma distribution and a distribution that draws from observed AWS latencies, when equivocating while voting for a decision (top), and while broadcasting the proposals (bottom), for $f = d = \lceil 5n/9 \rceil - 1$.

fication and reliable broadcast implementations [47] are not accountable, performing less verifications. From 40 processes on, Polygraph is slower because of our optimizations that remove redundant verifications and because Polygraph’s RSA verifications are larger than our ECDSA signatures and consume more bandwidth, even if we implement an accountable reliable broadcast in addition to an accountable consensus protocol.

5.4.7.2 Scalability of ZLB despite coalition attacks

To evaluate ZLB under failures, we implemented the following two possible coalition attacks. In Basilic, faulty processes can form a coalition of $f \geq n/3$ processes to lead honest processes to a disagreement by sending conflicting messages, with one of two coalition attacks:

1. **Reliable broadcast attack:** faulty processes misbehave during the reliable broadcast by sending different proposals to different partitions, leading honest processes to end up with distinct proposals at the same binary consensus index k . For example, faulty processes send block b_a with transaction tx_a to a subset A of honest processes, while block b_b with conflicting transaction tx_b to a subset B of honest processes, $A \cap B = \emptyset$, both at the same index k .

2. **Binary consensus attack:** faulty processes vote for each binary value in each of two partitions for the same binary consensus leading honest processes to decide different bits in the same index of their bitmask, where deciding 1 (resp. 0) at bitmask index k means to include (resp. not include) proposal at index k in ZLB. For example, faulty processes send messages to decide 1 and 0 to a subset of honest processes A , while they send messages to decide 0 and 1 to a subset B of honest processes, with $A \cap B = \emptyset$, on the binary consensus instances associated to block b_a with transaction tx_a and block b_b with conflicting transaction tx_b , respectively.

Note that faulty processes do not benefit from combining these attacks: If two honest processes deliver different proposals at index k , the disagreement comes from them outputting 1 at the corresponding binary consensus instance. Similarly, forcing two honest processes to disagree

during the k -th binary consensus only makes sense if they both have the same corresponding proposal at index k .

To disrupt communications between partitions of honest processes, we inject random communication delays between partitions based on the uniform and Gamma distributions, and the AWS delays obtained in previously published measurements traces [193, 194, 28]. (Attackers communicate normally with each partition.)

Figure 5.7(top) depicts the amount of disagreements as the number of distinct proposals decided by honest processes, caused by the binary consensus attack. First, we select uniformly distributed delays between the two partitions of 200, 500 and 1000 milliseconds. Then, we select delays following a Gamma distribution with parameters taken from previous work [193, 194] and a distribution that randomly samples the fixed latencies previously measured between AWS regions [28]. We automatically calculate the maximum amount of branches that the size of deceitful faults can create (i.e., 3 branches for $f + d < 5n/9$), we then create one partition of honest processes for each branch, and we apply these delays between any pair of partitions.

Interestingly, we observe that our agreement property is scalable: the greater the number of processes (maintaining the deceitful ratio), the harder for attackers to cause disagreements. This scalability phenomenon is due to an unavoidable increase of the communication latency between attackers as the scale enlarges, which gives relatively more time for the partitions of honest processes to detect the deceitful processes, hence limiting the number of disagreements. With more realistic network delays (Gamma distribution and AWS latencies) that are lower in expectation than the uniform delays, deceitful processes can barely generate a single disagreement. This confirms the scalability of our system.

Figure 5.7(bottom) depicts the amount of disagreements under the reliable broadcast attack. The number of disagreements is substantially higher during this attack than during the binary consensus attack. However, it drops faster as the system enlarges, because the attackers expose themselves earlier.

5.4.7.3 Disagreements due to failures and delays

We now evaluate the impact of even larger coalitions and delays on ZLB. We measure the number of disagreements as we increase the deceitful ratio and the partition delays in a system from 20 to 100 processes. Note that these delays could be theoretically achieved with man-in-the-middle attacks, but are notoriously difficult on real blockchains due to direct peering between the autonomous systems of mining pools [112].

While ZLB is quite resilient to attacks for realistic but not catastrophic delays (Figure 5.7), attackers can try to attack when the network collapses for a few seconds between regions. Our experiments, shown in Figure 5.9, show that attackers can reach up to 52 disagreeing proposals for a uniform delay of 10 seconds between partitions of honest processes for the binary consensus attack, and up to 33 disagreements for a uniform delay of 5 seconds, with $n = 100$. Further tests showed that the reliable broadcast attack reaches up to 165 disagreeing proposals with a 5-second uniform delay.

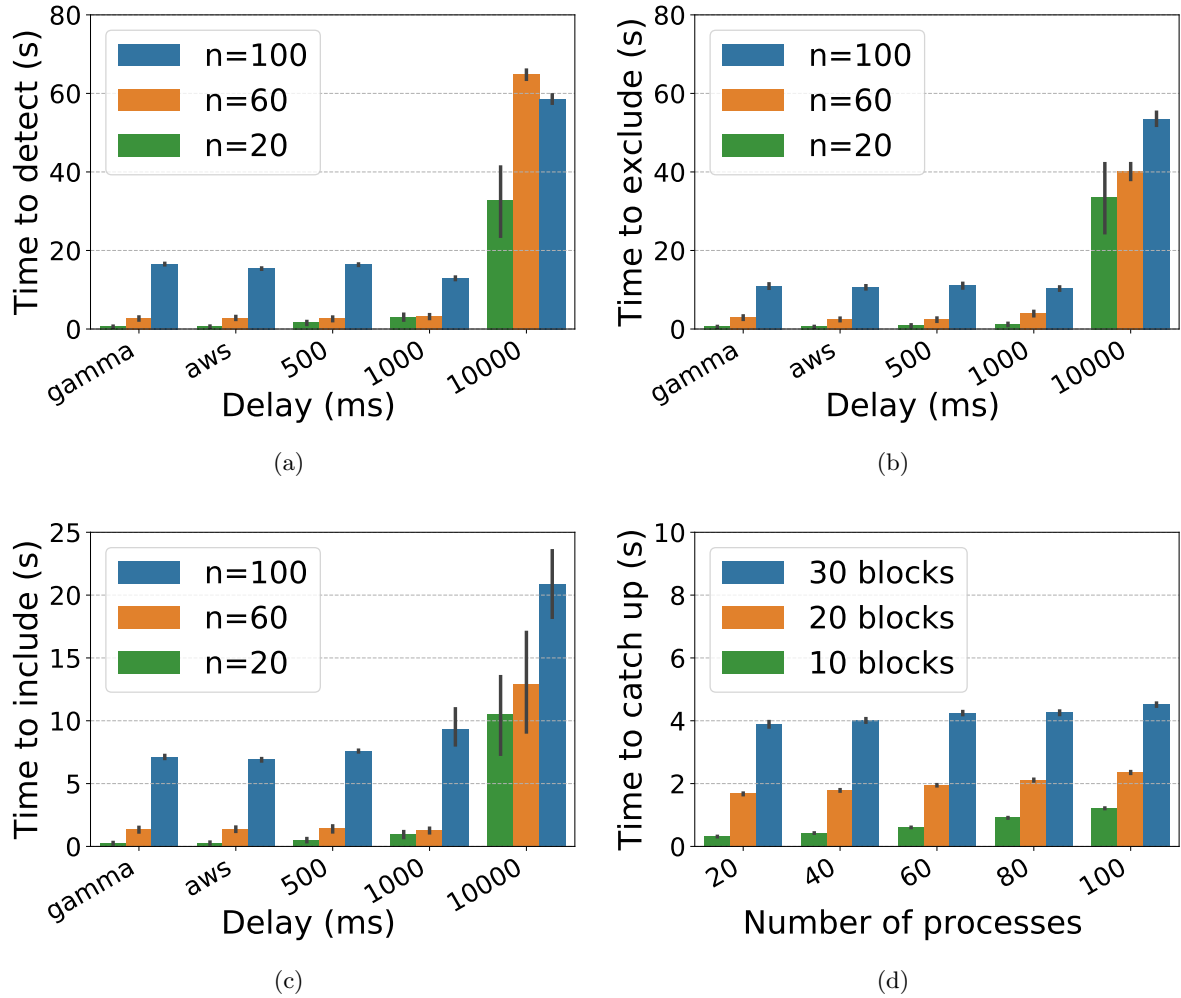


Figure 5.8: (Left to right, top to bottom) Time to detect $\lceil \frac{n}{3} \rceil$ deceitful processes, exclude them, include new processes, per delay distribution and number of processes; and catch up per number of blocks and processes, with $f = d = \lceil 5n/9 \rceil - 1$.

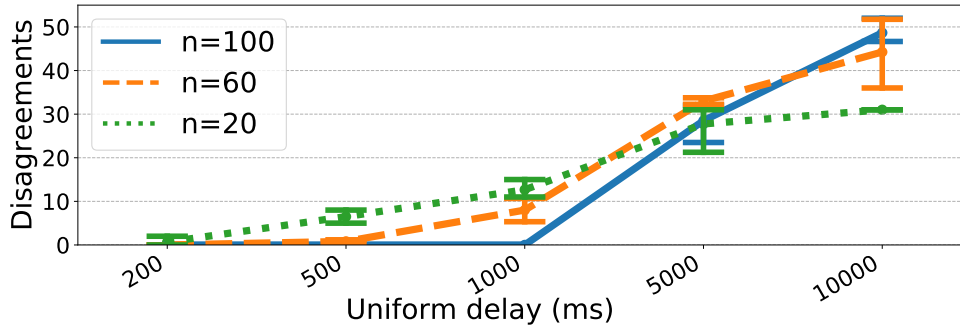


Figure 5.9: Disagreeing decisions for various catastrophic uniform delays with the binary consensus attack, for $f = d = \lceil 5n/9 \rceil - 1$.

5.4.7.4 Time to merge blocks and change members

To have a deeper understanding of the cause of ZLB delays, we measured the time needed to merge blocks and to change members by replacing deceitful processes by new ones. We show here the times to locally merge two blocks for different sizes assuming the worst case: all transactions conflict. This is the time taken in the worst case because processes can merge proposals that they receive concurrently (i.e., without halting consensus). Our experiments show that the times to merge two blocks of 100, 1000, and 10000 transactions are 0.55, 4.20 and 41.38 milliseconds, respectively. It is clear that this time to merge blocks locally is negligible compared to the time it takes to run the consensus protocol.

Figure 5.8 shows the time to detect f_d deceitful (top left), and to run the exclusion (top right) and inclusion (bottom left) consensus, for a variety of delays and numbers of processes. The time to detect reflects the time from the start of the attack until honest processes detect the attack: If the first f_d deceitful processes are forming a coalition together and cause a disagreement, then the times to detect the first deceitful and the first f_d deceitful processes overlap. (We detect all at the same time.) The time to exclude (57 seconds) is significantly larger than to include (21 seconds) for large communication delays, due to the proposals of the exclusion consensus carrying PoFs and leading processes to execute a time consuming cryptographic verification. With shorter communication delays, performance becomes practical. Finally, Figure 5.8 (bottom right) depicts the time to catch up depending on the number of proposals (i.e., blocks). As expected, this time increases linearly with the number of processes, due to the catchup requiring to verify larger certificates, but it remains practical at $n = 100$ processes. The advantage of using certificates is that processes can catch up by just verifying certificates, instead of having to verify all transactions in the block that the certificate refers to.

5.5 A Zero-Loss payment application

In this section, we describe how ZLB can be used to implement a *zero-loss payment system* where no honest process loses any coin. The key idea is to request the consensus processes to deposit a sufficient amount of coins in order to spend, in case of an attack, the coins of deceitful processes to avoid any honest process loss.

5.5.1 Assumptions

In order to measure the expected impact of a coalition attack succeeding with probability ρ in forking ZLB by leading a consensus to a disagreement, we first need to make the following assumptions:

1. **Fungible assets.** We assume that users can transfer assets (like coins) that are *fungible* in that one unit is interchangeable and indistinguishable from another of the same value. An example of a fungible asset is a cryptocurrency.

This zero-loss payment system can also work with non-fungible tokens (NFTs) and smart contracts, with the exception that one of the two recipients of the same NFT (or of disagreeing states) will see their NFT taken back (or their returned state reverted) in exchange for a previously agreed-upon reimbursement for the inconvenience.

2. **Deposit refund.** To limit the impact of one successful double-spending on a block, ZLB keeps the deposit for a number of blocks w , before returning it. A transaction should not be considered *final* (i.e. irreversible) until it reaches this *blockdepth* w . We call thus w the *finalization blockdepth*. Attackers can fork into a branches, and try to spend multiple times an amount \mathfrak{G} (per block), which we refer to as the *gain*, obtaining a maximum gain of $(a - 1)\mathfrak{G}$. Each honest process can calculate the gain by summing up all the outputs of all transactions in their decided block. Additionally, processes can limit the gain to an upper-bound by design, discarding blocks whose sum of outputs exceeds the bound, or they can allow the gain to be as much as the entire circulating supply of assets. The *deposit* \mathfrak{D} is a factor of the gain, i.e., $\mathfrak{D} = b \cdot \mathfrak{G}$. The goal is for every coalition to have at least \mathfrak{D} deposited, and since every coalition has at least size $\lceil n/3 \rceil$, this means that each process must deposit an amount $3b\mathfrak{G}/n$.

3. **Network control restriction.** Once faulty processes select the disjoint subsets (i.e., the partitions) of honest processes to suffer the disagreement, we need to prevent faulty processes from communicating infinitely faster than honest processes in different partitions. More formally, let X_1 (resp. X_2) be the random variables that indicate the time it takes for a message between two processes within the same partition (resp. two honest processes from different partitions). We have $E(X_1)/E(X_2) > \varepsilon$, for some $\varepsilon > 0$. Note that the definition of X_1 also implies that it is the random variable of the communication time of either two honest processes of the same partition or two faulty processes. This probabilistic synchrony assumption is similar to that of other blockchains (e.g. Bitcoin) that guarantee exponentially fast convergence, a result that also holds for ZLB under the same assumptions. In the following, we show an analysis focusing on the attack at each consensus iteration, considering a successful disagreement if there is a fork in a single consensus instance, even for a short period of time. We discuss in Section 5.5.3 the use of a random beacon for committee sortition in order to satisfy zero loss in a partially synchronous communication network.

5.5.2 Theoretical analysis

We show that attackers always fund at least as much as they steal. For ease of exposition, we consider that a membership change starts before the deposit is refunded or does not start, giving an advantage to the adversary in this analysis compared to the real scenario. Therefore, the attack represents a Bernoulli trial that succeeds with probability ρ (per block) that can be derived from ε . Out of one attack attempt, the attackers may gain up to $(a - 1)\mathfrak{G}$ coins by forking into a branches, or lose at least \mathfrak{D} coins as a punishment, which can be used to fund the stolen funds from successful attacks.

We introduce the random variable Y that measures the number of attempts for an attack to succeed and follows a geometric distribution with mean $E(Y) = \frac{1-\hat{\rho}}{\hat{\rho}}$, where $\hat{\rho} = 1 - \rho$ is the probability that the attack fails. Thus, we define the expected gain of attacking: $\mathcal{G}(\hat{\rho}) = (a - 1) \cdot (\mathbb{P}(Y > w) \cdot \mathfrak{G})$, and the expected punishment as: $\mathcal{P}(\hat{\rho}) = \mathbb{P}(Y \leq w) \cdot \mathfrak{D}$. We can then define the expected *deposit flux* per attack attempt as the difference $\xi = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$. Theorem 5.13 shows the values for which ZLB yields zero loss.

Theorem 5.13 (Zero-Loss Payment System). Let ρ be the probability of success of an attack per block, \mathfrak{D} the minimum deposit per coalition expressed as a factor of the upper-bound on the gain $\mathfrak{D} = b\mathfrak{G}$, and w the finalization blockdepth to return the deposit. If $g(a, b, \rho, w) = (1 - \rho^{w+1})b - (a - 1)\rho^{w+1} \geq 0$ then ZLB implements a zero-loss payment system.

Proof. Recall that the maximum gain of a successful attack is $\mathfrak{G} \cdot (a - 1)$, and the expected gain $\mathcal{G}(\hat{\rho})$ and punishment $\mathcal{P}(\hat{\rho})$ for the attackers in a disagreement attempt are as follows:

$$\begin{aligned}\mathcal{G}(\hat{\rho}) &= (a - 1) \cdot (\mathbb{P}(Y > w) \cdot \mathfrak{G}) = (a - 1) \cdot (\rho^{w+1} \cdot \mathfrak{G}), \\ \mathcal{P}(\hat{\rho}) &= \mathbb{P}(Y \leq w) \cdot \mathfrak{D} = (1 - \rho^{w+1})\mathfrak{D} = (1 - \rho^{w+1})b\mathfrak{G}.\end{aligned}$$

Thus the deposit flux $\xi = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$:

$$\xi = ((1 - \rho^{w+1})b - (a - 1)\rho^{w+1})\mathfrak{G} = g(a, b, \rho, w)\mathfrak{G}.$$

If $\xi < 0$ then a cost of $\mathcal{G}(\hat{\rho}) - \mathcal{P}(\hat{\rho})$ is incurred to the system, otherwise the punishment is enough to fund the conflicts. Since the gain is non-negative $\mathfrak{G} \geq 0$, it follows that $g(a, b, \rho, w) \geq 0$ for $\xi \geq 0$, obtaining zero loss. \square

Note that without some form of synchrony, the probability of success of an attack is $\rho = 1$ and thus no collateral is ever enough to satisfy zero-loss. This is because the attackers can always ensure that they perform the attack and retrieve back their collateral before they are caught by any correct process. On the other end, a fully synchronous assumption means a probability of success of $\rho = 0$ if the collateral is returned only after correct processes ensure they have waited enough time to have detected a hypothetical disagreement. This is why probabilistic synchrony not only is a better representation of reality (in which the network can often be influenced by not perfectly controlled neither by correct processes nor by the adversary), but also the one that enables this analysis here outlined.

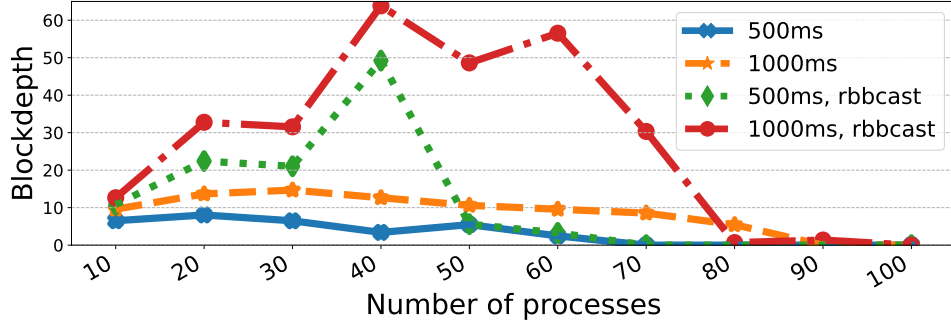


Figure 5.10: Minimum finalization blockdepth w to obtain zero loss for $\mathfrak{D} = \mathfrak{G}/10$, $f = d = \lceil 5n/9 \rceil - 1$ and $q = 0$.

5.5.2.1 Finalization blockdepth and deposit size

Setting $c = \frac{b}{a-1+b}$, we can either calculate the probability $\rho \leq c^{\frac{1}{w+1}}$ of success for an attack that ZLB tolerates given a finalization blockdepth w , or a needed finalization blockdepth $w \geq \frac{\log(c)}{\log(\rho)} - 1$ for a probability ρ to yield zero loss, once we fix the deposit \mathfrak{D} and upper-bound the gain \mathfrak{G} . For example, for $\delta = 0.5$ then $a = 3$, and for a probability $\rho = 0.55$, a finalization blockdepth of $w = 4$ blocks guarantees zero loss even if the deposit is a tenth of the maximum gain $\mathfrak{D} = \mathfrak{G}/10$, but with $\rho = 0.9$ then $w = 28$. Whereas a increases polynomially with ρ , it increases exponentially as the deceitful ratio δ approaches the asymptotic limit $2/3$, leading to $w = 37$ blocks for $\delta = 0.6$, while $w = 46$ for $\delta = 0.64$, or $w = 58$ for $\delta = 0.66$, with $\rho = 0.9$ and $\mathfrak{D} = \mathfrak{G}/10$ (Figure D.1.1).

5.5.2.2 Experimental evaluation of the payment system

Taking the experimental results of Section 5.4.7 and based on our aforementioned theoretical analysis, Figure 5.10 depicts the minimum required finalization blockdepth w for a variety of uniform communication delays for $\mathfrak{D} = \mathfrak{G}/10$, $f = d = \lceil 5n/9 \rceil - 1$ and $q = 0$. Again, we can see that the finalization blockdepth decreases with the number of processes, confirming that the zero loss property scales well. Additionally, small uniform delays yield zero loss at smaller values of w , with all of them yielding $w < 5$ blocks for $n > 80$. Although omitted in the figure, our experiments showed that even for a uniform delay of 10 seconds, setting $w = 50$ blocks (resp. $w = 168$ blocks) still yields zero loss in the case of a binary consensus attack (resp. reliable broadcast attack). Nevertheless, if the network performs normally, ZLB will support large values of f , and will actually benefit from attacks, obtaining more than enough funds to cover the stolen amount.

5.5.3 Discussion on probabilistic synchrony

We assumed probabilistic synchrony in Section 5.5 in order to introduce a probability of failure of an attack per consensus iteration. In partial synchrony, since the committee remains static until fraudsters are identified, the adversary can successfully perform an attack with probability of success $\rho = 1$. There are, however, other factors that could influence the probability ρ even in partial synchrony. For example, considering a blockdepth $w \geq 1$, the implementation of a

random beacon [26] that replaces the committee in every iteration can decrease the probability of success of an attack. In such a case, the probability of an attack succeeding depends on the probability that the random beacon selects enough processes of the coalition (and enough of each of the partitions of honest processes) for $w + 1$ consecutive iterations, so that the coalition is able to perform the attack for w additional blocks. The design and proof of a random beacon that tolerates coalitions of sizes greater than t_ℓ is part of Chapter 6.

5.6 Summary

In this chapter, we extended the TRAP protocol and results from Chapter 4 by presenting the BDB failure model. We then presented the Basilic class of protocols, that is resilient-optimal for the problem of consensus in both the BFT and BDB model, and optimal in the communication complexity, thanks to the active accountability property that states that deceitful behavior does not prevent liveness. We also showed that the Basilic class of consensus protocols solves the \diamond -consensus problem in the BDB model for $d + t < h_0$ and $q + t \leq n - h_0$, with $h_0 \in (n/2, n]$ being the initial voting threshold.

We then presented ZLB, a blockchain that tolerates a majority of faults. To this end, we first presented the LLB problem, to then detail ZLB and prove ZLB's correctness. Following, we built and evaluated ZLB against a majority of attackers, and compared it with previous works, offering competitive performance. We finally presented a zero-loss payment application built on top of ZLB that guarantees that no honest process or user loses any fund from temporary disagreements.

The motivation for ZLB and Basilic comes from the need to extend the tolerance to stronger adversaries as presented by TRAP in Chapter 4, not only for consensus, but to the problem of blockchains, or repeated consensus. However, both in Chapter 4 and in the zero-loss payment application of this chapter, we identified that a random beacon can mitigate disagreement attacks or even prevent them, thanks to attackers being replaced in the committee. This is what motivates Chapter 6 of this dissertation, the design of a novel random beacon that can be used for committee sortition.

Chapter 6

Kleroterion⁺, Randomness With Colluding Majorities

Our results from previous chapters make an effort in tolerating faults or deviations of more than t_ℓ participants. While these results are particularly relevant for blockchains, the results of these chapters do not apply yet to blockchains without synchrony, in that Chapter 4 considers a single-shot consensus and Chapter 5 needs to at least assume probabilistic synchrony to guarantee zero loss. In this chapter, we first show that a protocol for committee sortition implemented with a random beacon protocol allows for adopting the results from these chapters to repeated consensus in partial synchrony. For this reason, we first propose Kleroterion, a novel random beacon protocol that modifies the recent SPURT random beacon protocol [31] to make it more scalable. Kleroterion exchanges a number of bits per network channel independent of the size of the participants in the protocol per random output, except for one message of size n sent by the leader of the epoch and for the reconstruction phase, and without requiring a trusted setup. Then, we build Kleroterion⁺, an extension of Kleroterion that trades some of the optimizations of Kleroterion for tolerating colluding majorities.

Summary. In summary, we present the following contributions in this chapter:

1. We outline the need for a random beacon for committee sortition by analyzing the results from Chapters 4 and 5.
2. We explore the implications of sorting the committee for Platypus, TRAP, and ZLB.
3. We propose Kleroterion, a democratic random beacon protocol, and Pinakion, a protocol for PVSS that is used by Kleroterion.
4. We formulate the random beacon problem tolerating coalitions of up to t_s processes by stating the secure random beacon (SRB) problem, and the same for the PVSS problem with the accountable PVSS (APVSS) problem.
5. We present Kleroterion⁺ and Pinakion⁺, a secure random beacon protocol and APVSS protocol, respectively.

6. We analyze the security of Kleroterion⁺ for committee sortition in a blockchain application and compare it with the state of the art.

Chapter outline. In Section 6.1 we analyze the results of previous chapters for blockchains without synchrony and justify the need for a random beacon. Section 6.2 presents Kleroterion, a democratic random beacon with quadratic communication complexity and constant per route complexity for commitment and aggregation, and Pinakion, our novel PVSS protocol. We extend Kleroterion to tolerate colluding majorities in Section 6.3, where we first state the SRB and APVSS problems, by presenting Kleroterion⁺ and Pinakion⁺. We analyze the implications of using Kleroterion⁺ for blockchains and compare it with the state of the art in Section 6.4. Section 6.5 concludes the chapter.

6.1 The need for a random beacon

We justify in this section the need for a random beacon for blockchains based on BFT consensus without synchrony.

6.1.1 The need to depreciate future iterations

Suppose a coalition that contains some rational players (or deceitful faults) and is able to cause a disagreement. Let there be an attack of this coalition that lasts for x blocks. The (maximum) total gain at block i for the coalition is $\forall i, \mathfrak{G}_i = \mathfrak{G}$. We first consider that all decided blocks are finalized with a finalization blockdepth of $w = 1$. We consider a *discount factor* per block $\beta \in [0, 1)$ that decreases the gain for future iterations. Therefore, the expected gain of the attack lasting for x blocks is:

$$\sum_{i=0}^{x-1} \beta^i (i+1) \mathfrak{G} \quad (6.1)$$

Similarly, the attack can last forever by setting the limit: $\lim_{x \rightarrow \infty} \sum_{i=0}^{x-1} \beta^i (i+1) \mathfrak{G} = \sum_{i=0}^{\infty} \beta^i (i+1) \mathfrak{G}$. It is easy to see that $\beta < 1$ since otherwise the expected gain does not converge and tends to ∞ . The discount factor β is generally close to 1 in a normal system without attacks, due to the time value of money¹. However, the time value of money does not suffice to make β low enough to depreciate the impact of an attack that lasts for future consensus iterations.

As such, we need to decrease the expected gain from future iterations by increasing the probability that the attackers will be caught, so that either rational players will not deviate (robustness), or, even if all processes are deceitful and Byzantine and deviate, then the slashed amount will fund the stolen assets (zero loss).

Hence, the results from Chapters 4 and 5 can be applied to blockchains without synchrony by devising ways of iteratively knocking down future iterations through a discount factor β . Unfortunately, these chapters do not provide a discount factor in partial synchrony that ensures that the expected gain from causing a disagreement converges such that the reward can

¹https://en.wikipedia.org/wiki/Time_value_of_money

implement a baiting strategy or satisfy zero loss, other than the fact that β will be 0 after GST (because honest processes will see the disagreement).

We explore in Section 6.1.2 how one can obtain such discount factor in partial synchrony with a committee sortition protocol.

6.1.2 Depreciating future iterations with a random beacon

We assume that there is a committee sortition protocol that is random and sorts a new committee at the end of each consensus iteration.

Suppose a coalition \mathcal{C} of size $|\mathcal{C}| = \mathbf{c}_m$ players out of a pool of all users m that can be selected for the committee. At the beginning of each consensus iteration i , the committee sortition protocol selects $n < m$ processes that compose the committee for that iteration, we refer to the set of this committee as N_i , and we abuse notation by referring to N for the committee of the current consensus iteration. Let \mathbf{c}_n describe the number of colluding processes in \mathcal{C} that are part of the current committee N . Suppose that in the current consensus iteration $\mathbf{c}_n > t_\ell = \lceil n/3 \rceil - 1$. It is clear that, in the partially synchronous model, the discount factor for the next consensus iteration is at most $\Pr(\mathbf{c}_n > t_\ell)$, i.e. the probability of the next committee containing at least $t_\ell + 1$ faults (for a finalization blockdepth $w = 1$). This is because if in the next iteration $\mathbf{c}_n \leq t_\ell$ then the attackers cannot continue causing a disagreement, and thus by accountability they will be caught before honest processes terminate this next iteration. Hence, we calculate this probability:

$$\begin{aligned} \Pr(\mathbf{c}_n = j) &= \binom{\mathbf{c}_m}{j} \frac{\binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}} \\ \Pr(\mathbf{c}_n > t_\ell) &= 1 - \Pr(\mathbf{c}_n \leq t_\ell) = 1 - \sum_{j=0}^{j \leq t_\ell} \Pr(\mathbf{c}_n = j) \\ &= 1 - \sum_{j=0}^{j \leq t_\ell} \binom{\mathbf{c}_m}{j} \frac{\binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}} \end{aligned}$$

It is worth noting that if $\mathbf{c}_m \geq h$ (where $h \in (n/2, n]$ is the voting threshold, e.g. $h = 2n/3$) in one iteration then attackers can virtually generate infinite disagreements in that iteration (i.e. the number of branches a of the disagreement is only bounded by the number of users), but we show in Section 6.3 that this probability is negligible and should never occur if the committee size is big enough, depending on the percentage of users controlled by the adversary.

Notice that it is actually not enough for the coalition to sustain the proportional size of the committee for the next block, i.e. $\beta \lesssim \Pr(\mathbf{c}_n > t_\ell)$. In fact, for an equivocation attack into a different decisions (i.e. a branches of a fork), it is necessary to split the remaining honest processes into P_a partitions such that in each of the iterations there are enough honest processes of each partition to reach the voting threshold h and finalize the disagreement (Theorem 5.8). This is because if an honest process p_i is sent the disagreeing branch deriving from a partition P_1 , and then selected for the next committee and given a block that extends the branch from a different partition P_2 , p_i will not contribute to extending the branch for the partition P_2 , hindering the attack and facilitating detecting attackers.

This way, the partition must guarantee that enough members of each partition are selected in each of the consecutive consensus iterations for the attack to proceed in all partitions. Let $\Pr(|P_{i_n}| \geq h-j)$ be the probability that the proportional size of the partition P_i in the committee of size n is greater or equal to $h-j$, then the actual value of β is upper bounded by the following function $f_\beta(a)$:

$$\begin{aligned} f_\beta(a) &= \sum_{j>t_\ell}^{h-1} \Pr(\mathbf{c}_n = j) \cdot \prod_{i=0}^{a-1} \Pr(|P_{i_n}| \geq h-j) + \sum_{j=h}^n \Pr(\mathbf{c}_n = j) = \\ &= \sum_{j>t_\ell}^{h-1} \binom{\mathbf{c}_m}{j} \frac{\binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}} \cdot \prod_{i=0}^{a-1} \Pr(|P_{i_n}| \geq h-j) + \sum_{j=h}^n \binom{\mathbf{c}_m}{j} \frac{\binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}} \end{aligned} \quad (6.2)$$

We further unfold this equation for the case $a = 2$ with two partitions A_m and B_m , with $\mathbf{c}_m + |A_m| + |B_m| = m$:

$$\beta \preceq f_\beta(2) = \sum_{j>t_\ell}^{h-1} \sum_{k=h-j}^{n-(h-j)-j} \binom{\mathbf{c}_m}{j} \binom{|A_m|}{k} \frac{\binom{m-\mathbf{c}_m-|A_m|}{n-j-k}}{\binom{m}{n}} + \sum_{j=h}^n \binom{\mathbf{c}_m}{j} \frac{\binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}}$$

Notice the previous example does not give the value of $|A_m|$, in particular, as long as A_m and B_m are disjoint, both $|A_m| \geq h-j, |B_m| \geq h-j$ and also $|A_m| + |B_m| + \mathbf{c}_m = m$, any partition of honest processes into these two subsets would suffice. We also assume that $\mathbf{c}_m > n$ for ease of exposition. We explore any subset of honest players of size $\frac{m-\mathbf{c}_m}{2}$ to make A_m , with the remaining honest processes constituting B_m .

As such, we define the random variable Y that yields the number of blocks that an attack lasts for before being detected. It is immediate that $\Pr(Y = j) = f_\beta(a)^j (1 - f_\beta(a))$. The attack represents thus a Bernoulli trial with probability of $\rho = f_\beta(a)$. As a result, Y follows a geometric distribution with mean $E(Y) = \frac{1-\hat{\rho}}{\hat{\rho}}$ blocks, where $\hat{\rho} = 1 - \rho$ is the probability that the attack fails.

The expected gain $\mathcal{G}(\hat{\rho})$ from an attack like this for $w = 1$ is thus:

$$\mathcal{G}(\hat{\rho}) = \sum_{j \geq w} \Pr(Y = j) j(a-1) \cdot \mathfrak{G} \quad (6.3)$$

$$\mathcal{G}(\hat{\rho}) = \sum_{j \geq w} f_\beta(a)^j (1 - f_\beta(a)) j(a-1) \cdot \mathfrak{G} \quad (6.4)$$

Pipelining attacks. We also assume a deposit \mathfrak{D} as defined in Section 5.5.2. Contrary to the analysis based on probabilistic synchrony from Section 5.5.2, in this model attackers will always be identified in the first iteration that does not continue with the disagreement (i.e. the first iteration such that $\mathbf{c}_n < t_\ell$), and thus attackers maximize their gain by pipelining attacks.

Attackers pipeline attacks by overlapping finalizing an attack with starting a new one for $w > 0$. This means that if an attack starts at block i , and it must last for w until the $(i+w)$ -th block in order to be finalized, then the $(i+1)$ -th block helps finalize i -th block at the same time that it starts a new attack that will be finalized at the $(i+w+1)$ -th block. We show then Equation 6.3 for any $w \geq 0$:

$$\mathcal{G}(\hat{\rho}) = \sum_{j \geq w} f_\beta(a)^j (1 - f_\beta(a)) (j-w+1)(a-1) \cdot \mathfrak{G}$$

where $j - w + 1$ represents the pipelined attacks, and a the number of branches in each of the pipelined, attacked consensus iterations. If instead $j < w$ then $\mathcal{G}(\hat{\rho}) = 0$. Once attackers are caught then they will lose exactly $\mathfrak{D} = b\mathfrak{G}$, $b > 0$. Therefore, the deposit flux ξ (Section 5.5.2) results to:

$$\xi = \mathfrak{D} - \mathcal{G}(\hat{\rho}) = \left(b - \sum_{j \geq w} f_{\beta}(a)^j (1 - f_{\beta}(a)) (j - w + 1) (a - 1) \right) \cdot \mathfrak{G} \quad (6.5)$$

Implications to zero loss and rational agreement. As already noted (Section 5.5.2), if $\xi > 0$ in Equation 6.5 then we obtain zero loss from a committee sortition protocol without assuming probabilistic synchrony. Instead, if $\xi < 0$ but $r > |\xi|$ where r is the expected reward from exposing a disagreement, then the system can implement TRAP's baiting strategy for the repeated consensus problem of blockchains, and not just for a single-shot consensus. This shows that a random beacon can be used to implement a committee sortition protocol such that the results from TRAP and ZLB apply to blockchains without synchrony. For an analysis of why we focus on committee sortition based on random beacons and not for example on deterministic rotation or an election protocol, we refer to our explanation of the advantages of random beacons and dangers of these other rotation protocols in Section 2.1.5.

A random beacon for offchains. Since offchain protocols must rely on the blockchain that they are attached to (the parentchain), this means that protocols like Platypus (Chapter 3) can implement a childchain that solves LLB while guaranteeing zero loss as defined in Chapter 5, and that is (k, t) -robust and $(k + t, t)$ -crash-robust for $n > \max\left(\frac{3}{2}k + 3t, 2(k + t)\right)$. For this result, Platypus can implement a committee sortition protocol that uses the parentchain as a trusted mediator to randomly rotate the committee, and tweak the aforementioned values so that $\xi > 0$ (for zero loss in the BDB model), or instead $r > |\xi|$ if $\xi \leq 0$ (for ϵ -(k, t)-robustness).

A random beacon without a mediator. To the best of our knowledge, there is no committee sortition protocol that preserves its properties in the presence of a coalition of attackers of size greater than t_{ℓ} , without assuming a trusted third party, or synchronous communications. We present in the following Kleroterion, our proposal for a random beacon protocol that tolerates up to t_{ℓ} Byzantine faults, to then extend it to Kleroterion⁺, a random beacon that tolerates a colluding majority. We then showcase how to design a committee sortition protocol using instead Kleroterion⁺.

6.2 Kleroterion: a democratic random beacon

We present now Kleroterion, our proposal that solves the random beacon problem. Kleroterion solves the random beacon problem by executing n instances (one per process of the committee) of a PVSS protocol, which we refer to as Pinakion and show in Section 6.2.2, followed by a consensus protocol that selects $t_{\ell} + 1$ of the n secrets shared by the n instances of Pinakion. This is similar to how AABC instances select a number of AARB delivered values in Figure 5.2, except that instead of n AARB and n AABC instances, Kleroterion executes n Pinakion instances and one multi-valued consensus protocol (without a reduction to n binary consensus

instances), respectively. Even though it only executes one instance of a multi-valued consensus protocol, Kleroterion is still democratic (following its definition in Section 2.2.9), as the inputs are provided from different processes (given the n Pinakion instances).

We extend our model in Section 6.2.1. We show in Section 6.2.2 our PVSS protocol Pinakion. Section 6.2.3 shows the Kleroterion random beacon protocol, and Section 6.2.4 shows optimizations and observations for the Kleroterion protocol.

6.2.1 Additional model

Let \mathbb{G}_0 , \mathbb{G}_1 and \mathbb{G}_T be cyclic groups of prime order q and \mathbb{Z}_q the group of integer modulo q , and let λ be the security parameter such that $\lambda = \log_2(q)$. We assume that at the start of the protocol, all processes agree on public parameters $g_0, h_0 \in \mathbb{G}_0$ and $g_1, h_1 \in \mathbb{G}_1$, which are uniformly randomly and independently chosen generators of each cyclic group. This is known as a *common reference string* (CRS) setup. We iteratively execute our random beacon protocol, Kleroterion, with a static committee N of size $|N| = n$.

Bilinear pairings. Similarly to previous work [31], we rely on the decisional *bilinear Diffie-Hellman* assumption [195] (DBDH), for which we assume the reader is familiar with the standard definition of computationally indistinguishable distribution ensembles [196, 197]:

Definition 6.2.1 (Bilinear pairing). Let \mathbb{G}_0 , \mathbb{G}_1 and \mathbb{G}_T be three cyclic groups of prime order q where $g_0 \in \mathbb{G}_0$ and $g_1 \in \mathbb{G}_1$ are generators. A pairing is an efficiently computable function $e : \mathbb{G}_0 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ satisfying the following properties:

1. bilinearity: For all $u, u' \in \mathbb{G}_0$ and $v, v' \in \mathbb{G}_1$ we have:

$$\begin{aligned} e(u \cdot u', v) &= e(u, v) \cdot e(u', v), \text{ and} \\ e(u, v \cdot v') &= e(u, v) \cdot e(u, v') \end{aligned}$$

2. non-degeneracy: $g_T = e(g_0, g_1)$ is a generator of \mathbb{G}_T .

We refer to \mathbb{G}_0 and \mathbb{G}_1 as the pairing groups or source groups, and refer to \mathbb{G}_T as the target group.

Definition 6.2.2 (Decisional bilinear Diffie-Hellman). Given pairing groups G_0, G_1 , target group G_T , each of size q , let $e : G_0 \times G_1 \rightarrow G_T$ be an efficient bilinear pairing map. For generators $g_0 \in G_0, g_1 \in G_1$, random values $\alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{Z}_q$ and $a_0 \leftarrow g_0^\alpha, a_1 \leftarrow g_1^\alpha, b_0 \leftarrow g_0^\beta, b_1 \leftarrow g_1^\beta$, the following distributions D_0 and D_1 are computationally indistinguishable:

$$\begin{aligned} D_0 &= (a_0, a_1, b_0, b_1, e(g_0, g_1)^{\alpha\beta\gamma}) \\ D_1 &= (a_0, a_1, b_0, b_1, e(g_0, g_1)^\delta) \end{aligned}$$

In order to implement the Kleroterion protocol, we will use a variant of Shamir's threshold secret sharing [198] to implement a variant of the publicly-verifiable secret sharing (PVSS) Π_{DBDH} protocol [31]. We define threshold secret sharing here below, and our PVSS, Pinakion, in Section 6.2.2.

Zero knowledge proof of knowledge. Our Kleroterion protocol uses zero-knowledge proofs about equality of discrete logarithms in order to satisfy *knowledge soundness*, in that processes know the secret that they each are sharing. This guarantees to honest processes that their shared secrets are independent of any other shared secret.

In particular, given a CRS setup as mentioned above, $x \in \mathbb{G}_0$, $y \in \mathbb{G}_1$, each process p_i wants to prove that there exists a witness α such that $x = g_0^\alpha$ and $y = g_1^\alpha$, and that p_i knows α .

Thus, in addition to bilinear pairings, we use the non-interactive version of the Chaum-Pedersen Σ -protocol in the random oracle model [199, 200, 201], a standard cryptographic assumption of a perfect, uniformly random hash function (although in practice this will only be possible to implement pseudo-randomly). The knowledge soundness of this protocol implies that if p_i convinces an honest process p_j with non-negligible probability, there exists an efficient (polynomial time) extractor that can extract α from p_i with non-negligible probability. Let us denote by `dleq` the call to a non-interactive version of the Chaum-Pedersen protocol, such that `dleq.Prove`(α, g_0, x, g_1, y) generates the proof π and `dleq.Verify`(π, g_0, x, g_1, y) verifies the proof [31].

Threshold secret sharing. A (t, n) -threshold secret sharing scheme allows a process, known as the dealer, to share a secret $s \in \mathbb{Z}_q$ with n other processes, such that any $t + 1$ of them can reconstruct the message, but no t of them can. Analogously to SPURT [31], we also base off Shamir's secret sharing [198] scheme, in which a secret $s \in \mathbb{Z}_q$ is embedded in a polynomial $p(\cdot)$ of degree t such that $p(0) = a_0 = s$. The remaining t coefficients $\{a_i\}_{i=1}^t$ are chosen uniformly at random being thus $p(x) = \sum_{i=0}^t a_i x^i$.

The dealer then shares with process p_i the evaluation of $p(i)$. One can efficiently reconstruct the polynomial using Lagrange interpolation upon obtaining $t + 1$ evaluations of $p(x)$. Moreover, an adversary cannot learn the secret with any t or less evaluations of $p(x)$, except with the same probability of randomly guessing the secret.

Adversary. We let the adversary \mathcal{M} control $f = t \leq t_\ell$ Byzantine processes for Kleroterion, increasing the tolerance to faults to t_s for Kleroterion⁺ in Section 6.3.

6.2.2 The Pinakion protocol

In this section, we illustrate our Pinakion protocol that solves PVSS. We extend SPURT's [31] Π_{DBDH} PVSS. The setup phase is the same to that of Π_{DBDH} [31]:

PVSS.Setup(1^λ) $\rightarrow (g_0, h_0, g_1, h_1, (sk_i, pk_i))$: The setup algorithm chooses uniform random and independent generators $g_0, h_0 \in \mathbb{G}_0$ and $g_1, h_1 \in \mathbb{G}_1$ and publishes them in a trusted PKI (which is only used in this step). Each process p_i also generates a secret key $sk_i \in \mathbb{Z}_q$ and public key $pk_i = h_0^{sk_i}$, and publishes pk_i in the public ledger.

Algorithm 13 illustrates the rest of the Pinakion protocol. After the setup phase, the dealer p_d selects a *secret* s to share. For this purpose, processes select a polynomial $p(x)$ of degree t_ℓ whose coefficients have been chosen uniformly at random from \mathbb{Z}_q , such that $p(0) = s$ (line 9). Then, p_d computes *secret shares* $p(j) \forall j \in [n] \setminus \{i\}$ which it encrypts with the public key of the recipient $c_{j,d} = pk_j^{p(j)}$, obtaining the vector \mathbf{c}_d (line 11). Additionally, p_d also computes a non-

interactive zero-knowledge proof vector \mathbf{v}_d such that $v_{j,d} = g_1^{p(j)}$ that serves as a *commitment* to the secret shares and to verify the validity of the encrypted shares \mathbf{c}_d (lines 12-14).

Algorithm 13 Pinakion protocol with dealer p_d

```

1: State:
2:    $g_0, h_0 \in \mathbb{G}_0$ ;  $g_1, h_1 \in \mathbb{G}_1$ , uniformly random and independent generators
3:    $sk_i \in \mathbb{Z}_q$  secret key of  $p_i$ 
4:    $pk_j = h_0^{sk_j}$  public key of  $p_j$ , for  $j \in [n]$ 
5:    $S = e(h_0^s, h_1)$ , secret that process  $p_d$  shares, with  $s \xleftarrow{\$} \mathbb{Z}_q$ 

6: Pinakion.Share: ▷ executed by  $p_d$ 
7:   for  $j \in [1, t_\ell]$  do
8:      $a_k \xleftarrow{\$} \mathbb{Z}_q$ 
9:    $p(x) \leftarrow s + a_1x + \dots + a_{t_\ell}x^{t_\ell}$ 
10:  for  $j \in [0, n-1]$  do
11:     $c_{j,d} \leftarrow pk_j^{p(j)}$  ▷ encrypt with  $p_j$ 's public key
12:     $v_{j,d} \leftarrow g_1^{p(j)}$ 
13:   $\mathbf{v}_d \leftarrow \{v_{0,d}, v_{1,d}, \dots, v_{n-1,d}\}$ 
14:   $\mathbf{c}_d \leftarrow \{c_{0,d}, c_{1,d}, \dots, c_{n-1,d}\}$ 
15:  RBV-broadcast( $\{\mathbf{v}_d, \mathbf{c}_d\}$ ) ▷ distribution & Verification

16: Pinakion.Reconstruction: ▷ executed by each  $p_i$ 
17:   $shares \leftarrow \{\}$ 
18:  when ( $p_i$  RBV-delivers  $\{\mathbf{v}_d, \mathbf{c}_d\}$ ) do
19:     $s_{i,d} \leftarrow c_{i,d}^{1/sk_i}$  ▷ decrypt secret share
20:    broadcast( $s_{i,d}$ ) ▷ broadcast secret share
21:  when ( $s_{j,d}$  is delivered) do
22:    if (Pinakion.check( $h_0, v_{j,d}, s_{j,d}, g_1$ )) then  $shares[j] \leftarrow s_{j,d}$ 
23:    if (size( $shares$ ) >  $t_\ell$ ) then
24:       $h_0^s \leftarrow \text{Pinakion.interpolate}(h_0, h_1, shares)$  ▷ reconstruct
25:      return ( $e(h_0^s, h_1)$ )

26: Pinakion.check( $a, b, c, d$ ):
27:  return  $e(a, b) = e(c, d)$ 

```

Following, in line 15 process p_d calls RBV-broadcast with the commitments and encrypted shares $(\mathbf{v}_d, \mathbf{c}_d)$. The RBV-broadcast protocol is almost identical to the reliable broadcast protocol outlined by recent works [162, 23, 163]. The only modification we add is for honest processes to only deliver a message containing \mathbf{c}_d and \mathbf{v}_d from p_i if the verification of Pinakion checks in the calls to Pinakion.verify. We call this variant *reliable broadcast with verification* (RBV). We show in Algorithm 14 the RBV-broadcast protocol, which consists of an accountable reliable broadcast that covers the distribution and verification steps of the Pinakion protocol. In this case, p_d RBV-broadcasts the list of shares \mathbf{c}_d and zero-knowledge proofs \mathbf{v}_d . Our RBV-broadcast protocol ensures that honest processes only RBV-deliver values that pass the verification, along with the mentioned properties of reliable broadcast [163, 23] (Section 2.2.6.1).

As such, Algorithm 13 presents one major modification compared with SPURT's Π_{DBDH} protocol, in that instead of relying on the leader to share the same value to all processes, we

require all processes to reliably broadcast their inputs. This modification allows the random beacon that we propose in Section 6.2.3 to use one bit to reference each input (one per process), while ensuring all honest processes store locally the same value without the need for a leader broadcasting a digest of the secret shared as input.

Notice that processes verify all messages as soon as they are received and discard all messages that do not pass the verification. We outline in Algorithm 14 the straw man approach, omitting immediate optimizations, such as not verifying messages that were already verified, or only sharing the hash of the message in all broadcast except for the INITIAL message. We refer to previous work for more details on these optimizations [23, 28].

The verification shown in `Pinakion.verify` reuses properties of error-correcting code already used in SPURT [31]. We however restate the properties of the verification step, particularly that sharing a secret s using a degree t_ℓ polynomial among n processes is equivalent to encoding the message $(x, a_1, a_2, \dots, a_{t_\ell})$ using a $[n, t_\ell + 1, n - t_\ell]$ Reed-Solomon code C [202, 203], where a $[n, k, d]$ linear error correcting code over \mathbb{Z}_q of length n , minimum distance d and dimension k . Also, we define C^\perp as the dual code of C i.e., C^\perp consists of vectors $x^\perp \in \mathbb{Z}_q^n$ such that for all $x \in C$, $x \cdot x^\perp = 0$ where \cdot is the inner product operation. The call to `Pinakion.verify` uses the result of Lemma 6.1 on linear error correcting code, proved by Cascudo et al. [143].

Lemma 6.1. If $x \in \mathbb{Z}_q^n \setminus C$, and y^\perp is chosen uniformly at random from C^\perp , then the probability that $x \cdot y^\perp = 1$ is exactly $1/q$.

Finally, once process p_j RBV-delivers the values $(\mathbf{v}_d, \mathbf{c}_d)$, it decrypts its secret share and broadcasts it in line 20. Then, it waits until it delivers at least another t_ℓ valid decrypted secret shares to reconstruct the secret using Lagrange interpolation in line 24, finalizing the reconstruction of the secret. All received decrypted shares have to first pass a simpler check than the verification at the RBV-broadcast, represented in the call to `Pinakion.check`($h_0, v_{j,d}, s_{j,d}, g_1$) in line 22, which consists of checking whether $e(a, b) = e(c, d)$. Honest processes construct h_0^s in the call to `Pinakion.interpolate` using Lagrange interpolation:

$$\prod_{k \in T} (s_{k,d})^{\mu_k} = \prod_{k \in T} h_0^{\mu_k \cdot p(k)} = h^{p(0)} \quad (6.6)$$

where T is the set of processes from which p_i received valid decrypted shares, $|T| > t_\ell$, and $\mu_k = \prod_{j \neq k} \frac{j}{j-k}$ are the Lagrange coefficients [31].

6.2.3 The (unoptimized) Kleroterion protocol

In this section, we detail the Kleroterion protocol shown in Algorithm 16. As we show in Figure 6.1, the Kleroterion protocol generates a random output by first running n executions of the Pinakion protocol, one per process. However, instead of having each Pinakion execution terminate independently, we execute an instance of consensus in order to have processes decide on $t_\ell + 1$ inputs before reconstructing them, that is, before processes know the exact value associated with each execution of Pinakion. After deciding on exactly $t_\ell + 1$ secrets, honest processes reconstruct and then aggregate these $t_\ell + 1$ secrets to generate the random output.

Algorithm 14 RBV-broadcast

```

1: State:
2:    $g_0, h_0 \in \mathbb{G}_0$ ;  $g_1, h_1 \in \mathbb{G}_1$ , uniformly random and independent generators
3:    $sk_i \in \mathbb{Z}_q$  secret key of  $p_i$ 
4:    $pk_j = h_0^{sk_j}$  public key of  $p_j$ , for  $j \in [n]$ 

5: RBV-broadcast( $\{\mathbf{v}_d, \mathbf{c}_d\}$ ):  $\triangleright$  executed by  $p_d$ 
6:   if (Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}_d, \mathbf{c}_d\}$ )) then  $\triangleright$  verification
7:     broadcast(INITIAL,  $\{\mathbf{v}_d, \mathbf{c}_d\}$ )  $\triangleright$  broadcast to all
8:   Upon receiving a message (INITIAL,  $\{\mathbf{v}_d, \mathbf{c}_d\}$ ) from  $p_d$  do
9:     if (Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}_d, \mathbf{c}_d\}$ )) then  $\triangleright$  verification
10:      broadcast(ECHO,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ )  $\triangleright$  echo to all
11:   Upon receiving  $n - t_\ell$  distinct (ECHO,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ ) and not having sent any READY do
12:     if (Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}_d, \mathbf{c}_d\}$ )) then  $\triangleright$  verification
13:       broadcast(READY,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ )  $\triangleright$  send READY to all
14:   Upon receiving  $t_\ell + 1$  distinct (READY,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ ) and not having sent any READY do
15:     if (Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}_d, \mathbf{c}_d\}$ )) then  $\triangleright$  verification
16:       broadcast(READY,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ )  $\triangleright$  send READY to all
17:   Upon receiving  $n - t_\ell$  distinct (READY,  $\{\mathbf{v}_d, \mathbf{c}_d\}, j$ ) and not having RBV-delivered any message do
18:     if (Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}_d, \mathbf{c}_d\}$ )) then  $\triangleright$  verification
19:       return( $\{\mathbf{v}_d, \mathbf{c}_d\}$ )  $\triangleright$  deliver and send READY to all

20: Pinakion.verify( $g_1, \{pk\}_{j=1}^n, \{\mathbf{v}, \mathbf{c}\}$ ):
21:    $\mathbf{x}^\perp \xleftarrow{\$} \mathbf{C}^\perp$ 
22:   if ( $(\prod_{j \in [n]} v_j^{x_j^\perp} \neq 1_{\mathbb{G}_1})$ ) then return False
23:   for  $j \in [n]$  do
24:     if (not Pinakion.check( $pk_j, v_j, c_j, g_1$ )) then return False
25:   return True

```

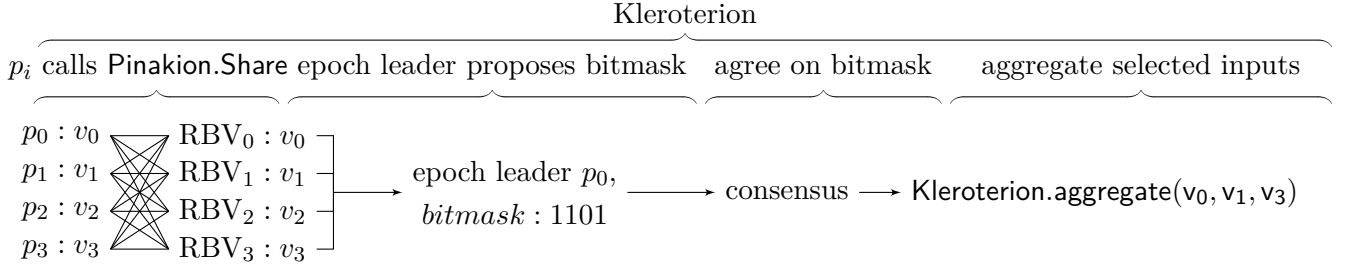


Figure 6.1: Kleroterion execution example with $n = 4$ processes. First, each process p_i selects their input value v_i (we omit secret shares and reconstruction for simplicity), which they share with everyone executing their respective instance of RBV-broadcast as part of their respective call to Pinakion.Share. Then, processes execute one leader-based consensus protocol that proceeds in epochs (HotStuff), in which the leader proposes a bitmask of n bits, with Hamming weight $t_\ell + 1$. Upon deciding on a bitmask, processes reconstruct and aggregate the $t_\ell + 1$ input secrets whose associated bit of the bitmask decided to 1.

For the consensus protocol, we use the variant of HotStuff proposed by SPURT [31, 125], for nearly-simultaneous decision, i.e. all processes learn the decision within two message delays. This protocol proceeds in epochs, with a rotating leader per epoch that proposes a value to decide, as we show in Algorithm 15. Thus, this variant differs from HotStuff only in that processes broadcast their signed PREPARE, PRECOMMIT and COMMIT messages, instead of sending them to the leader. As a result, this variant preserves the safety and liveness properties of HotStuff, as well as its responsiveness property [125] (i.e. outputs are generated at the real network latency and not necessarily at Δ).

However, contrary to SPURT, we do not require the leader of an epoch to propose to the rest a digest of his proposed secrets. Instead, leaders propose to decide on a bitmask of n bits, in which the i -th bit is associated with the secret shared by process p_i in a call of Pinakion.share. The bits that are set to 1 are secrets that will be used for the aggregation, while those set to 0 are not to be used for the aggregation. Honest processes only contribute to consensus in epochs whose proposed bitmask contains exactly $t_\ell + 1$ bits set to 1, so that exactly $t_\ell + 1$ shared secrets are aggregated into the final output. We refer to the number of bits set to 1 of a bitmask as its *Hamming weight*.

As such, the leader for this round starts participating in an epoch of the consensus protocol only if it first RBV-delivers $t_\ell + 1$ values from the n previous RBV-broadcast executed. Then, it proposes a bitmask in which all associated bits to these $t_\ell + 1$ values are set to 1, and the rest to 0. The rest of the processes execute then an exact copy of the SPURT variant of HotStuff, in which they only participate in that epoch if the Hamming weight of the proposed bitmask is $t_\ell + 1$ and they RBV-delivered the $t_\ell + 1$ associated secrets.

Once an honest process decides on a bitmask, it decrypts its share of each of the $t_\ell + 1$ decided secrets (line 20) and invokes Kleroterion.Reconstruct(*decided_secrets*, *decrypted_shares*) with its decrypted shares and the decided vectors in order to reconstruct each of the secrets and aggregate them into one final random output. We illustrate the decision of the secrets with

Algorithm 15 Steady state of a modified HotStuff [125] protocol presented by Das et al. [31] that does not use threshold signatures and has a bit complexity $\mathcal{O}(|\zeta|n^2)$ per decision ζ .

Let r be the current epoch and L be its leader. Also, let $ht - 1$ be the latest finalized iteration of the protocol.

- **Propose.** L proposes a value ζ to be finalized at height ht by sending $\langle \text{PROPOSE}, \zeta, r, ht, X \rangle$ message to all the processes. X is the view change certificate (if any) that validates that the proposal is safe.
 - **Prepare.** Each process p_j , upon receiving the proposal, checks whether the proposal is consistent with HotStuff specifications using X , and $P(\zeta)$ is true for an external predicate $P(\mathfrak{u})$. If both checks pass, process p_j sends $\langle \text{PREPARE}, \zeta, r, ht \rangle$ to all processes.
 - **Pre-Commit.** Upon receiving $2t_\ell + 1$ PREPARE messages for the proposal ζ at height ht and epoch r , process p_j sends $\langle \text{PRECOMMIT}, \zeta, r, ht \rangle$ message to every process.
 - **Commit.** Upon receiving $2t + 1$ PRECOMMIT messages for the proposal ζ at height ht and epoch r , p_j broadcasts $\langle \text{COMMIT}, \zeta, r, ht \rangle$.
 - Each process outputs ζ upon receiving $2t + 1$ COMMIT messages corresponding to ζ .
-

which to compute the random output in Algorithm 16, which integrates thus the share and verification steps of the Pinakion protocol into Kleroterion.

Setup. The Kleroterion protocol's setup phase thus consists of both SPURT's consensus protocol's and Pinakion's setup phases. That is, the setup phase of Kleroterion takes part with the creation of the keys of each process at the beginning of the Kleroterion protocol, stored in a PKI (which is not used later), along with the aforementioned CRS setup. These keys and the rest of values can be reused in all iterations of the random beacon, with the exception of the randomly chosen polynomial coefficients, which must vary in each execution [31].

Share and decide. The call to `Pinakion.Share` in line 6 creates a random input per instance of Pinakion, and RBV-broadcasts the secret shares and commitments of the random input. However, in this case each process also computes and shares their `dleq` proofs $\pi_{i,j} = \text{dleq.Prove}(g_1, v_{i,j}, pk_j, c_{i,j}, p_i(j))$ in the call to `Pinakion.Share`, where $p_i(j)$ is the share of p_i 's secret for process p_j . Process p_i then RBV-broadcasts $\{\mathbf{v}_i, \mathbf{c}_i, \boldsymbol{\pi}_i\}$, and each process p_j verifies the `dleq` proof by calling `dleq.Verify`($\pi_{i,j}, pk_j, c_{i,j}, g_1, v_{i,j}$) when delivering messages from p_i , instead of calling `Pinakion.check`, as this call becomes redundant with `dleq.Verify` [31]. This guarantees that the secrets of honest processes are independent of all other secrets by the knowledge soundness property.

Once process p_i RBV-delivers $t_\ell + 1$ proposals (line 30), and if p_i is the leader of this epoch, then p_i starts the respective consensus with the intention to decide 1 on such proposals (line 16), selecting the secrets with which to compute the final random output. If a process p_i is not the leader of this epoch but p_i receives a bitmask proposed from the leader in this epoch, then p_i checks whether p_i RBV-delivered all the proposed secrets associated to the bitmask (line 17). If

it does, then it contributes to reaching consensus in this epoch. Otherwise, it does not respond in this epoch.

Once the consensus protocol terminates deciding a bitmask, process p_i waits until it RBV-delivers the proposals associated with each bit set to 1 in line 24. This can happen if consensus terminated without the participation of p_i . Following, p_i decrypts its corresponding secret share of each decided secret in line 27. Finally, p_i calls `Kleroterion.Reconstruct` (line 28) with its decrypted shares and list of decided proposals, and returns the random output resulted from the call to `Kleroterion.Aggregate` (line 29) which aggregates the decided reconstructed secrets. Honest processes can then return the computed final random output of this iteration of the random beacon, and start the next iteration.

Algorithm 16 `Kleroterion.Decide` for process p_i

```

1: State:
2:    $g_0, h_0 \in \mathbb{G}_0$ ;  $g_1, h_1 \in \mathbb{G}_1$ , uniformly random and independent generators
3:    $sk_i \in \mathbb{Z}_q$  secret key of  $p_i$ 
4:    $pk_j = h_0^{sk_j}$  public key of  $p_j$ , for  $j \in [n]$ 

5:  $s \xleftarrow{\$} \mathbb{Z}_q$ 
6: Pinakion.Share(s) ▷ create and RBV-broadcast secret
7: repeat:
8:   if (consensus.epoch_leader() =  $p_i$ ) then
9:     if ( $|proposals_i| \geq t_\ell + 1$ ) then
10:       $bitmask \leftarrow \{\}$ 
11:       $k = 0$ 
12:      while  $|bitmask| < n$  do
13:        if ( $k \in proposals_i.keys()$  and  $\text{Hamming\_weight}(bitmask) < t_\ell + 1$ ) then
14:           $bitmask[k] \leftarrow 1$ 
15:           $k \leftarrow k + 1$ 
16:         $\langle \text{PROPOSE}, bitmask, consensus.epoch, ht, X \rangle$ 
17:      else if (received  $\langle \text{PROPOSE}, bitmask, r, ht, X \rangle$  and  $consensus.epoch = r$ ) then
18:        if ( $j \in proposals_i.keys() \forall j$  s.t.  $bitmask[j] = 1$  and  $\sum_j bitmask[j] = t_\ell + 1$ ) then
19:          contribute to consensus in epoch  $r$ 
20:      until consensus.finished() ▷  $t_\ell + 1$  secrets chosen
21:       $decrypted\_shares \leftarrow \{\}$ 
22:       $decided\_secrets \leftarrow \{\}$ 
23:      for each  $j$  s.t.  $consensus.decision[j] = 1$  do ▷ for each decided secret
24:         $\text{wait\_until}(proposals_i[j] \neq \perp)$ 
25:         $c_j \leftarrow proposals_i[j]$ 
26:         $decided\_secrets[j] \leftarrow proposals_i[j]$ 
27:         $decrypted\_shares[j] \leftarrow c_{i,j}^{1/sk_i}$  ▷ decrypt  $p_i$ 's share
28:       $decisions \leftarrow \text{Kleroterion.Reconstruct}(decided\_secrets, decrypted\_shares)$  ▷ reconstruct
29:      return Kleroterion.Aggregate(decisions) ▷ aggregate into random output

30: when  $p_i$  RBV-delivers  $\{v_j, c_j, \pi_j\}$  do:
31:    $proposals_i[j] \leftarrow c_j$ 

```

Reconstruct and aggregate. After terminating consensus, the reconstruction phase starts

to reconstruct the corresponding $t_\ell + 1$ decided secrets. We show in Algorithm 17 the call to `Kleroterion.Reconstruct`. First, p_i broadcasts its decrypted shares of the decided secrets in line 7. Upon receiving a list of decided secrets and decrypted shares from p_j , p_i verifies them in the call to `Kleroterion.Verify` (line 11). `Kleroterion.Verify` checks first that the local and delivered list of decided secrets is the same (line 17) and that each decrypted share passes the `Pinakion.check` (line 20) previously described. If the received message verifies, then its decrypted shares are added to the list of decrypted shares (line 12), which is used to reconstruct all the decided secrets with the call to `Kleroterion.MultipleRecon` (line 14) once the list contains at least $t_\ell + 1$ decrypted shares for each secret. The call to `Kleroterion.MultipleRecon` gathers all the decrypted shares (line 27) for each secret and reconstructs them calling `Pinakion.interpolate` in line 28.

Algorithm 17 `Kleroterion.Reconstruct` for process p_i

```

1: State:
2:    $g_0, h_0 \in \mathbb{G}_0$ ;  $g_1, h_1 \in \mathbb{G}_1$ , uniformly random and independent generators
3:    $sk_i \in \mathbb{Z}_q$  secret key of  $p_i$ 
4:    $pk_j = h_0^{sk_j}$  public key of  $p_j$ , for  $j \in [n]$ 
5:   decided_secretsi, list of decided encrypted secret shares of  $p_i$ 
6:   decrypted_sharesi, list of decided decrypted shares of  $p_i$ 

7: broadcast(decrypted_shares)
8: list_decrypted_shares  $\leftarrow \{\}$ 
9: random_outputs  $\leftarrow \perp$ 
10: Upon receiving {decided_secretsj, decrypted_sharesj} from  $p_j$ :
11:   if (Kleroterion.verify(decided_secretsj, decrypted_sharesj,  $h_0, g_1$ ) and random_outputs =  $\perp$ ) then
12:     list_decrypted_shares[ $j$ ]  $\leftarrow$  decrypted_sharesj
13:     if ( $\text{size}(\text{list\_decrypted\_shares}) > t_\ell$ ) then  $\triangleright$  enough to reconstruct
14:       random_outputs  $\leftarrow$  Kleroterion.MultipleRecon( $h_1, h_0, \text{list\_decrypted\_shares}$ )
15:       if (random_outputs  $\neq \perp$ ) then return random_outputs

16: Kleroterion.verify(decided_secretsj, decrypted_sharesj,  $h_0, g_1$ ):
17:   if (decided_secretsj  $\neq$  decided_secretsi) then return False  $\triangleright$  different secrets
18:   for each  $s_{j,k}$  in decrypted_sharesj do
19:      $v_{j,k} \leftarrow \text{decided\_secrets}[j].\mathbf{v}_j[k]$ 
20:     if (not Pinakion.check( $h_0, v_{j,k}, s_{j,k}, g_1$ )) then return False
21:   return True

22: Kleroterion.MultipleRecon( $h_1, h_0, \text{list\_decrypted\_shares}, \text{decided\_secrets}$ ):
23:   random_outputs  $\leftarrow \{\}$ 
24:   for  $k$  in decided_secrets.keys() do
25:     aux  $\leftarrow \{\}$ 
26:     for  $j$  in list_decrypted_shares.keys() do  $\triangleright$  for each secret
27:       aux[ $j$ ]  $\leftarrow$  list_decrypted_shares[ $j$ ][ $k$ ]  $\triangleright$  gather all decrypted shares
28:     random_outputs[ $k$ ]  $\leftarrow$  Pinakion.interpolate( $h_0, h_1, \text{aux}$ )  $\triangleright$  reconstruct
29:   return random_outputs

```

6.2.3.1 Proofs of correctness

The Pinakion protocol shown in Section 6.2.2 is an instantiation of SPURT's Π_{DBDH} in which the broadcast primitive is replaced by our RBV-broadcast protocol. Similarly, our RBV-Broadcast is almost identical to the implementation by Bracha et al. [162], with the only modification that processes only deliver a value if it passes the verification step from Π_{DBDH} . The proofs of Pinakion solving the PVSS problem and of RBV-broadcast solving the reliable broadcast problem are thus equivalent to the proofs of these two previous works. Liveness of both reliable broadcast and PVSS are subject to the dealer (or source) being honest, meaning that replacing the broadcast primitive by our RBV-broadcast preserves liveness. On the safety side, RBV-broadcast further enhances safety thanks to the rest of the properties of reliable broadcast compared with a general broadcast primitive.

The same occurs with Kleroterion. The only additional difference between the above-shown unoptimized Kleroterion and SPURT is that while in SPURT the leader of the epoch shares a digest of the $t_\ell + 1$ selected shares, in Kleroterion the leader shares a bitmask with Hamming weight $t_\ell + 1$, that associates each bit to a particular secret RBV-broadcast by each process. We thus need to prove only that if an honest process decides a bitmask in an epoch r , then all honest processes eventually reconstruct and output the same final random output from that epoch. We show this in Lemma 6.2. But before that, we show first that Kleroterion solves SBC (Definition 2.2.4).

Theorem 6.1. Kleroterion solves SBC.

Proof. SBC-Agreement derives from RB-Unicity, RB-Receive and the agreement property of the HotStuff protocol. That is, by the agreement property of HotStuff all processes agree on the bitmask. By RB-Unicity and RB-Receive all processes agree on the values that the bits set to 1 of the bitmask refer to.

SBC-Termination derives from RB-Send and the termination property of HotStuff. By RB-Send all processes eventually deliver at least the $n - t_\ell$ values shared by honest processes. If honest processes keep trying different bitmasks when they are the leader of an epoch, eventually there is an epoch after GST whose leader is honest, and proposes RBV-delivered values that have been RBV-delivered by all other honest processes. Processes can terminate in that epoch.

SBC-Validity and SBC-Nontriviality are trivial as the decision is a bitmask of proposals. \square

Lemma 6.2. Suppose an honest process decides a bitmask `bitmask` in epoch r , let I be the set of decrypted polynomials referenced by the bits of `bitmask` set to 1, and let \hat{p} be the aggregated polynomial $\hat{p}(\cdot) = \sum_{i \in I} p_i(\cdot)$. Then, every honest process outputs $e(h_0^a, h^1)$ where $a \in \mathbb{Z}_q$, for $a = \hat{p}(0)$.

Proof. For an honest process to terminate in epoch r , at least $t_\ell + 1$ honest processes participated in the consensus protocol in r . By construction, honest processes only participate in an epoch of the consensus protocol if they have first RBV-delivered all the values referenced by the bitmask, and if the Hamming weight of the bitmask is exactly $t_\ell + 1$. Also by construction, an honest process only RBV-delivers a share of secrets if it passes the verification step.

By Theorem 6.1, all honest processes will eventually decide on the same bitmask and all honest processes will eventually RBV-deliver all the values associated with each bit of the bitmask set to 1.

As a result, except with negligible probability, the degree of $\hat{p}(\cdot)$ is at most t_ℓ . This is because any polynomial of degree greater than t_ℓ passes the verification step of RBV-broadcast with probability only $1/q$; hence, the probability that it passes the check at $t_\ell + 1$ honest processes is $\binom{2t_\ell+1}{t_\ell+1} \frac{1}{q^{t_\ell+1}} \leq \frac{1}{q}$, which is negligible [31].

Every honest process p_j that participated in consensus holds the witness $h_0^{sk_j \cdot \hat{p}(j)}$ if the `dleq.Verify` check passed [31]. Thus, at least $t_\ell + 1$ honest processes will broadcast their decrypted shares during the reconstruction such that anyone can verify them.

Finally, after performing Lagrange interpolation over the exponent with these $t_\ell + 1$ decrypted shares (which pass the equality check [31]), honest processes can recover the beacon output $e(h_0^{\hat{p}(0)}, h_1)$. \square

Lemma 6.2 is our analogous result of SPURT's Lemma 2 [31]. We refer to SPURT for the rest of the proofs of correctness, as they are identical. We analyze in Section 6.2.3.2 the complexities of an unoptimized Kleroterion, and provide optimizations that reduce the complexity of Kleroterion to $\mathcal{O}(n^2)$ per decision, with the advantage of scattering the shared bits throughout all pairwise channels of the network (instead of channels to and from the leader, as is the case for non-democratic protocols like SPURT).

6.2.3.2 Complexities of naive Kleroterion

Table 6.1 shows the time, computational, message and bit complexities of unoptimized implementations Kleroterion, Pinakion, and RBV-broadcast protocols. The RBV-broadcast protocol requires each n processes to broadcast to all n processes the n encrypted secret shares, meaning a message of size $\mathcal{O}(\lambda \cdot n)$, which needs to be verified for each of its elements. Pinakion provides the same complexities, as the bottleneck of Pinakion is the RBV-broadcast. The Kleroterion protocol runs n concurrent executions of Pinakion, increasing message and bit complexities by a linear factor compared to Pinakion's complexities. The HotStuff protocol's message complexity is $\mathcal{O}(n^2)$. However, SPURT's variant requires all processes to broadcast messages in order to satisfy nearly-simultaneous decision, instead of sending messages to the leader for aggregation. As a result, this variant has message complexity $\mathcal{O}(n^3)$, and since the proposal is a bitmask of n bits, the resulting bit complexity is $\mathcal{O}(n^4)$. The bottleneck of naive Kleroterion is thus the n instances of Pinakion, resulting in a message complexity of $\mathcal{O}(n^3)$ and a bit complexity of $\mathcal{O}(\lambda n^4)$.

6.2.4 Optimizations and observations

The straw man implementation we showed in Section 6.2.3, while correct, is suboptimal compared with SPURT. We detail here the optimizations that decrease the bit complexity of Kleroterion from $\mathcal{O}(\lambda n^4)$ to $\mathcal{O}(\lambda n^2)$ per decision. Furthermore, we observe in this section that Kleroterion is in fact better suited to be implemented in WANs (e.g. the Internet) than other

Protocol	Complexities		
	Time	Message	Bit
Naive RBV-broadcast	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^3)$
Naive Pinakion	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$	$\mathcal{O}(\lambda n^3)$
Naive Consensus	$\mathcal{O}(t_\ell)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^4)$
Naive Kleroterion	$\mathcal{O}(t_\ell)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\lambda n^4)$

Table 6.1: Time, computational, message and bit complexities of naive RBV-broadcast, Pinakion and Kleroterion.

PVSS-based random beacons with the same bit complexity, as the number of bits processes send per each channel per decision is independent of the number of participants. In contrast, SPURT saturates channels to the leader sending $\mathcal{O}(\lambda n)$ bits through them. We also observe the advantages of decoupling the consensus proposal from the actual shared values by proposing a bitmask, in that the associated bit enables batching.

6.2.4.1 From quartic to quadratic bit complexity

In this section, we show how to reduce Kleroterion's bit complexity from quartic to quadratic. For this purpose, we present three optimizations.

Bitmask digest. The first optimization refers to the HotStuff consensus protocol. In order to reduce the bit complexity, instead of having processes decide on a bitmask of size $\mathcal{O}(n)$ bits, the leader broadcasts the bitmask initially and then proposes a digest of the bitmask, of size $\mathcal{O}(\lambda)$ bits. This results in a bit complexity of $\mathcal{O}(\lambda n^3)$ for the consensus protocol, since processes do not need to broadcast a message of size $\mathcal{O}(n)$ but instead only the digest of size $\mathcal{O}(\lambda)$. Honest processes can still satisfy correctness of the protocol by contributing to reaching consensus only if they received the bitmask that corresponds to the proposed digest. This is an advantage of having a leader that proposes the subset of inputs during consensus.

Aggregated inputs. In Algorithm 13, each dealer p_d shares a list of secrets \mathbf{c}_d and commitments \mathbf{v}_d , with the addition of the vector of dleq proofs $\boldsymbol{\pi}_d$ for Kleroterion. SPURT requires all processes to send these vectors to a leader so that it aggregates them, but this saturates the number of bits sent through the channels to the leader. Furthermore, Kleroterion cannot aggregate commitments and secrets from different processes, since it does not have a leader that will receive all these vectors from all processes. As such, instead, Kleroterion requests processes to generate n secrets per process, and aggregate them locally before sharing, such that process p_i generates n secrets $\{s_{i,k}\}_{k \in [n]}$ and then generates his shares $\{c_{i,j,k}\}_{k \in [n], j \in [n]}$, commitments $\{v_{i,j,k}\}_{k \in [n], j \in [n]}$, and dleq proofs $\{\pi_{i,j,k}\}_{k \in [n], j \in [n]}$. Then, p_i aggregates the shares and commitments by multiplying the shares and commitments encrypted with the public key of the same recipient, for example for process p_0 the aggregation of the shares results in $\hat{c}_{i,0} = \prod_{k=0}^{n-1} c_{i,0,k}$, and those of the commitment results in $\hat{v}_{i,0} = \prod_{k=0}^{n-1} v_{i,0,k}$. Following previous terminology [31], we use $\bar{\mathbf{c}}_{i,j}$ to refer to all the shares encrypted by process p_i with the public key of process p_j , i.e. $\bar{\mathbf{c}}_{i,j} = \{c_{i,j,k}\}_{k=0}^{n-1}$, and the same for $\bar{\mathbf{v}}_{i,j}$ and $\bar{\boldsymbol{\pi}}_{i,j}$. We show the updated RBV-broadcast

protocol with this optimization in Algorithm 18. Figure 6.2 shows this optimization for process p_d 's secret shares $c_{d,i,k}$ (the same occurs for the commitments $v_{d,i,k}$). Each column contains the secret shares of a different new input secret (i.e. $s_{d,i}$, shown above the matrix). Each row contains the secret share from each different secret encrypted with the public key of the same recipient. We use $\bar{c}_{d,i}$ to refer then to the secret shares from each secret with the same recipient p_i (i.e. the i -th row of the matrix). Then, $\hat{c}_{d,i}$ is the result of multiplying each value in the i -th row (i.e. aggregated secret shares by recipient). Finally, $\hat{\mathbf{c}}_d$ is the list of all the aggregated secret shares by recipients.

In Figure 6.3, we illustrate the beginning of Algorithm 18, to showcase what p_d sends in the initial message to each process. Note that p_d sends to each process a message of size $\mathcal{O}(n)$ bits which contains $\Omega(n)$ secret inputs, thanks to the aggregation performed in Figure 6.2.

Algorithm 18 Optimized RBV-broadcast

```

1: State:
2:    $g_0, h_0 \in \mathbb{G}_0$ ;  $g_1, h_1 \in \mathbb{G}_1$ , uniformly random and independent generators
3:    $sk_i \in \mathbb{Z}_q$  secret key of  $p_i$ 
4:    $pk_j = h_0^{sk_j}$  public key of  $p_j$ , for  $j \in [n]$ 

5: RBV-broadcast( $\{\{c_{d,j,k}\}, \{v_{d,j,k}\}, \{\pi_{d,j,k}\}\}$ ):
6:   for  $j \in [n]$  do ▷ executed by  $p_d$ 
7:      $\hat{c}_{d,j} \leftarrow \prod_{k \in [n]} c_{d,j,k}$ 
8:      $\hat{v}_{d,j} \leftarrow \prod_{k \in [n]} v_{d,j,k}$ 
9:   for  $j \in [n]$  do
10:    if (Pinakion.verify( $g_1, \{pk_k\}, \{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d\}$ )) then ▷ verification
11:      send(INITIAL,  $\{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, \bar{\mathbf{c}}_{d,j}, \bar{\mathbf{v}}_{d,j}, \bar{\pi}_{d,j}\}$ ) ▷ dealer broadcasts to all
12:  Upon receiving a message (INITIAL,  $\{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, \bar{\mathbf{c}}_{d,i}, \bar{\mathbf{v}}_{d,i}, \bar{\pi}_{d,i}\}$ ) from  $p_d$  do
13:    if (Pinakion.verify-opt( $g_1, \{pk\}_{j=1}^n, \{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, \bar{\mathbf{c}}_{d,i}, \bar{\mathbf{v}}_{d,i}, \bar{\pi}_{d,i}\}$ )) then ▷ verification
14:      broadcast(ECHO,  $\{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, i\}$ ) ▷ echo to all
15:  Upon receiving  $n - t_\ell$  distinct (ECHO,  $\{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, i\}$ ) and not having sent any READY do
16:    broadcast(READY,  $\{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, j\}$ ) ▷ send READY to all

17: Pinakion.verify-opt( $g_1, \{pk\}_{j=1}^n, \{\hat{\mathbf{v}}_d, \hat{\mathbf{c}}_d, \bar{\mathbf{c}}_{d,i}, \bar{\mathbf{v}}_{d,i}, \bar{\pi}_{d,i}\}$ ):
18:    $\mathbf{x}^\perp \xleftarrow{\$} \mathbf{C}^\perp$ 
19:   if ( $(\prod_{j \in [n]} \hat{v}_{d,j}^{x_j^\perp} \neq 1_{\mathbb{G}_1})$ ) then return False
20:   for  $j \in [n]$  do
21:     if (not dleq.Verify( $\bar{\pi}_{d,i,j}, pk_i, \bar{c}_{d,i,j}, g_1, \bar{v}_{d,i,j}$ )) then return False
22:   if ( $\hat{v}_{d,i} \neq \prod_{j \in [n]} \bar{v}_{d,i,j}$  or  $\hat{c}_{d,i} \neq \prod_{j \in [n]} \bar{c}_{d,i,j}$ ) then return False
23:   return True

```

One can note that it is possible that Byzantine processes broadcast shares that only pass the local verification of up to $t_\ell + 1$ honest processes. Nevertheless, this is not a problem because either the consensus protocol terminates with a decided bitmask that contains some secrets shared by Byzantine processes, which would mean that at least $t_\ell + 1$ honest processes can decrypt their share and reconstruct all $n \cdot (t_\ell + 1)$ shared secrets associated to the decided bitmask, or instead eventually there is an epoch after GST whose leader is an honest process

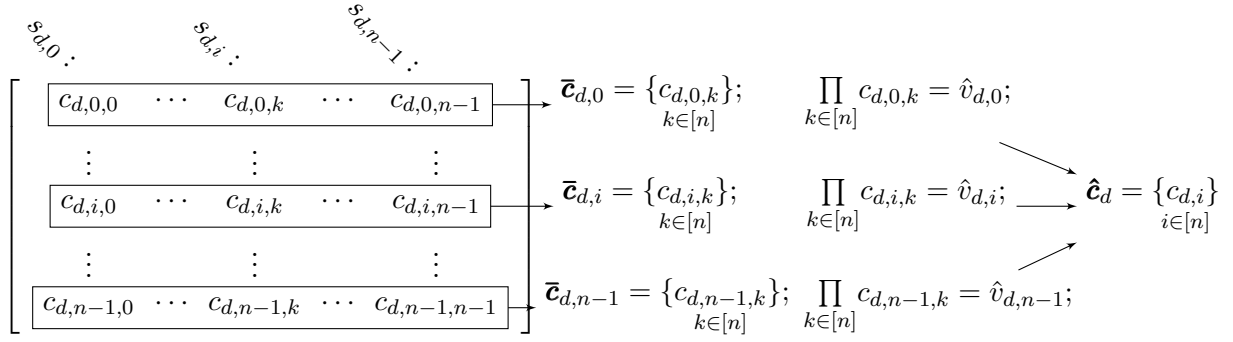


Figure 6.2: Example of the aggregation of the secret shares $\{c_{d,i,k}\}$ generated from n secrets $\{s_{d,i}\}_{i \in [n]}$ locally by each process p_d (the same aggregation occurs for commitments instead of secret shares).

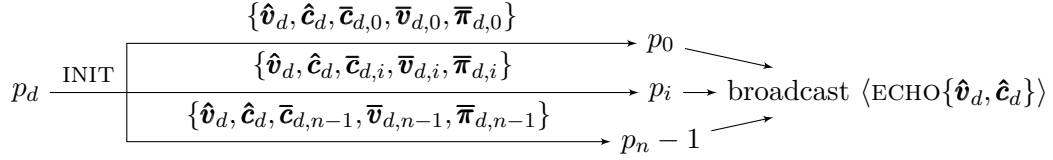


Figure 6.3: Example of the beginning of the optimized RBV-broadcast protocol for process p_d .

and this honest process proposes a bitmask containing $t_\ell + 1$ secrets RBV-delivered by all honest processes, and in this case the consensus protocol terminates in this epoch. That is, thanks to the consensus protocol it is impossible for Byzantine processes to prevent termination, or to decide a result that honest process cannot reconstruct, for $t_\ell < n/3$.

This optimization reduces the bit complexity of Kleroterion to $\mathcal{O}(\lambda n^3)$ per decision, since one iteration generates n random outputs. That is, the normalized complexity of Kleroterion is $\mathcal{O}(\lambda n^3)$ due to this optimization (see Section 2.2.9).

The optimized RBV-Broadcast showed in Algorithm 18 can be combined with the Kleroterion.Decide and Kleroterion.Reconstruct functions we showed in Algorithms 16 and 17, respectively, with the modifications that honest processes verify the correct aggregation of the received decided secrets and decrypted shares during the reconstruction, as well as replacing Pinakion.check with dleq.verify, as we did for Algorithm 18 because of their redundancy [31]. That is, within the call to Kleroterion.verify of line 11 of Algorithm 17, honest processes also check $\hat{v}_{d,k} = \prod_{j \in [n]} \bar{v}_{d,k,j}$ and $\hat{c}_{d,k} = \prod_{j \in [n]} \bar{c}_{d,k,j}$ to verify the received shares from process p_k . We now show how to reduce the bit complexity of Kleroterion to make it quadratic per decision.

Amortized complexity. The final observation that results in the quadratic bit (and message) complexity of Kleroterion was already noted by previous work [125, 31] in what we define in Section 2.2.9 as amortized complexity. Since a linear factor of the complexity derives from the possibility that $\mathcal{O}(t_\ell)$ leaders are faulty after GST, this means that, in the presence of a static adversary, for n consecutive executions of Kleroterion, there will be $\Omega(n)$ outputs (even $\Omega(n^2)$ with the aforementioned aggregated inputs optimization). As a result, the amortized and normalized bit complexity of Kleroterion is $\mathcal{O}(\lambda n^2)$, same as the amortized and normalized

bit complexity of SPURT. We justify in the following the advantages of a democratic protocol compared with a non-democratic one.

6.2.4.2 Distributing bits and computation

To the best of our knowledge, previous PVSS-based random beacons implement a protocol that is not democratic, i.e. where all inputs are routed through the leader of the epoch. In contrast, democratic protocols like Kleroterion distribute inputs across all processes, although there is still a leader to select the subset of all inputs using a bitmask to reference inputs. Thus, contrary to previous random beacons based on PVSS [31], the leader of a Kleroterion epoch only has to share the bitmask, of size $\mathcal{O}(n)$ bits (and the digest of size $\mathcal{O}(\lambda)$ bits). We summarize the distribution of computation and communication of Kleroterion compared to SPURT in Table 6.2. We omit the reconstruction phase as it is equivalent in both SPURT and Kleroterion, requiring an exchange of $\mathcal{O}(\lambda n^2)$ bits ($\mathcal{O}(\lambda n)$ per channel) and a computation of $\mathcal{O}(n)$ per process.

Table 6.2: Comparison of normalized and amortized bit complexity per each pairwise channel of the network, and computational complexity per process, in SPURT [31] and Kleroterion, after GST.

	Phase	Computation		Bits per channel	
		Leader	Non-leaders	Leader	Non-leaders
SPURT	Commitment	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(\lambda)$
	Aggregation	$\mathcal{O}(n^2)$	-	-	-
	Agreement	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda n)$	$\mathcal{O}(\lambda)$
Kleroterion	Commitment	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
	Aggregation	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda)$	$\mathcal{O}(\lambda)$
	Agreement	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(\lambda)$

Distributing communication. For the case of SPURT, since the leader needs to perform the aggregations of each share of secrets, the channels to the leader transfer $\mathcal{O}(\lambda n)$ bits per decision, all with the same recipient. Then, the leader computes the aggregated values, outputting also $\mathcal{O}(\lambda n)$ bits to each process. With the aforementioned optimizations, Kleroterion only sends $\mathcal{O}(\lambda)$ bits per pairwise channel per decision, and distributes the verification and computation of aggregated values, also decreasing the computation complexity, as we show later.

This distinction means that the bandwidth of the network routes from and to the leader of SPURT and other non-democratic protocols will be the bottleneck of these protocols, whereas Kleroterion exploits all pairwise channels of the network. Recent results prove that consensus protocols with comparable bit complexity but lower per route complexity perform at significantly greater throughputs than their non-democratic counterparts (Section 2.2.9). Notice also that

the $\mathcal{O}(n)$ bits sent by the leader during the agreement phase of Kleroterion enables batching, as we discuss later in Section 6.2.4.3.

Distributing computation. The democratic approach of Kleroterion also helps distributing computation between processes (Section 2.2.9). In particular, while the commitment phase requires $\mathcal{O}(n)$ exponentiations per decision (same as SPURT), the aggregation differs significantly. SPURT requires the leader to verify the PVSS shares from all processes and aggregate them. As a result, SPURT’s leader performs $\mathcal{O}(n^2)$ exponentiations per decision to verify all PVSS shares, and $\mathcal{O}(n^2)$ multiplications to compute the aggregations, while the rest of $n - 1$ non-leader processes perform no work waiting for the leader to perform these computations. In contrast, in Kleroterion each process p_i locally aggregates their n secret shares by recipient, and verify the shares from the $n - 1$ other processes for which p_i is the recipient, resulting in $\mathcal{O}(n)$ exponentiations and aggregations per decision per process, whether that is a leader or not.

Furthermore, the leader of SPURT needs to hash $\mathcal{O}(n)$ group elements to compute the digest that will be decided during the consensus protocol. In contrast, Kleroterion’s digest is just one hash of a bitmask of $\mathcal{O}(n)$ bits. The computational complexities per decision of the remaining agreement and reconstruction phases, as well as the complexities of publicly verifying outputs, are the same for Kleroterion and SPURT. This means that Kleroterion also removes the quadratic computational bottleneck at the leader, in addition to removing the bandwidth bottleneck at network channels involving the leader.

6.2.4.3 Decoupling proposals from consensus

Another advantage of our construction, that executes first RBV-broadcast and then a consensus that references these RBV-delivered values, is the possibility to make the bitmask of the consensus decision reference more information than the secret shares of that process. This does not affect correctness, as processes only decide on $t_\ell + 1$ bits that they can verify and whose associated data has been reliably broadcast, and there are $n - t_\ell > t_\ell + 1$ honest processes.

To motivate this, we present an example where this decoupling is useful is blockchains. In a blockchain application, processes can decide a superblock [23, 28] (i.e. an union of proposals) by reliably broadcasting these blocks and then deciding on a bitmask that represents the blocks to merge into a superblock, as it is the case for many democratic consensus protocols [27, 80, 81, 23, 28]. An example of this is our Basilic class of protocols used by ZLB (Chapter 5) that solves SBC (Definition 2.2.4). Such a blockchain could benefit from our RBV-broadcast protocol by implementing a random beacon in the blockchain (e.g. for committee sortition or as a source of randomness to be used by other services of the system) only at the additional cost of executing n Pinakion.Share iterations, but reusing the same iteration of consensus already existing in the blockchain.

As a result, the final complexity of Kleroterion is competitive with that of recent works, with the advantage that Kleroterion democratization exploits better the computation and bandwidth resources of the network.

6.2.5 Benefits of democratic, leader-based protocols

	Network	fault tolerance	Adaptive adversary	Liveness	Unpredictability	Bias-resistance	Bit complexity	Computational complexity	Public-verifiability complexity	Cryptographic primitives	Setup
Cachin et al. [35]	A.	1/3	✗	✓	✓	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Uniq. th-sig.	DKG
RandHerd [36]	P.	1/3 [†]	✗	✓	✓	✓	$\mathcal{O}(\lambda c^2 \log n)$ [†]	$\mathcal{O}(\lambda c^2 \log n)$	$\mathcal{O}(1)$	PVSS+CoSi	DKG
Dfinity [37]	S.	1/3	✗	✓	✓	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Uniq. th-sig.	DKG
Drand [38]	S.	1/2	✗	✓	✓	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$	Uniq. th-sig.	DKG
HERB [39]	P.	1/3	✗	✓	✓	✓	$\mathcal{O}(\lambda n^4)$ [‡]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	Partial HE	DKG
Algorand [26]	P.	1/3 [†]	✗	✓	\approx^*	✗ ^{**}	$\mathcal{O}(\lambda c n)$ [†]	$\mathcal{O}(c)$	$\mathcal{O}(1)$	VRF	CRS
Bitcoin [1]	S.	1/2	✗	✓	\approx^*	✗ ^{**}	$\mathcal{O}(\lambda n)$	very high	$\mathcal{O}(1)$	Hash func.	CRS
Ouroboros [24]	S.	1/2	✗	✓	✓	✓	$\mathcal{O}(\lambda n^4)$ [‡]	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	CRS
Scrape [143]	S.	1/2	✗	✓	✓	✓	$\mathcal{O}(\lambda n^4)$ [‡]	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS+Bcast.	CRS
Hydrand [40]	S.	1/3	✗	✓	\approx^*	✓	$\mathcal{O}(\lambda n^2 \log n)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS	CRS
RandRunner [144]	S.	1/2	✓	✓	\approx^*	✓	$\mathcal{O}(\lambda n^2)$	VDF	$\mathcal{O}(1)$	VDF	CRS
GRandPiper [145]	S.	1/2	✗	✓	\approx^*	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	q-SDH
BRandPiper [145]	S.	1/2	✓	✓	✓	✓	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	VSS	q-SDH
Nguyen et al. [204]	S.	1	✗	✗	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	FHE+VRF	–
ProofOfDelay [205]	S.	1/2	✗	✓	✓	✓	$\mathcal{O}(n)$ [§]	high	$\mathcal{O}(1)$	VDF	–
No-dealer [41]	S.	1/2	✗	✓	✓	✓	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	Shamir+HE	–
SPURT [31]	P.	1/3	✗	✓	✓	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n)$	PVSS+Pairing	CRS
Kleroterion	P.	1/3	✗	✓	✓	✓	$\mathcal{O}(\lambda n^2)$	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS+Pairing	CRS

[†] Algorand and Randherd use a randomly sampled committee of size c . This improves scalability at the cost of slightly reducing fault tolerance [31].

[‡] The complexity counting that of their broadcast channel (blockchain) [31].

[§] $\mathcal{O}(n)$ + Ethereum [141].

* The adversary can withhold inputs and try to compute output which can break unpredictability [206, 31].

** The adversary can discard undesirable beacon outputs [31].

Table 6.3: Comparison of distributed random beacons [31, 141].

We summarize the state of the art and compare it with Kleroterion in Table 6.3. Contrary to all previous random beacon protocols, Kleroterion asks each process to locally generate n inputs, and aggregate their own inputs locally, to then broadcast them. This allows Kleroterion to relieve SPURT's strong dependency on the leader's computation and communication, as seen in Table 6.2, resulting in as little as a number of bits exchanged per each pair of processes per decision that is independent of the number of processes running the protocol, except for one message sent by the leader and for the reconstruction phase.

However, Kleroterion also presents interesting observations compared to more recent leader-less protocols [79, 31, 27, 80, 81, 28]. The majority of leader-less protocols that we know of start by having processes exchange their inputs through an all-to-all broadcast, like Kleroterion. However, unlike Kleroterion and like the Basilic class (Chapter 5), these protocols then execute one binary consensus instance per input, resulting in a bit complexity of at least $\Omega(n)$ for the combined execution of all binary consensus instances (since there are $\Omega(n)$ binary executions, each exchanging at least $\Omega(1)$ bits). By contrast, Kleroterion proceeds instead by executing a

leader-based consensus to simply propose a digest of a bitmask, providing a proposal of size $\mathcal{O}(\lambda)$ bits, where $\lambda \leq n$ is the security parameter. Although it is possible to create implementations of leader-less protocols that benefit from a similar optimization [47], Kleroterion emphasizes the importance of democratizing protocols, and not necessarily removing its leader altogether.

6.3 Kleroterion⁺: tolerating colluding majorities

We show in this section how to extend Kleroterion to tolerate colluding majorities. For this reason, we now refer to an adversary that can control $t \leq t_\ell$ Byzantine processes and $t + d \leq t_s$ Byzantine and deceitful processes as a (t_ℓ, t_s) -bounded adversary.

6.3.1 Secure random beacon and accountable PVSS problems

We define the secure random beacon (SRB) problem and the accountable PVSS (APVSS) problem in order to tolerate coalitions of size greater than t_ℓ .

6.3.1.1 Secure random beacon

For cases where $t_\ell < t + d \leq t_s$, with $t_s = \lceil 2n/3 \rceil - 1$, Kleroterion⁺ satisfies accountability instead of agreement [163], along with the rest of the properties of our novel definition of the secure random beacon problem.

Definition 6.3.1 (Secure random beacon). Let \mathcal{M} be a (t_ℓ, t_s) -bounded adversary and λ be a security parameter. Let a committee N of $|N| = n$ processes execute an epoch based protocol σ which outputs an output $Z \in \mathbb{Z}_q$ per epoch. Then, σ is an $(n, \mathcal{M}, \lambda)$ -secure random beacon $((n, \mathcal{M}, \lambda)$ -SRB) protocol if it satisfies all of the following properties with probability at least $1 - \epsilon(\lambda)$:

- Agreement for all epochs where the total number of faulty processes is $t + d \leq t_\ell$:
 - **Agreement:** All honest processes agree on the same random output Z .
- The following properties for all epochs where $t + d \leq t_s$, $t \leq t_\ell$:
 - **Availability:** Every honest process eventually outputs one value Z .
 - **Verifiability:** If an honest process decides Z , then every honest process can verify it.
 - **Unpredictability:** Before at least $t_s + 1$ processes output Z , no process can predict the value of Z with probability greater than $1/q + \epsilon(\lambda)$ (i.e. randomly guessing the secret).
 - **Bias-resistance:** No process can fix some c bits of Z for any epoch with probability better than $\epsilon(c) + \epsilon(\lambda)$.

- **Accountability:** If honest processes output different values, then the set Υ of all verifiable decided outputs contains at most $|\Upsilon| \leq t_\ell + 1$ outputs, and honest processes will identify at least $t_\ell + 1$ faulty processes that are provably guilty of the disagreement.

In order to consider properties beyond the classical t_ℓ threshold, we split verifiability into two properties instead, adding the agreement property, which has been implicitly stated in some previous works. Additionally, even if faulty processes can cause a disagreement, faulty processes can neither predict nor bias the decision of each honest process, nor can they make an honest process decide less or more than one output (before finding about the disagreement and resolving it). Also, all decided outputs are publicly verifiable.

The additional property of accountability replaces agreement for the epochs where the adversary controls more than t_ℓ processes, satisfying that honest processes will identify a disagreement and the faulty processes responsible for it. It also limits the disagreement into at most $t_\ell + 1$ distinct outputs for the same epoch, a result previously shown [46] that we also show in Theorem 5.8.

6.3.1.2 Accountable PVSS

In this section, we present the accountable PVSS (APVSS) problem, the natural extension of PVSS when dealing with $t + d \leq t_s$ and $t \leq t_\ell$ faults.

Definition 6.3.2 (Accountable publicly-verifiable secret-sharing). Let \mathcal{M} be a (t_ℓ, t_s) –bounded adversary, and λ be a security parameter, and let a dealer p_d share a secret s with $n - 1$ additional processes following a protocol σ , which has a dealer p_d and executes in four different phases: setup, distribution, verification and reconstruction (See Definition 2.2.6). Then, σ is an $(n, \mathcal{M}, \lambda)$ -APVSS protocol if it satisfies the following properties:

- *Verifiability:* If the check in the verification step returns 1, i.e. succeeds, then with probability at least $1 - \epsilon(\lambda)$ the encryptions \mathbf{c} of a secret s shared by the dealer p_d are valid shares of some secret. Furthermore, if the check in the Reconstruction phase passes then the communicated values \mathbf{c} are indeed the shares of a secret distributed by the dealer.
- *Correctness:* if p_d is honest, then with probability at least $1 - \epsilon(\lambda)$ the checks in the verification and reconstruction steps succeed, and honest processes can reconstruct s .
- *Secrecy:* If p_d is honest, then the probability of \mathcal{M} learning any information about p_d 's secret s prior to the reconstruction phase is at most $\epsilon(\lambda)$.
- *Agreement:* If $t + d \leq t_\ell$, then honest processes do not reconstruct different secrets, even if p_d is faulty, with probability at least $1 - \epsilon(\lambda)$.
- *Accountability:* If $t + d \leq t_s$, $t \leq t_\ell$ and honest processes reconstruct different secrets, then honest processes eventually identify $t_\ell + 1$ processes responsible for such a disagreement, p_d being one of the faulty processes, with probability at least $1 - \epsilon(\lambda)$. Also, if Υ is the set of all secrets reconstructed by honest processes, then $|\Upsilon| \leq t_\ell + 1$.

The properties of verifiability, correctness and secrecy are common to PVSS schemes. Agreement is an adaptation of the commitment property [145] to the partially synchronous model, where one cannot rule out non-termination from a Byzantine dealer. We include an additional property for the APVSS problem, accountability, in order to cope with an adversary controlling more than t_ℓ processes. Since we keep the same n , t_ℓ , t_s and λ parameters, in the remainder of this document we abuse notation by referring to an $(n, \mathcal{M}, \lambda)$ -APVSS protocol as an APVSS protocol, and to an $(n, \mathcal{M}, \lambda)$ -SRB protocol as an SRB protocol.

6.3.2 Pinakion⁺ and Kleroterion⁺

We illustrate here the modifications for Pinakion and Kleroterion in order to solve the APVSS and SRB problems, in what we call Pinakion⁺ and Kleroterion⁺.

Pinakion⁺. For Pinakion⁺, we simply add verification to the AARB-broadcast protocol shown in Algorithm 10. The verification is the same as that of RBV-broadcast, in that all received messages are discarded unless they pass the call to `Pinakion.verify`. Following previous terminology, we refer to this variant as AARBV-broadcast (AARB-broadcast with verification).

Kleroterion⁺. Similarly to how Kleroterion executes n instances of Pinakion, Kleroterion⁺ executes n instances of Pinakion⁺. We also refer to other works on how to modify the consensus protocol of Algorithm 15 to make it accountable [181, 132]. Although there is no limitation preventing this algorithm from being actively accountable following the same approach as Basilic, this modification is out of the scope of this dissertation. As a result, for ease of exposition we simply assume that deceitful faults eventually give up on trying to cause a disagreement and contribute to reaching consensus. The reconstruction and aggregation remains the same as that of Kleroterion (except for the reconstruction threshold and number of secrets aggregated, which increases from t_ℓ to t_s), and is executed after terminating the ASMR consensus instance.

Then, we embed these n instances of Pinakion⁺ followed by this actively accountable variant of SPURT's consensus replacing the ASMR consensus of ZLB, allowing Kleroterion⁺ to deal with potential disagreements on the decided secrets analogously to how ZLB merged decisions in the event of a disagreement in Chapter 5. We explain later in this section how to merge random outputs in the event of a disagreement in order to preserve unpredictability and bias-resistance. We use the variant of ZLB that tolerates t_s faults, (i.e. $h'_0 = 2n/3$ as shown in Table 5.3), although Kleroterion⁺ can be tweaked in the same way as ZLB to tolerate different properties and adversaries.

Reconstruction threshold. Since Kleroterion⁺ must preserve unpredictability and bias-resistance against an adversary that controls up to t_s processes, secrets are embedded in polynomials of degree t_s (instead of t_ℓ as was the case for Kleroterion). This means that secrets can only be reconstructed upon receiving $t_s + 1$ secret shares. For the same reason, we require the consensus protocol to decide on and aggregate at least $t_s + 1$ secrets, so that at least one of the secrets will be shared by an honest process, by increasing the required Hamming weight of the bitmask proposal to $t_s + 1$. These two modifications ensure that t_s faulty processes can neither predict nor bias the final random output, as at least one of the inputs will remain unknown to

them until at least one honest process participates in the reconstruction.

Optimizations. The aggregation of $\mathcal{O}(n)$ secret-shares per recipient by each dealer performed in Kleroterion was possible because the consensus algorithm guaranteed that at least $t_\ell + 1$ honest processes had received a valid aggregation, and thanks to the reconstruction requiring secret shares from only $t_\ell + 1$ processes. Unfortunately, a direct consequence of increasing the reconstruction threshold in order to require $t_s + 1$ secret shares means that this optimization is not possible in Kleroterion⁺. Instead, each process broadcasts all secret shares, and locally verifies all shares (even if they are not the recipient of the share) of all secrets.

Merging decisions. Since the adversary can now cause a disagreement, in the event of a disagreement into a branches, an adversary controlling f processes can break unpredictability and bias-resistance once it receives at least $t_s + 1 - f$ decrypted shares from honest processes in at least $t_s + 1 - f$ secrets shared by honest processes.

Figure 6.4 is an example attack that an adversary \mathcal{M} controlling more than t_ℓ of the processes can perform in order to bias one of the disagreeing outputs. The adversary executes the protocol normally with one of the honest partitions (A in the example), only with the purpose of learning the secret inputs shared by A . Then, it changes its input so that the result of aggregating its new input with the input it learned from A is both predictable and biased, and forces the other partition B to decide on the output resulting from aggregating these inputs. As a result, the output of B from Kleroterion⁺ is not really random, although processes in B will not know until they find out about the disagreement through accountability.

A variant of the attack can instead target the final output of merging the disagreeing outputs. Given a function that determines how to merge disagreeing outputs from Kleroterion⁺, \mathcal{M} can instead target a new input such that the output from merging the disagreeing decisions is the biased, predictable output.

These two attacks can be prevented by making sure that there is at least a pair of distinct secrets aggregated for Z_A and Z_B , one for each, shared by honest processes. For this reason, we make an additional modification to the consensus protocol: once the leader proposes a bitmask, processes only decide a proposal if they gather a signature justifying the decision from each dealer of all values included in the proposal, signing the hash of the list of decided proposals according to the bitmask, and epoch. Processes thus wait at least Δ in each phase before proceeding on to the next. While this modification can impact performance (in that some bitmasks will include proposals from processes that have become non-responsive), it does not prevent termination, since eventually there will be an epoch after GST in which all deceitful processes behave as honest (at least in this branch of a disagreement, or instead after a period of time), the leader is honest and the leader proposes a list of secrets containing $t_s + 1$ secrets shared by responsive processes, and thus termination is guaranteed in this epoch.

6.4 Using Kleroterion⁺ for committee sortition

In this section, we analyze the probability that an adversary will break the randomness of our random beacon, Kleroterion⁺, and compare it with the state of the art. In this example, an

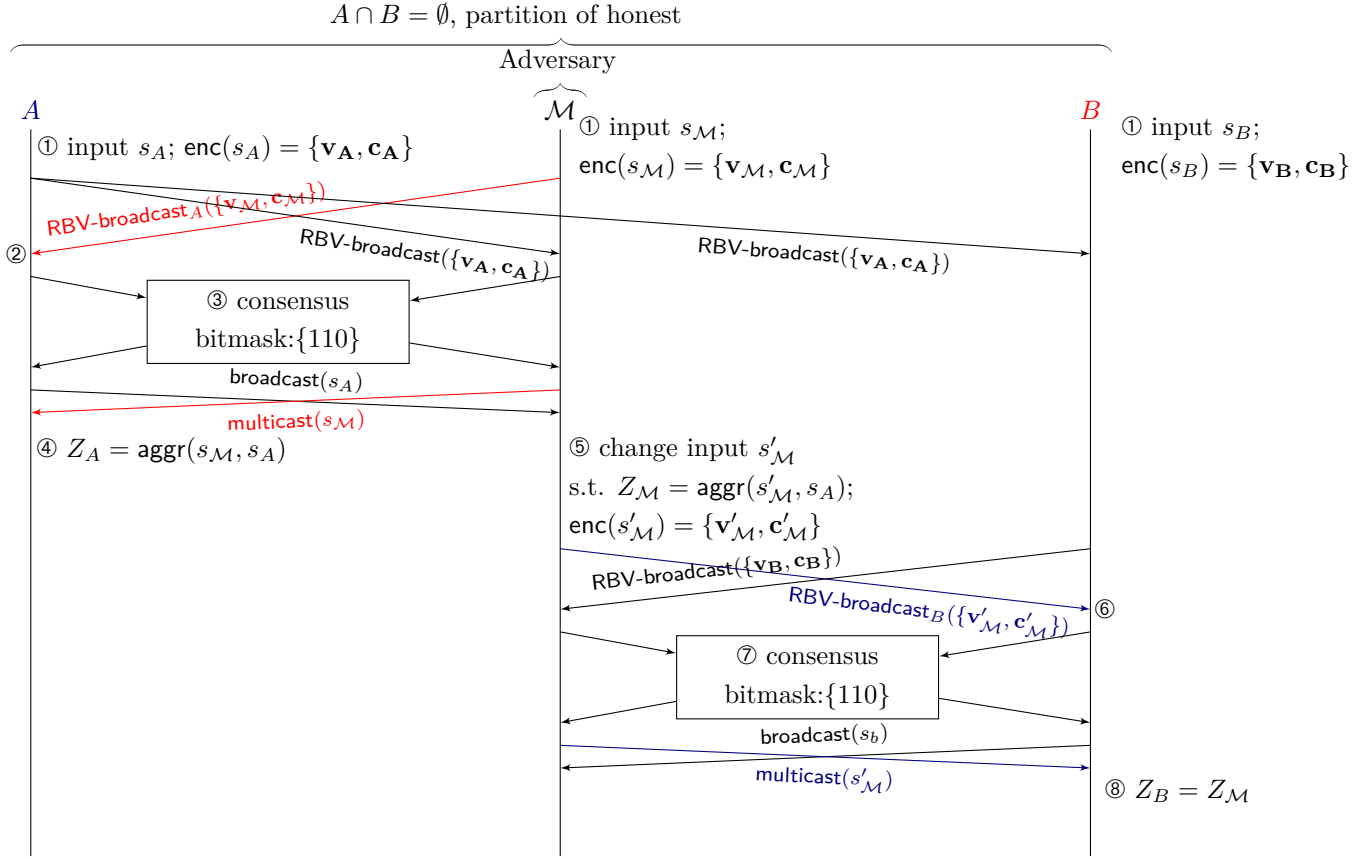


Figure 6.4: Example attack on Kleroterion⁺ by adversary \mathcal{M} controlling t_s faults with $n = 3$. First ① each process generates their respective inputs and encrypt them, generating secret shares and commitments. Then, ② \mathcal{M} RBV-broadcasts its commitments and secret shares only with process A, and RBV-delivers their commitments and secret shares. Following, ③ \mathcal{M} forces deciding on the bitmask 110 in order to aggregate their input with A's. Then, ④ A decides on a random output Z_A derived from aggregating their secret s_A with that of \mathcal{M} s_M . This means that both A and \mathcal{M} have disclosed their inputs to one another. At this point, ⑤ \mathcal{M} can compute a new input s'_M such that the result of aggregating s'_M and s_A is a new biased and predictable output Z_M . Then, \mathcal{M} RBV-broadcasts the new commitments and secret shares only with process B, ⑦ forcing decision on the same bitmask 110 during consensus. As a result, not only do A and B disagree on the output, but B's output, seemingly random to B, is in fact ⑧ the biased, predictable output Z_M . \mathcal{M} can instead do the analogous attack so that the resulting resolution of both outputs upon detecting the disagreements is the desired random and predictable output Z_M .

output of Kleroterion⁺ in an iteration i selects the committee for the next iteration $i + 1$ of Kleroterion⁺. Notice then that the membership change from ZLB is executed only to generate an output in that particular consensus instance (for convergence), but it is instead the Kleroterion⁺ committee sortition protocol that will select the committee for the following iteration.

Model. In each iteration i of the protocol, there is a committee N_i of size $|N_i| = n$ that executes the i -th iteration of the random beacon. Processes are selected for the i -th committee following the output of the $(i - 1)$ -th committee, which selects processes from a set M of users, of size $|M| = m$. We refer to the set M as the pool of process candidates.

We define here an output of a random beacon being *random* if that output is unbiased and unpredictable. We say that a coalition that breaks one of these properties *breaks the randomness* of the output. The Kleroterion⁺ protocol that we propose in Section 4.3.1 implements a random committee sortition protocol as per the categorization shown in Section 2.1.5, and thus it can be vulnerable to a coalition breaking the randomness of the output. However, Kleroterion⁺ extends the threshold at which the coalition can break the randomness of the output from $t_\ell + 1$ to $t_s + 1$, significantly decreasing the probability of an adversary biasing or predicting an output. We assume for the model in this section that cryptography does not break, in order to see the probabilities of failure of the Kleroterion⁺ protocol and compare it with other protocols using comparable cryptographic assumptions.

For this purpose, let us denote with t_m and d_m the total number of Byzantine and deceitful processes in the pool of process candidates, respectively, and $\mathbf{c}_m = t_m + d_m$.

6.4.1 Probability of randomness of the random beacon

The random variable $X(n, m, \mathbf{c}_M, j, h)$ represents the probability that there will be j processes controlled by the adversary in a committee of size n , given that the total number of processes is m of which \mathbf{c}_M are controlled by the adversary. h refers to the voting threshold, which is set to $h = t_s + 1$. We are interested in making $Pr(X > t_s)$ negligible:

$$Pr(X \geq h) = \sum_{j=\min(\mathbf{c}_m, h)}^{\min(\mathbf{c}_m, n)} X(n, m, \mathbf{c}_m, j, h) = \sum_{j=\min(\mathbf{c}_m, h)}^{\min(\mathbf{c}_m, n)} \frac{\binom{\mathbf{c}_m}{j} \binom{m-\mathbf{c}_m}{n-j}}{\binom{m}{n}}$$

This value can be approximated when m is much greater than n [26], by setting $p_{\mathbf{c}_m} = \frac{\mathbf{c}_m}{m}$ with the following binomial cumulative distribution function $Y(n, m, p_{\mathbf{c}_m}, h)$:

$$Pr(Y \geq h) = 1 - Pr(Y < h) = 1 - \sum_{j=0}^{\lfloor h-1 \rfloor} \binom{m}{j} p_{\mathbf{c}_m}^j (1 - p_{\mathbf{c}_m})^{n-j}$$

Figure 6.5 shows the probability that the adversary breaks the randomness of the random beacon (i.e. $Pr(Y > t_s)$) for a percentage of faulty processes $p_{\mathbf{c}_m} = \frac{\mathbf{c}_m}{m}$ ranging between 0.2 and $29/45 = 0.6\bar{4}$, and a committee size between 5 and 100 processes (Figure 6.5a), and also between 100 and 2000 processes (Figure 6.5b). Note that we delimit the lowest value to be $\frac{1}{2^{53}} = 1.1102230246251565 \cdot 10^{-16}$ for convenience, ease of exposition, and because 10^{16} (a quadrillion) is the order of magnitude of the number of minutes that have passed in the

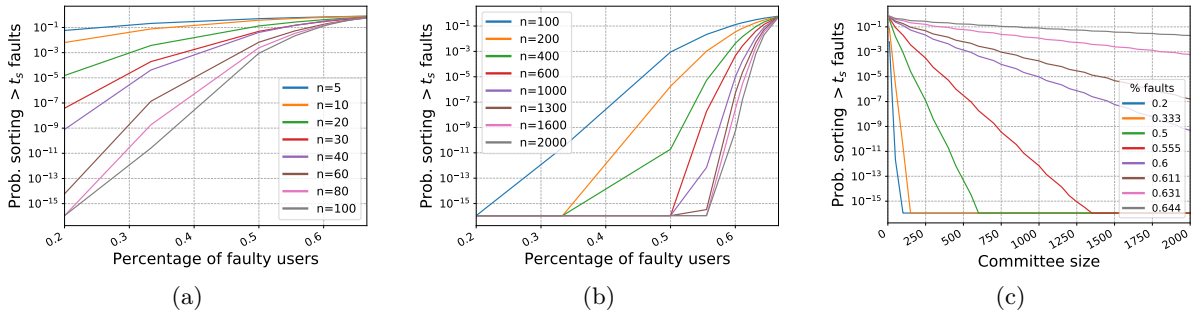


Figure 6.5: Probability of the adversary having at least $t_s + 1$ processes in the committee $Pr(Y > t_s)$ per percentage of faulty processes p_{cm} for a variety of committee sizes n (left and center), and per committee size n for a number of percentages of faulty processes p_{cm} (right).

Universe since the Big Bang ², meaning that this attack should never happen. Nevertheless, the probabilities of many of these parameters are even much smaller than this limit of the plot, reaching it's lowest value for $n = 2000$ and $p_{cm} = 0.2$ for which $Pr(Y > t_s) < 9 \cdot 10^{-446}$, while for the same parameters an adversary can violate safety of Algorand with probability $5 \cdot 10^{-9}$ [26]. To put it in perspective, $9 \cdot 10^{-446}$ is so low that it is more likely that two people randomly guess the same atom out of all the atoms in the observable universe 5 consecutive times ³.

Additionally, in Figure 6.5c, we show the probability that $Pr(Y > t_s)$ per committee size, given specific percentages of faulty processes p_{cm} . Note that it is evident from Figure 6.5 that as the percentage of faulty processes approaches asymptotically the bound of $p_{cm} = \frac{2}{3}$, the probability that the adversary will break the randomness of the random beacon increases dramatically. For example, for a committee size of $n = 1000$, if $p_{cm} = 0.5$ then the probability $Pr(Y > t_s)$ falls within the bound of $Pr(Y > t_s) \leq \frac{1}{2^{53}}$, same as for all lower values $p_{cm} < 0.5$. However, still for $n = 1000$, only increasing $p_{cm} = \frac{5}{9} \approx 0.555$ increases the probability to $Pr(Y > t_s) = 6.97 \cdot 10^{-13}$, while $p_{cm} = 0.6$ yields $Pr(Y > t_s) = 9.7 \cdot 10^{-6}$, or for $p_{cm} = \frac{24}{38} \approx 0.632$ then $Pr(Y > t_s) = 0.012674$, meaning that the output of the beacon will not be random after 100 iterations, in expectation.

In Figure 6.6 we compare the safety of our random beacon to the safety of Algorand. For this purpose, we check the required committee size n and percentage of faulty processes p_{cm} to provide the same security that Algorand's authors provide in their work (figure 3 [26]). We can see that for the same level of safety of $5 \cdot 10^{-9}$, Kleroterion⁺ resists an adversary controlling a much greater number of processes, up to 60% of them for just $n = 2000$, compared with 20% in Algorand. Similarly, for the same percentage of faulty processes, Kleroterion⁺ requires a much smaller size of the committee, with $n = 39$ for a $p_{cm} = 0.2$ compared with Algorand's $n = 2000$. This is particularly relevant for Kleroterion⁺, since that means that, even if the adversary would control between $t_\ell + 1$ and t_s processes in the committee, by accountability the adversary could only cause a disagreement into at most $t_\ell + 1 = 13$ values for $n = 39$, all of which would be unpredictable and bias-resistant, and would eventually converge to just 1 value, also random.

²<https://81018.com/universeclock/>

³<https://www.physicsoftheuniverse.com/numbers.html>

Thus, the added complexity of Kleroterion⁺ is counteracted by the decrease in the number of processes in the committee compared to Algorand, in that the number of processes running the protocol decreases by more than 80%.

6.4.2 Comparison with the state of the art

Note that Figure 6.6 compares the probability of violating safety or liveness of Algorand with the probability of breaking randomness of the random beacon in Kleroterion⁺ $Pr(Y > t_s)$. A violation of safety in Algorand means the adversary is able to select the most favorable one random output from among all the deterministically valid, predictable proposals. The number of valid proposals range between 1 and 70 for Algorand [26]. In contrast, Kleroterion⁺'s sortition is a random committee rotation, meaning that if the adversary can select one output once, it can select all future random beacon outputs from then on.

Also recall that, as we already showed, if $t_s \geq t + d > t_\ell$ then the adversary cannot predict nor bias outputs, but it could cause a temporary disagreement into at most $t_\ell + 1$ distinct random outputs, before the disagreement is discovered, resolved, and the adversary is punished. We also show the probability of this case $Pr(Y > t_\ell)$ in Figure 6.6 by fixing this probability to the same values $5 \cdot 10^{-9}$ and 10^{-14} and comparing the committee size n and percentage of faulty processes p_{c_m} with Algorand. We can see that this case, though more comparable to Algorand's violation of safety, still requires a significantly lower committee size than that of Algorand's in order to guarantee the same level of safety, with $n > 391$ for $p_{c_m} = 0.2$ and $Pr(Y > t_\ell) < 5 \cdot 10^{-9}$. If we only consider Byzantine processes for the random variable Y and for p_{c_m} , then the plot $Pr(Y > t_\ell)$ shows the probability of losing liveness (availability) of Kleroterion⁺'s sortition. Also under this figure, we have the probability of a coalition breaking the randomness of the random beacon for state-of-the-art sortition protocols based on random beacons such as SPURT [31], Scrape [143], RandSolomon [141] or Kleroterion (Section 6.2). Also, these works satisfy neither agreement nor availability with probability $Pr(Y > t_\ell)$, and they already do not satisfy accountability. We thus show a significant improvement of Kleroterion⁺ compared with previous works, since it satisfies bias-resistance, unpredictability and accountability with probability $Pr(Y \leq t_s)$, while it maintains the rest of the properties of the state of the art for $Pr(Y \leq t_\ell)$.

6.5 Summary

In this chapter, we outlined the importance of randomly sorting the committee for blockchains and other applications of repeated consensus. We then democratized the recent SPURT protocol [31] to improve its performance in what we call the Kleroterion protocol, with the help of our novel Pinakion protocol for PVSS. Kleroterion exchanges a number of bits per network channel independent of the size of the participants in the protocol per random output, except for one message of size n sent by the leader of the epoch and for the reconstruction phase, and without requiring a trusted setup. Following, we formulated the SRB problem and APVSS problem and propose Kleroterion⁺ and Pinakion⁺ that solve SRB and APVSS, respectively. Finally, we

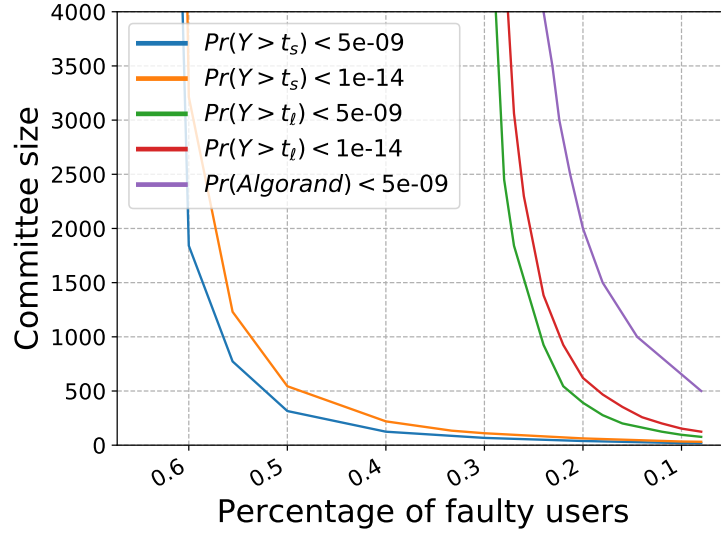


Figure 6.6: Minimum committee size per percentage of faulty processes in order to guarantee a probability of having at least $t_s + 1$ and $t_\ell + 1$ faulty processes in the committee to at most $5 \cdot 10^{-9}$ and 10^{-14} , compared with the probability of having a violation of safety or liveness in Algorand ([26], Figure 3), respectively.

illustrated the security of Kleroterion⁺ in a blockchain application that uses it for committee sortition, compared to recent blockchains, random beacons, and committee sortition protocols.

We believe this chapter closes our argument in the quest for a blockchain that delivers provable guarantees of a level of security and decentralization that, to the best of our knowledge, has never been reached before in the same model, and with performance competitive to recent works that provide less security or decentralization. We put together the components of this so-called 'Blockchain of Oz' in Chapter 7, concluding this dissertation.

Chapter 7

Conclusion

In this dissertation, we have taken exciting steps towards narrowing down the space between the theoretical limitations of blockchains and the solutions that approach them. Our methodology has allowed us to maintain a strong bond between our results and previous works both from science and systems in the real world. The models and problems introduced throughout this dissertation were strongly motivated by previous, novel results that show vulnerabilities, attacks and impossibility results, highlighting a balanced compromise between security and performance. An example of this is our decision to design BFT protocols without strong assumptions like synchrony.

We considered with our protocols the impossibilities and trade-offs that are inherent to the problems here presented, in most cases reaching the impossibility bound in terms of both performance and tolerance to faults. Some of our solutions are not only justified and formally proven and analyzed, but also disruptive and requiring a combined knowledge of game theory, distributed computing and cryptography. We hope that the works outlined in this dissertation inspire its readers for future directions.

The combined works of this dissertation provide a blockchain that:

1. tolerates an **unprecedented percentage of users controlled by the adversary** while preserving **decentralization** and randomness with very high probability, thanks to Kleroterion⁺'s random beacon and its derived committee sortition, or instead ensures decentralization through a random beacon protocol that exchanges a number of **bits independent of the size of the participants** in the protocol, without requiring a trusted setup, per network channel and random output, thanks to Kleroterion's random beacon,
2. ensures that **no rational coalition** controlling a minority of the committee will succeed at **breaking consensus**, thanks to TRAP's baiting and financial component,
3. further enhances security by **tolerating almost a supermajority of faults** provided they are heterogeneous and not necessarily all Byzantine faults, thanks to the Basilic class,
4. resolves unlucky cases in which either a majority of rational attackers or more than a third of irrational (or Byzantine) attackers collude, ensuring that **no honest participant suffers** the outcome of the attack, thanks to ZLB and zero loss,

5. provides **scalability** measured in its communication and computation complexities relative to previous works,
6. but also providing **trade-offs that allow our protocols to adjust** to the level of security and type of models and performance requirements, such as the Basilic class, the voting threshold of ZLB's membership change, or selecting between Kleroterion⁺ or Kleroterion.

Although we believe that our journey in the making of this dissertation has been fertile, as we show in the outcome of our research objectives in Section 7.1, and fortunately for researchers and engineers in the blockchain world, a myriad of questions and research directions remain to be explored. We list a number of them in Section 7.2, with the goal that a so-called 'marvelous Blockchain of Oz' that provably achieves the best possible metrics for all trade-offs in the desired properties becomes a reality, and not just an empty pledge.

7.1 Outcome of Research Objectives

In this section, we cluster the contributions of this dissertation around the research objectives previously outlined.

Objective 1: formally define blockchains and its properties

The formal definition of our protocols has been done primarily in Chapter 2, inspired from previous work. However, we presented in each chapter novel formalizations, problems and definitions of our own making. Chapter 3 introduces a number of formal definitions, such as those of the offchain, childchain and sidechains problems. Chapter 4 defines baiting strategies and the rational agreement problem, as well as an adaption of the asynchronous and synchronous models of Abraham et al. [71, 72] to partial synchrony, and to the crash-fault model, with the introduction of crash-robustness, crash-baiting strategies and crash-immunity. Chapter 5 formalizes the Byzantine-deceitful-benign (BDB) model, as well as the Longlasting Blockchain problem, and additional properties like α -confirmation, awareness, active accountability, or zero loss. Finally, Chapter 6 introduces the accountable PVSS and secure random beacon problems.

Objective 2: state and formally prove impossibility bounds and trade-offs, and propose sensible metrics for comparison

Chapter 2 recalls previous impossibility results [207, 11, 13], and states complexity metrics like the computation and communication complexities. It extends communication complexities to justify considering normalized and amortized complexities, and per route complexities. It also proposes qualitative metrics in that our protocols work without synchrony or other strong assumption. Chapter 3 proposes a number of metrics to measure the success of the Lockdown attack depending on the strength of the adversary and other parameters of the channel network, such as the Attack Effort Ratio (AER) or the Total Blocked Time (TBT). This chapter also shows that it is impossible to have a non-trivial offchain protocol without synchrony if $f > t_1$

or $f > t_0$ in the Byzantine model. Chapter 4 measures the robustness of a protocol by the combination of k rational and t Byzantine players controlled by this adversary. It also measures the reward and deposit required per upper-bound on the maximum gain from deviating. Additionally, it shows that it is impossible to solve the rational agreement problem without implementing a baiting strategy. This result is later extended in the crash-fault rational model, posing an interesting trade-off in that protocols that are resilient-optimal in the crash-fault model do not tolerate even one rational player in addition to these crash faults. Similarly, Chapter 5 measures the security of a consensus protocol in the BDB model by its tolerance to a combination of t Byzantine, d deceitful and q benign faults, while also proving that it is impossible to solve consensus in the BDB model if $n \leq 3t + d + 2q$. Furthermore, an interesting trade-off derives in that protocols must commit to a specific voting threshold $h \in (n/2, n]$, tolerating then at most $d + t < 2h - n$ and $q + t \leq n - h$ faults. Our Zero-loss Blockchain (ZLB) then proposes three different settings depending on the voting threshold that are resilient-optimal in the properties they guarantee, each with its own advantages and drawbacks. Finally, Chapter 6 proposes two versions of our random beacon, and analyzes its advantages and drawbacks, as well as it proposes a metric to compare with previous works in an application to committee sortition.

Objective 3: design and prove solutions with competitive metrics and bounds

Chapter 3 proposes Platypus, an offchain protocol without synchrony, and shows that its communication complexity is lower or comparable to consensus protocols in the same model. Chapter 4 shows that TRAP is an ϵ -(k, t)-robust consensus protocol for $n > \max(\frac{3}{2}k + 3t, 2(k + t))$. We also show in the same chapter that TRAP is also ϵ -($k + t, t$)-crash-robust for the same values of k and t . Chapter 5 proves the optimal resilience of the Basilic class of consensus protocols in both the BDB and the Byzantine failure models, while also proving its optimal communication complexity as proven for accountable consensus protocols [181]. Building upon different protocols from the Basilic class, ZLB also proves to be a blockchain that tolerates a colluding majority while being resilient-optimal in the Byzantine fault model. We also analyze in the same chapter the models in which ZLB solves the properties of awareness, α -confirmation and zero loss. We show in Chapter 6 that our Kleroterion protocol is a resilient-optimal random beacon protocol that exchanges a number of bits per network channel independent of the size of the participants in the protocol without requiring a trusted setup, providing a significant improvement with respect to previous work. Finally, our analysis in Chapter 6 shows that Kleroterion⁺ offers an unprecedented level of decentralization for blockchains, given that the probability that an adversary controlling even half of the coins of the network will be able to break the randomness in order to select the committee is lower than 10^{-9} for a committee size of $n = 300$ processes.

Objective 4: implement and test proposed solutions in a real environment

Chapter 3 shows the results from implementing the Lockdown attack on the official Lightning Network testnet, while specifying that the only reason why the attack was not performed in the mainnet was for ethical concerns that might impact real applications of the mainnet. At the

same time, we performed a responsible disclosure to the developers of the LN implementations. Then, we showed the results of a replicated topology derived from a snapshot of the network, for further results. Chapter 5 shows the empirical results of implementing ZLB and Basilic and testing them in two distributed settings. Chapter 5 also shows results with a simulation of various numbers of attackers trying to cause a disagreement and artificially injecting communication delays between partitions of honest processes, validating the performance and security of the claimed results. Unfortunately, the time constraint and wide range of proposed projects led us to focus on formalizing, proving and specifying first all of our aforementioned protocols. As a result, we left the implementation of some of our protocols as part of our future work, as we list in Section 7.2.

7.2 Future Work

Implement all protocols. We took a sensible decision to focus on the formalization, design, and proofs of all of these protocols before implementing them. Although the results presented in this dissertation are extensive and justified, the time constraint prevented us from implementing and testing them all, which we set as key goals for future work. It is important to note that testing these protocols requires of careful consideration, such as modeling rational behavior in an implementation of rational players, or testing multiple parameters like the number and percentage of users controlled by the adversary in a committee sortition application of Kleroterion and Kleroterion⁺, which may lead to multiple future works.

Improve robustness of TRAP thanks to Pinakion⁺. The results of this dissertation feed one another in ways that we had not foreseen at the beginning of each of them. One of these instances is the fact that our Pinakion⁺ solving APVSS can be used to replace the BFTCR protocol of the baiting component of our TRAP consensus protocol, potentially improving its robustness beyond its current bound of less than half of Byzantine faults and rational players.

Study variants of Kleroterion⁺. Kleroterion⁺ showcases an unprecedented level of randomness at a cost on performance and without improving liveness. We argue in Appendix E that it would be interesting to study the performance of our proposed approach with an equivalent implementation that instead runs multiple parallel executions of smaller committees. Similarly, we believe a study of Kleroterion⁺ in the BDB model with a varying threshold to be of great interest for blockchains.

Measure democratic and leader-less protocols. Another instance of our projects benefiting from one another is the recent discussion that gave way to Kleroterion and Kleroterion⁺ implementing a democratic, leader-based consensus protocol instead of a leader-less one. This leads to an interesting research question on the advantages and drawbacks of both approaches, which makes for compelling future work.

Integrate these results with additional properties. Although we focused here on the core properties of blockchains, we do not mention other properties that are relevant to many

applications. Examples of these are supporting smart contracts and its implications, totally ordering transactions (i.e. Byzantine ordered consensus [208]), or preventing proposal duplication in democratic protocols [27]. Although we conjecture these properties to be orthogonal to our protocols, a proper system that integrates them makes an interesting research direction.

Integrating all protocols into one system. We believe that the solutions presented in this dissertation, after being improved, implemented and tested, and the orthogonal techniques that provide interesting properties, must converge towards an implementation of one system that provides a number of customizable parameters depending on the application. This system would benefit from the decentralization of Kleroterion⁺ or Kleroterion, the security provided by TRAP, Basilic, and ZLB, and support for hierarchies of consensus thanks to Platypus, and a competitive level of scalability that can be improved for specific applications that can relax the security assumptions. We design all of our protocols with a system like this one in mind, the so-called 'Blockchain of Oz'.

Notations

General

N	committee of participants executing the protocol
n	$n = N $ committee size
p_i	a process of the committee p_i
\mathcal{M}	the adversary
t_ℓ	classical bound for Byzantine fault tolerance $t_\ell = \lceil \frac{n}{3} \rceil - 1$
t_s	$t_s = \lceil \frac{2n}{3} \rceil - 1$
f	total number of faulty processes
t	number of Byzantine processes

Layer-2 without synchrony

θ	expiration blockheight of a payment
δ	decremental value of θ per hop of a payment
T_{max}	maximum locking time of a payment (measured in number of blocks)
C_{AB}	capacity of channel between A and B
o_{AB}	balance of A in channel between A and B
Ω	blockchain
Γ	offchain protocol
Ψ	childchain
M_Ψ	set of users in Ψ
m_Ψ	size of M_Ψ
N_Ψ	set of processes in Ψ 's committee
n_Ψ	size of N_Ψ
t_0	maximum number of tolerated Byzantine faults in blockchain, i.e. $t_0 = \lceil n_\Omega/3 \rceil - 1$
t_1	maximum number of tolerated Byzantine faults in childchain, i.e. $t_0 = \lceil n_\Psi/3 \rceil - 1$
h_0	threshold of signatures to create childchain
$\varphi(\mathfrak{c}_i, \delta_i)$	function that takes a coin and a time and returns the owner
$\gamma(z)$	function that, given account z , returns the user that controls z
TR	transfer relation, such that $z_i TR_{\delta_{i+1}, text{\mathfrak{c}}_i} z_j \iff \varphi(\mathfrak{c}_i, \delta_{i+1}) = u_j$

Rationality for blockchains' consensus

$u_i(\vec{\sigma})$	utility for player i provided all players follow joint strategy $\vec{\sigma}$
k	number of rational players
t	number of Byzantine (or crash) players
t'	number of crash players
e	required number of baiting players
$e(k, t)$	minimum integer such that $e(k, t) \geq e$
\mathfrak{G}	gain for attackers from causing a disagreement into 2 outputs
g	gain per colluding rational player, i.e. $g = \frac{\mathfrak{G}}{k}$
\mathfrak{R}	reward for baiting
\mathfrak{L}	deposit per player $\mathfrak{L} = \mu \mathfrak{G}$ for some $\mu > 0$
$\vec{\sigma}$	protocol, recommended joint strategy
$\vec{\eta}$	baiting strategy
ρ	probability of being selected as winner of the bait $\rho(e) = \frac{1}{e}$
$\bar{\rho}$	$\bar{\rho} = 1 - \rho$

ZLB, a blockchain tolerating colluding majorities

N'	updated set of processes in the committee
n'	$n' = N' $ updated number of processes
d	number of deceitful processes
q	number of benign processes
t	number of Byzantine processes
f	$f = t + d + q$, total number of faulty processes
δ	deceitful ratio $\delta = \frac{d+t}{n}$
h	voting threshold of the consensus protocol
h_0	initial voting threshold
d_r	number of detected faulty processes
$h(d_r)$	updated threshold $h(d_r) = h_0 - d_r$
f_d	threshold of proofs-of-fraud to start membership change
Φ_k	Consensus instance k of ZLB
m	number of processes in the pool of process candidates
a	number of branches in a disagreement
h'	voting threshold of the inclusion and exclusion consensus
\mathfrak{G}	gain from each additional branch of a disagreement, for a total gain of $(a - 1)\mathfrak{G}$
\mathfrak{D}	deposit per coalition of attackers, with $\mathfrak{D} = b\mathfrak{G}$ for some $b > 0$
ρ	probability of attack on a consensus instance being detected
$\hat{\rho}$	$\hat{\rho} = 1 - \rho$
\mathcal{G}	expected gain from attack
\mathcal{P}	expected deposit loss (as punishment) from attack
ξ	deposit flux from attack $\xi = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$, $\xi \geq 0$ for zero loss
w	finalization blockdepth, measured in number of appended blocks required

Kleroterion⁺, randomness with colluding majorities

\mathfrak{G}	maximum total gain per disagreeing block
h	voting threshold
β	discount factor that decreases the gain from disagreeing in future iterations $\beta \in [0, 1)$
x	length of disagreeing attack, measured in number of blocks
M	total set of users
m	number of users $m = M $ that can be selected for the committee
\mathcal{C}	set of the adversarial coalition of users
\mathfrak{c}_m	coalition size $\mathfrak{c}_m = \mathcal{C} $
$p_{\mathfrak{c}_m}$	percentage of adversarial users $p_{\mathfrak{c}_m} = \frac{\mathfrak{c}_m}{m}$
\mathfrak{c}_n	number of users from the coalition selected for the committee
$\Pr(\mathfrak{c}_n > t_\ell)$	Probability of selecting at least t_ℓ users from the coalition for the committee
w	finalization blockdepth, measured in number of blocks before finalizing a decision
\mathfrak{D}	deposit per coalition of attackers, with $\mathfrak{D} = b\mathfrak{G}$ for some $b > 0$
\mathcal{G}	expected gain from attack
Υ	set of verifiable decided outputs from a disagreement
a	number of branches in a disagreement
ξ	deposit flux from attack $\xi = \mathcal{P}(\hat{\rho}) - \mathcal{G}(\hat{\rho})$, $\xi \geq 0$ for zero loss
ρ	probability that the attack succeeds
$\hat{\rho}$	$\hat{\rho} = 1 - \rho$

Bibliography

- [1] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2008. URL: <https://bitcoin.org/bitcoin.pdf>.
- [2] *Bitcoin Gold*. URL: <https://bitcoingold.org/>.
- [3] *Bitcoin Cash*. URL: <https://bitcoincash.org/>.
- [4] *Litecoin*. URL: <https://litecoin.org/>.
- [5] Investopedia, ed. *A History of Bitcoin Hard Forks*. URL: <https://www.investopedia.com/tech/history-bitcoin-hard-forks/>.
- [6] *Dogecoin*. URL: <https://dogecoin.com/>.
- [7] *Shiba Token*. URL: <https://shibatoken.com/>.
- [8] David Segal. *Going for Broke in Cryptoland*. Ed. by The New York Times. URL: <https://www.nytimes.com/2021/08/05/business/hype-coins-cryptocurrency.html>.
- [9] Leslie Lamport. “Using time instead of timeout for fault-tolerant distributed systems.” In: *ACM Transactions on Programming Languages and Systems*. 1984.
- [10] Leslie Lamport and Nancy Lynch. “Distributed computing: Models and methods”. In: *Formal models and semantics*. Elsevier.
- [11] Leslie Lamport, Robert Shostak, and Marshall Pease. “The Byzantine Generals Problem”. In: *ACM Transaction on Programming Languages and Systems*. 1982.
- [12] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. “Impossibility of distributed consensus with one faulty process”. In: *Journal of the ACM*. 1985.
- [13] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. “Consensus in the presence of partial synchrony”. In: *Journal of the ACM*. 1988.
- [14] Vitalik Buterin. *Why sharding is great: demystifying the technical properties*. 2021. URL: <https://vitalik.ca/general/2021/04/07/sharding.html>.
- [15] Emmanuelle Anceaume, Antonella Del Pozzo, Romaric Ludinard, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. “Blockchain Abstract Data Type”. In: *Symposium on Parallelism in Algorithms and Architectures*. 2019.
- [16] Vitalik Buterin. *Long Range Attacks: The Serious Problem with Adaptive Proof-of-Work*. 2014. URL: <https://blog.ethereum.org/2014/05/15/long-range-attacks-the-serious-problem-with-adaptive-proof-of-work/>.

- [17] *Tezos' Proof-of-stake*. 2019. URL: https://tezos.gitlab.io/active/proof_of_stake.html.
- [18] *EOS' Consensus Protocol*. 2020. URL: https://developers.eos.io/welcome/v2.0/protocol-guides/consensus_protocol.
- [19] *Polkadot: Learn Randomness*. 2020. URL: <https://wiki.polkadot.network/docs/en/learn-randomness>.
- [20] *Babe: Blind Assignment for Blockchain Extension protocol*. 2020. URL: <https://w3f-research.readthedocs.io/en/latest/polkadot/block-production/Babe.html>.
- [21] *Cosmos Validator FAQ, becoming a validator*. URL: <https://hub.cosmos.network/main/validators/validator-faq.html>.
- [22] *VeChain's Proof-of-authority*. 2020. URL: <https://docs.vechain.org/thor/learn/proof-of-authority.html>.
- [23] Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. "DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains". In: *NCA*. 2018.
- [24] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol". In: *CRYPTO*. 2017.
- [25] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. "Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain". In: *EUROCRYPT*. 2018.
- [26] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies". In: *SOSP*. 2017.
- [27] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolic. *Mir-BFT: High-Throughput Robust BFT for Decentralized Networks*. 2019. URL: <http://arxiv.org/abs/1906.05552>.
- [28] Tyler Crain, Christopher Natoli, and Vincent Gramoli. "Red Belly: A Secure, Fair and Scalable Open Blockchain". In: *S&P*. 2021.
- [29] Danny Dolev and Rüdiger Reischuk. "Bounds on Information Exchange for Byzantine Agreement". In: *Journal of the ACM*. 1985.
- [30] Guillaume Vizier and Vincent Gramoli. "ComChain: A blockchain with Byzantine fault-tolerant reconfiguration". In: *Concurrency and Computation: Practice and Experience*. 2020.
- [31] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. "SPURT: Scalable Distributed Randomness Beacon with Transparent Setup". In: *S&P*. 2022.
- [32] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. "Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures." In: *CCS*. 2020.
- [33] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. "High-threshold avss with optimal communication complexity". In: *Financial Cryptography and Data Security*. 2021.

- [34] Joseph Y. Halpern and Xavier Vilàça. “Rational Consensus”. In: *PODC*. 2016.
- [35] Christian Cachin, Klaus Kursawe, and Victor Shoup. “Random oracles in Constantino-ple: Practical asynchronous Byzantine agreement using cryptography”. In: *Journal of Cryptology*. 2005.
- [36] 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: *SECP*. 2017.
- [37] Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. *Internet Computer Consensus*. 2021. URL: <https://ia.cr/2021/632>.
- [38] *Drand - a distributed randomness beacon daemon*. 2020. URL: <https://github.com/drand/drand>.
- [39] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. *Homomorphic Encryption Ran-dom Beacon*. 2019. URL: <https://eprint.iacr.org/2019/1320>.
- [40] Berry Schoenmakers. “A Simple Publicly Verifiable Secret Sharing Scheme and Its Ap-plication to Electronic Voting”. In: *CRYPTO*. 1999.
- [41] Mikhail Krasnoselskii, Grigorii Melnikov, and Yury Yanovich. “No-Dealer: Byzantine Fault-Tolerant Random Number Generator”. In: *INFOCOM workshops*. 2020.
- [42] Christopher Natoli, Jiangshan Yu, Vincent Gramoli, and Paulo Jorge Esteves Veríssimo. *Deconstructing Blockchains: A Comprehensive Survey on Consensus, Membership and Structure*. 2019. URL: <http://arxiv.org/abs/1908.08316>.
- [43] Gavin Wood. *Polkadot: Vision for a heterogeneous multi-chain framework*. 2016. URL: <https://polkadot.network/PolkaDotPaper.pdf>.
- [44] Gavin Wood. *Ethereum: A secure decentralized generalized transaction ledger*. 2015. URL: <https://gavwood.com/paper.pdf>.
- [45] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. “The Consensus Number of a Cryptocurrency”. In: *PODC*. 2019.
- [46] Atul Singh, Pedro Fonseca, Petr Kuznetsov, Rodrigo Rodrigues, and Petros Maniatis. “Zeno: Eventually Consistent Byzantine-Fault Tolerance”. In: *NSDI*. 2009.
- [47] Pierre Civit, Seth Gilbert, and Vincent Gramoli. “Polygraph: Accountable Byzantine Agreement”. In: *ICDCS*. 2021.
- [48] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, and Adi Serendinschi. *Crime and Punishment in Distributed Byzantine Deci-sion Tasks*. 2022. URL: <https://eprint.iacr.org/2022/121>.
- [49] Yonatan Aumann and Yehuda Lindell. “Security against covert adversaries: Efficient protocols for realistic adversaries”. In: *Journal of Cryptology*. 2010.
- [50] Vipul Goyal, Payman Mohassel, and Adam Smith. “Efficient two party and multi party computation against covert adversaries”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2008.

- [51] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. “Global-Scale Secure Multiparty Computation”. In: *CCS*. 2017.
- [52] Cheng Hong, Jonathan Katz, Vladimir Kolesnikov, Wen-jie Lu, and Xiao Wang. “Covert security with public verifiability: Faster, leaner, and simpler”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2019.
- [53] Gilad Asharov and Claudio Orlandi. “Calling out cheaters: Covert security with public verifiability”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. 2012.
- [54] Ivan Damgård, Claudio Orlandi, and Mark Simkin. “Black-box transformations from passive to covert security with public verifiability”. In: *Annual International Cryptology Conference*. 2020.
- [55] Peter Scholl, Mark Simkin, and Luisa Siniscalchi. *Multiparty Computation with Covert Security and Public Verifiability*. 2021. URL: <https://eprint.iacr.org/2021/366>.
- [56] Xavier Vilça, Oksana Denysyuk, and Luís Rodrigues. “Asynchrony and Collusion in the N-party BAR Transfer Problem”. In: *Structural Information and Communication Complexity*. 2012.
- [57] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, F Paris, and Sara Tucci-Piergiovanni. “Rational vs Byzantine Players in Consensus-based Blockchains”. In: *International Conference on Autonomous Agents and MultiAgent Systems*. 2020.
- [58] Adam Groce, Jonathan Katz, Aishwarya Thiruvengadam, and Vassilis Zikas. “Byzantine Agreement with a Rational Adversary”. In: *Automata, Languages, and Programming*. 2012.
- [59] Zahra Ebrahimi, Bryan Routledge, and Ariel Zetlin-Jones. *Getting blockchain incentives right*. 2019. URL: https://www.andrew.cmu.edu/user/azj/files/Blockchain_erz.pdf.
- [60] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. “BAR Fault Tolerance for Cooperative Services”. In: *SIGOPS Operating Systems Review*. 2005.
- [61] Itay Harel, Amit Jacob-Fanani, Moshe Sulamy, and Yehuda Afek. “Consensus in Equilibrium: Can One Against All Decide Fairly?” In: *PODC*. 2020.
- [62] Xavier Vilça, João Leitão, and Luís Rodrigues. “N-Party BAR Transfer: Motivation, Definition, and Challenges”. In: *International Workshop on Theoretical Aspects of Dynamic Distributed Systems*. 2011.
- [63] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J. Knottenbelt. “Balance: Dynamic Adjustment of Cryptocurrency Deposits”. In: *CCS*. 2019.
- [64] Adam Groce, Jonathan Katz, Aishwarya Thiruvengadam, and Vassilis Zikas. “Byzantine Agreement with a Rational Adversary”. In: *Automata, Languages, and Programming*. 2012.

- [65] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. “Distributed Protocols for Leader Election: A Game-Theoretic Perspective”. In: *ACM Transactions on Economics and Computation*. 2019.
- [66] Yehuda Afek, Yehonatan Ginzberg, Shir Landau Feibish, and Moshe Sulamy. “Distributed Computing Building Blocks for Rational Agents”. In: *PODC*. 2014.
- [67] Xiaohui Bei, Wei Chen, and Jialin Zhang. *Distributed Consensus Resilient to Both Crash Failures and Strategic Manipulations*. 2021. URL: <http://arxiv.org/abs/1203.4324>.
- [68] Anna Lysyanskaya and Nikos Triandopoulos. “Rationality and Adversarial Behavior in Multi-party Computation”. In: *Advances in Cryptology*. 2006.
- [69] Georg Fuchsbauer, Jonathan Katz, and David Naccache. “Efficient Rational Secret Sharing in Standard Communication Networks”. In: *Theory of Cryptography*. 2010.
- [70] Varsha Dani, Mahnush Movahedi, Yamel Rodriguez, and Jared Saia. “Scalable rational secret sharing”. In: *PODC*. 2011.
- [71] Ittai Abraham, Danny Dolev, Rica Gonen, and Joe Halpern. “Distributed Computing Meets Game Theory: Robust Mechanisms for Rational Secret Sharing and Multiparty Computation”. In: *PODC*. 2006.
- [72] Ittai Abraham, Danny Dolev, Ivan Geffner, and Joseph Y. Halpern. “Implementing Mediators with Asynchronous Cheap Talk”. In: *PODC*. 2019.
- [73] Dahlia Malkhi, Kartik Nayak, and Ling Ren. “Flexible Byzantine Fault Tolerance”. In: *CCS*. 2019.
- [74] J. Neu, E. Tas, and D. Tse. “Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma”. In: *SE&P*. 2021.
- [75] Joachim Neu, Ertem Nusret Tas, and David Tse. *The Availability-Accountability Dilemma and its Resolution via Accountability Gadgets*. 2021. URL: <http://arxiv.org/abs/2105.06075>.
- [76] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. “YOSO: You Only Speak Once”. In: *CRYPTO*. 2021.
- [77] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Michael Dahlin, and Taylor Riche. “Upright cluster services”. In: *SOSP*. 2009.
- [78] Emmanuelle Anceaume, Antonella Del Pozzo, Thibault Rieutord, and Sara Tucci-Piergiovanni. “On Finality in Blockchains”. In: *OPODIS*. 2022.
- [79] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. “HotStuff: BFT Consensus with Linearity and Responsiveness”. In: *PODC*. 2019.
- [80] Chrysoula Stathakopoulou, Matej Pavlovic, and Marko Vukolić. “State Machine Replication Scalability Made Simple”. In: *EuroSys*. 2022.
- [81] Gauthier Voron and Vincent Gramoli. *Dispel: Byzantine SMR with Distributed Pipelining*. 2019. URL: <http://arxiv.org/abs/1912.10367>.

- [82] Joseph Poon and Thaddeus Dryja. *The Bitcoin lightning network: Scalable off-chain instant payments*. 2016. URL: <https://www.bitcoinlightning.com/wp-content/uploads/2018/03/lightning-network-paper.pdf>.
- [83] Alejandro Ranchal-Pedrosa, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. “Scalable lightning factories for Bitcoin”. In: *ACM/SIGAPP SAC*. 2019.
- [84] Joseph Poon and Vitalik Buterin. *Plasma: Scalable autonomous smart contracts*. 2017. URL: <https://plasma.io/plasma.pdf>.
- [85] *The raiden network*. 2019. URL: <https://raiden.network>.
- [86] Christian Decker and Roger Wattenhofer. “A fast and scalable payment network with bitcoin duplex micropayment channels”. In: *Symposium on Self-Stabilizing Systems*. 2015.
- [87] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. *eltoo: A simple layer2 protocol for Bitcoin*. 2018. URL: <https://blockstream.com/eltoo.pdf>.
- [88] Daniel Robinson. *HTLCS-considered-harmful*. Stanford Blockchain Conference. 2019. URL: <http://j.mp/2m7BsKf>.
- [89] Akash Khosla, Evan Schwartz, and Adrian Hope-Bailie. *Interledger RFCs, 0018 DRAFT 3, Connector Risk Mitigations*. Github. 2019. URL: <http://j.mp/2m20vfP>.
- [90] Weizhao Tang, Weina Wang, Giulia Fanti, and Sewoong Oh. *Privacy-Utility Tradeoffs in Routing Cryptocurrency over Payment Channel Networks*. 2019. URL: <http://arxiv.org/abs/1909.02717>.
- [91] Jordi Herrera-Joancomartí, Guillermo Navarro-Arribas, Alejandro Ranchal-Pedrosa, Cristina Pérez-Solà, and Joaquin Garcia-Alfaro. “On the Difficulty of Hiding the Balance of Lightning Network Channels”. In: *Asia CCS*. 2019.
- [92] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. “Concurrency and privacy with payment-channel networks”. In: *CCS*. 2017.
- [93] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, and Matteo Maffei. “SilentWhispers: Enforcing Security and Privacy in Decentralized Credit Networks.” In: *NDSS*. 2017.
- [94] Stefanie Roos, Pedro Moreno-Sanchez, Aniket Kate, and Ian Goldberg. *Settling payments fast and private: Efficient decentralized routing for path-based transactions*. 2017. URL: <http://arxiv.org/abs/1709.05748>.
- [95] Ayelet Mizrahi and Aviv Zohar. “Congestion Attacks in Payment Channel Networks”. In: *Financial Cryptography and Data Security*. 2021.
- [96] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. *Enabling blockchain innovations with pegged sidechains*. 2014. URL: <https://blockstream.com/sidechains.pdf>.
- [97] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. “Proof-of-stake sidechains”. In: *SE&P*. 2019.
- [98] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. “RapidChain: Scaling Blockchain via Full Sharding”. In: *CCS*. 2018.

- [99] Ittay Eyal, Adem Efe Gencer, Emin Gun Sirer, and Robbert Van Renesse. “Bitcoin-NG: A Scalable Blockchain Protocol”. In: *NSDI*. 2016.
- [100] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. “A Secure Sharding Protocol For Open Blockchains”. In: *CCS*. 2016.
- [101] Rafael Pass and Elaine Shi. “Hybrid Consensus: Efficient Consensus in the Permissionless Model”. In: *DISC*. 2017.
- [102] Michael Ben-Or, Ran Canetti, and Oded Goldreich. “Asynchronous Secure Computation”. In: *Symposium on Theory of Computing*. 1993.
- [103] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. “Asynchronous Secure Computations with Optimal Resilience”. In: *PODC*. 1994.
- [104] Tier Nolan. *Atomic swaps using cut and choose*. 2016. URL: <https://bitcointalk.org/index.php?topic=1364951>.
- [105] Maurice Herlihy. “Atomic cross-chain swaps”. In: *PODC*. 2018.
- [106] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. “Xclaim: Trustless, interoperable, cryptocurrency-backed assets”. In: *S&P*. 2019.
- [107] Victor Zakhary, Divyakant Agrawal, and Amr El Abbadi. *Atomic Commitment Across Blockchains*. 2019. URL: <http://arxiv.org/abs/1905.02847>.
- [108] Rusty Russell. *Lightning Networks Part II: Hashed Timelock Contracts (HTLCs)*. 2017. URL: <https://rusty.ozlabs.org/?p=462>.
- [109] Maurice Herlihy, Barbara Liskov, and Liuba Shrira. “Cross-chain deals and adversarial commerce”. In: *The VLDB journal*. 2021.
- [110] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. “Eclipse Attacks on Bitcoin’s Peer-to-Peer Network.” In: *USENIX Security*. 2015.
- [111] Maria Apostolaki, Aviv Zohar, and Laurent Vanbever. “Hijacking Bitcoin: Routing Attacks on Cryptocurrencies”. In: *S&P*. 2017.
- [112] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. “Impact of Man-In-The-Middle Attacks on Ethereum”. In: *SRDS*. 2018.
- [113] Arthur Gervais, Hubert Ritzdorf, Ghassan O Karame, and Srdjan Capkun. “Tampering with the delivery of blocks and transactions in bitcoin”. In: *CCS*. 2015.
- [114] Vitalik Buterin. *The problem of censorship*. 2015. URL: <https://blog.ethereum.org/2015/06/06/the-problem-of-censorship/>.
- [115] Seth Gilbert and Nancy Lynch. “Perspectives on the CAP Theorem”. In: *Computer*. 2012.
- [116] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. “Optimal selfish mining strategies in bitcoin”. In: *Financial Cryptography and Data Security*. 2016.

- [117] Meni Rosenfeld. *Analysis of hashrate-based double spending*. 2014. URL: <http://arxiv.org/abs/1402.2009>.
- [118] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. “Stubborn mining: Generalizing selfish mining and combining with an eclipse attack”. In: *EuroS&P*. 2016.
- [119] Samiran Bag, Sushmita Ruj, and Kouichi Sakurai. “Bitcoin block withholding attack: Analysis and mitigation”. In: *Transactions on Information Forensics and Security*. 2016.
- [120] Joseph Bonneau. “Why Buy When You Can Rent? Bribery Attacks on Bitcoin-Style Consensus”. In: *Financial Cryptography and Data Security Workshops*. 2016.
- [121] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. “Double-spending fast payments in bitcoin”. In: *CCS*. 2012.
- [122] 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: *CCS*. 2016.
- [123] Christopher Natoli and Vincent Gramoli. “The Balance Attack or Why Forkable Blockchains are Ill-Suited for Consortium”. In: *DSN*. 2017.
- [124] Parinya Ekparinya, Vincent Gramoli, and Guillaume Jourjon. “The Attack of the Clones against Proof-of-Authority”. In: *NDSS*. 2022.
- [125] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. “Sync hotstuff: Simple and practical synchronous state machine replication”. In: *S&P*. 2020.
- [126] Jae Kwon. *Tendermint: Consensus without mining*. 2014. URL: https://cdn.relayto.com/media/files/LPgoW018TCeMIggJVakt_tendermint.pdf.
- [127] Ethan Buchman, Jae Kwon, and Zarko Milosevic. *The latest gossip on BFT consensus*. 2018. URL: <http://arxiv.org/abs/1807.04938>.
- [128] Ethan Buchman, Rachid Guerraoui, Jovan Komatovic, Zarko Milosevic, Dragos-Adrian Seredinschi, and Josef Widder. “Revisiting Tendermint: Design Tradeoffs, Accountability, and Practical Use”. In: *DSN*. 2022.
- [129] Dominik Harz, Lewis Gudgeon, Arthur Gervais, and William J. Knottenbelt. “Balance: Dynamic Adjustment of Cryptocurrency Deposits”. In: *CCS*. 2019.
- [130] Jinyuan Li, Maxwell Krohn, David Mazières, and Dennis Shasha. “Secure Untrusted Data Repository (SUNDR)”. In: *OSDI*. 2004.
- [131] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. “FairLedger: A Fair Blockchain Protocol for Financial Institutions”. In: *OPODIS*. 2019.
- [132] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. “BFT Protocol Forensics”. In: *CCS*. 2021.
- [133] Luciano Freitas de Souza, Petr Kuznetsov, Thibault Rieutord, and Sara Tucci Piergiovanni. “Brief Announcement: Accountability and Reconfiguration - Self-Healing Lattice Agreement”. In: *DISC*. 2021.

- [134] Vitalik Buterin and Virgil Griffith. *Casper the Friendly Finality Gadget*. 2019. URL: <http://arxiv.org/abs/1710.09437v4>.
- [135] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, et al. “{IA-CCF}: Individual Accountability for Permissioned Ledgers”. In: *NSDI*. 2022.
- [136] Elrond Team. *Highly Scalable Public Blockchain via Adaptive State Sharding and Secure Proof of Stake*. 2019. URL: <https://elrond.com/assets/files/elrond-whitepaper.pdf>.
- [137] David Galindo, Jia Liu, Mihair Ordean, and Jin-Mann Wong. “Fully Distributed Verifiable Random Functions and their Application to Decentralised Random Beacons”. In: *EuroSP*. 2021.
- [138] Adam Gagol, Damian Leundefinedniak, Damian Straszak, and Michał undefinedwiunde-
finedtek. “Aleph: Efficient Atomic Broadcast in Asynchronous Networks with Byzantine Nodes”. In: *Advances in Financial Technologies*. 2019.
- [139] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. “The Honey Badger of BFT Protocols”. In: *CCS*. 2016.
- [140] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. *Efficient Asynchronous Byzantine Agreement without Private Setups*. 2021. URL: <http://arxiv.org/abs/2106.07831>.
- [141] Luciano Freitas de Souza, Sara Tucci-Piergiovanni, Renaud Sirdey, Oana Stan, Nicolas Quero, and Petr Kuznetsov. *RandSolomon: optimally resilient multi-party random number generation protocol*. 2021. URL: <http://arxiv.org/abs/2109.04911>.
- [142] Timo Hanke, Mahnush Movahedi, and Dominic Williams. *Dfinity technology overview series, consensus system*. 2018. URL: <http://arxiv.org/abs/1805.04548>.
- [143] Ignacio Cascudo and Bernardo David. “SCRAPE: Scalable Randomness Attested by Public Entities”. In: *Applied Cryptography and Network Security*. 2017.
- [144] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar Weippl. *Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness*. 2021. URL: <https://eprints.cs.univie.ac.at/6629/1/2020-942.pdf>.
- [145] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. *RandPiper – Reconfiguration-Friendly Random Beacons with Quadratic Communication*. 2020. URL: <https://eprint.iacr.org/2020/1590>.
- [146] *RANDAO: A DAO working as RNG of Ethereum*. 2019. URL: <https://github.com/randao/randao>.
- [147] Gang Wang and Mark Nixon. “RandChain: Practical Scalable Decentralized Randomness Attested by Blockchain”. In: *International Conference on Blockchain*. 2020.
- [148] Elette Boyle, Shafi Goldwasser, and Yael Tauman Kalai. “Leakage-resilient coin tossing”. In: *Distributed Computing*. Springer, 2014.

- [149] Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. *CRAFT: Composable Randomness and Almost Fairness from Time*. 2020. URL: <https://eprint.iacr.org/2020/784>.
- [150] Ignacio Cascudo and Bernardo David. “ALBATROSS: Publicly Attestable BATCHed Randomness Based On Secret Sharing”. In: *ASIACRYPT*. 2020.
- [151] Marcos K Aguilera and Sam Toueg. “The correctness proof of Ben-Or’s randomized consensus algorithm”. In: *Distributed Computing*. Springer, 2012.
- [152] Ran Canetti and Tal Rabin. “Fast asynchronous Byzantine agreement with optimal resilience”. In: *Symposium on Theory of computing*. 1993.
- [153] Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K. Reiter, and Emin Gün Sirer. “Efficient Verifiable Secret Sharing with Share Recovery in BFT Protocols”. In: *CCS*. 2019.
- [154] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. *hbACSS: How to Robustly Share Many Secrets*. 2021. URL: <https://eprint.iacr.org/2021/159>.
- [155] Lorenz Breidenbach, Phil Daian, Florian Tramèr, and Ari Juels. “Enter the Hydra: Towards Principled Bug Bounties and Exploit-Resistant Smart Contracts”. In: *USENIX Security*. 2018.
- [156] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts”. In: *S&P*. 2016.
- [157] Yingjie Xue and Maurice Herlihy. “Hedging Against Sore Loser Attacks in Cross-Chain Transactions”. In: *PODC*. 2021.
- [158] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. “Reaching Consensus for Asynchronous Distributed Key Generation”. In: *PODC*. 2021.
- [159] Michael Backes and Christian Cachin. “Reliable Broadcast in a Computational Hybrid Model with Byzantine Faults, Crashes, and Recoveries.” In: *DSN*. 2003.
- [160] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. “Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited”. In: *PODC*. 2020.
- [161] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. “Secure and Efficient Asynchronous Broadcast Protocols”. In: *CRYPTO*. 2001.
- [162] Gabriel Bracha. “Asynchronous Byzantine agreement protocols”. In: *Information and Computation*. 1987.
- [163] Pierre Civid, Seth Gilbert, and Vincent Gramoli. “Brief Announcement: Polygraph: Accountable Byzantine Agreement”. In: *DISC*. 2020.
- [164] Sisi Duan, Michael K. Reiter, and Haibin Zhang. “BEAT: Asynchronous BFT Made Practical”. In: *CCS*. 2018.

- [165] Swan Dubois, Rachid Guerraoui, Petr Kuznetsov, Franck Petit, and Pierre Sens. “The Weakest Failure Detector for Eventual Consistency”. In: *PODC*. 2015.
- [166] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance and Proactive Recovery”. In: *ACM Transaction on Computer Systems*. 2002.
- [167] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. “Zyzyva: Speculative Byzantine Fault Tolerance”. In: *SOSP*. 2007.
- [168] Tuanir França Rezende and Pierre Sutra. “Leaderless State-Machine Replication: Specification, Properties, Limits”. In: *DISC*. 2020.
- [169] Önder Gürcan, Alejandro Ranchal-Pedrosa, and Sara Tucci-Piergiovanni. “On Cancellation of Transactions in Bitcoin-like Blockchains”. In: *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*. Springer. 2018.
- [170] Sarah Azouvi, Patrick McCorry, and Sarah Meiklejohn. *Winning the caucus race: Continuous leader election via public randomness*. 2018. URL: <http://arxiv.org/abs/1801.07965>.
- [171] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. *On bitcoin as a public randomness source*. 2021. URL: <https://eprint.iacr.org/2018/1239>.
- [172] Benoit Libert and Damien Vergnaud. “Unidirectional chosen-ciphertext secure proxy re-encryption”. In: *IEEE Transactions on Information Theory*. 2011.
- [173] Alexandre Ruiz and Jorge L Villar. “Publicly verifiable secret sharing from Paillier’s cryptosystem”. In: *Western European Workshop on Research in Cryptology*. 2005.
- [174] Joan Antoni Donet Donet, Cristina Pérez-Sola, and Jordi Herrera-Joancomartí. “The bitcoin P2P network”. In: *Financial Cryptography and Data Security*. 2014.
- [175] Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. *The Lightning Network Daemon*. 2018. URL: <https://github.com/lightningnetwork/lnd>.
- [176] Elements Project. *c-lightning – a Lightning Network implementation in C*. 2019. URL: <https://github.com/ElementsProject/lightning>.
- [177] ACINQ. *Eclair GitHub Repository*. 2008. URL: <https://github.com/ACINQ/eclair>.
- [178] Dan Boneh, Manu Drijvers, and Gregory Neven. “Compact multi-signatures for smaller blockchains”. In: *Asiacrypt*. 2018.
- [179] Oded Goldreich, Silvio Micali, and Avi Wigderson. “How to play any mental game”. In: *ACM Symposium on Theory of Computing*. 2019.
- [180] John Nash. “Equilibrium points in n-person games.” In: *Proc. National Academy of Sciences* 36. 1950.
- [181] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. “As easy as ABC: Optimal (A)ccountable (B)yzantine (C)onsensus is easy!” In: *IPDPS*. 2022.
- [182] Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. “On the (Limited) Power of Non-Equivocation”. In: *PODC*. 2012.

- [183] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. “State machine replication in the libra blockchain”. In: *The Libra Assn., Tech. Rep.* 2019.
- [184] Kim Potter Kihlstrom, Louise E. Moser, and P. M. Melliar-Smith. “Byzantine Fault Detectors for Solving Consensus”. In: *The Computer Journal*. 2003. URL: <https://doi.org/10.1093/comjnl/46.1.16>.
- [185] Dahlia Malkhi, Michael Merritt, and Ohad Rodeh. “Secure reliable multicast protocols in a WAN”. In: *IEEE International Conference on Distributed Computing Systems*. 1997.
- [186] Pierre Tholoniati and Vincent Gramoli. “Formal Verification of Blockchain Byzantine Fault Tolerance”. In: *FRIDA*. 2019.
- [187] Ethan Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. MS Thesis. 2016.
- [188] Daniel Collins, Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Yvonne Anne Pignolet, Dragos-Adrian Seredinschi, Andrei Tonkikh, and Athanasios Xydkis. “Online Payments by Merely Broadcasting Messages”. In: *DSN*. 2020.
- [189] Ethereum. *Ethereum 2.0 phase 0*. 2021. URL: <https://notes.ethereum.org/@djrtwo/Bkn3zpxB>.
- [190] Zachary Amsden et al. *The Libra Blockchain*. Revised version of September 25. 2019.
- [191] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. “Dumbo: Faster Asynchronous BFT Protocols”. In: *CCS*. 2020.
- [192] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. *HotStuff: BFT Consensus in the Lens of Blockchain*. Version 6 (accessed 21 May 2019). 2019. URL: <http://arxiv.org/abs/1803.05069>.
- [193] Amarnath Mukherjee. *On the dynamics and significance of low frequency components of Internet load*. 1992.
- [194] Mark E Crovella and Robert L Carter. “Dynamic server selection in the Internet”. In: *HPCS*. 1995.
- [195] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short signatures from the Weil pairing”. In: *Journal of cryptology*. Springer, 2004.
- [196] A Yao. “Theory and applications of trapdoor functions.” In: *Symposium on Foundations of Computer Science*. 1982.
- [197] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.
- [198] Adi Shamir. “How to Share a Secret”. In: *Communications of the ACM*. 1979.
- [199] David Chaum and Torben Pryds Pedersen. “Wallet Databases with Observers”. In: *CRYPTO*. 1993.
- [200] Amos Fiat and Adi Shamir. “How To Prove Yourself: Practical Solutions to Identification and Signature Problems”. In: *CRYPTO*. 1987.

- [201] David Pointcheval and Jacques Stern. “Security Proofs for Signature Schemes”. In: *EUROCRYPT*. 1996.
- [202] Irving S Reed and Gustave Solomon. “Polynomial codes over certain finite fields”. In: *Journal of the society for industrial and applied mathematics*. 1960.
- [203] Robert J. McEliece and Dilip V. Sarwate. “On sharing secrets and Reed-Solomon codes”. In: *Communications of the ACM*. 1981.
- [204] Thanh Nguyen-Van, Tuan Nguyen-Anh, Tien-Dat Le, Minh-Phuoc Nguyen-Ho, Tuong Nguyen-Van, Nhat-Quang Le, and Khuong Nguyen-An. “Scalable distributed random number generation based on homomorphic encryption”. In: *International Conference on Blockchain*. 2019.
- [205] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. “Proofs-of-delay and randomness beacons in ethereum”. In: *SECP*. 2017.
- [206] Mayank Raikwar. “Competitive Decentralized Randomness Beacon Protocols”. In: *International Symposium on Blockchain and Secure Critical Infrastructure*. 2022.
- [207] Marshall Pease, Robert Shostak, and Leslie Lamport. “Reaching Agreement in the Presence of Faults”. In: *Journal of the ACM*. 1980.
- [208] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. “Byzantine Ordered Consensus without Byzantine Oligarchy”. In: *OSDI*. 2020.
- [209] Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. *Lightning Network In-Progress Specifications. BOLT 4: Onion Routing Protocol*. 2018. URL: "<https://github.com/lightningnetwork/lightning-rfc>".
- [210] Andrew Samokhvalov, Joseph Poon, and Olaoluwa Osuntokun. *Basis of lightning technology (BOLTs)*. 2018. URL: <https://github.com/lightningnetwork/lightning-rfc>.

Appendix A

Lockdown Attack

We illustrate in this chapter the Lockdown attack in detail.

A.1 Attack design

The atomicity needed in a multihop payment enforces that the intermediate payments in a multihop route should be held until the complete route is constructed and all payments can be performed together. During the time the route is being constructed, nodes in the route lock the balance of the payment until such payment takes place. With this underlying mechanism, a malicious user can lock a total amount p of balance in a channel AB_i , during the time a payment is being constructed, by sending a payment of value p through that channel AB_i . However, such action, that we label as a naive attack, has two main drawbacks from an adversarial point of view. The first one is related to the cost of the attack and the second one is related to the time the balance is locked.

Regarding the cost, in a naive attack, to block balance p in channel AB_i , the adversary needs to perform a payment of value p so the adversary needs to hold the same capacity that the attack is locking. In that sense, we can define the Attack Effort Ratio.

Definition A.1.1 (Attack Effort Ratio). The *Attack Effort Ratio* (AER) is the ratio between the capacity needed to perform the attack and the capacity that the attack blocks, i.e.,

$$AER = \frac{C_{attack}}{C_{blocked}}$$

The naive attack achieves $AER = 1$ and can be considered a brute force attack, since it always can be performed by design of multihop payments. Notice that AER measures the profitability of the attack, and if the adversary can reduce the AER then the attack becomes more profitable for the adversary, meaning that the adversary is more incentivized to perform such attack.

Regarding the time during which the balance is locked, in a naive attack the adversary only locks the balance during the time the whole payment is being constructed and, in regular conditions, this period is often very short, since the final receiver of the payment in a multihop route “executes” the payment as soon as the payment arrives. For more powerful attacks we can define the χ function.

Definition A.1.2. The $\chi(t)$ **function** is a time based decreasing function that measures the total capacity blocked w.r.t. the time during which the attack has been conducted. The block generation time, t , is used as the time unit for this function.

For instance, $\chi(0) = C_{blocked}$ as it provides the total capacity blocked at the initial time of the attack. Eventually, $\chi(t) = 0$ for a large t , given that the blocking efficiency of the attack decreases over time. In a naive attack, $\chi(1) = 0$ because the capacity is unblocked almost instantly after the payment, long before the appearance of the first block ($t = 1$) after the attack execution.

As we will detail later, an attack is performed through multiple payments. For that reason, the $\chi(t)$ function is computed taking into account the expiration values of each payment that forms the attack. If we define $\chi_i(t)$ as the capacity blocked by payment i during t blocks, then $\chi(t) = \sum_i \chi_i(t), \forall i \in attack$.

For comparison purposes, we define two single value metrics that compress the $\chi(t)$ function: the Total Blocked Time and the Normalized Total Blocked Time.

The **Total Blocked Time** (TBT) of the attack is the sum of the $\chi(t)$ values:

$$TBT = \sum_{t=0}^{\infty} \chi(t)$$

The **normalized TBT** (\widetilde{TBT}) is defined as:

$$\widetilde{TBT} = \frac{TBT}{C_{blocked} \cdot \max\{T_{max}\}},$$

where $\max\{T_{max}\}$ is the maximum default value of T_{max} seen in any implementation. Therefore, $0 < \widetilde{TBT} \leq 1$, and the ideal attack with $\widetilde{TBT} = 1$ would be blocking $C_{blocked}$ capacity during 5000 blocks, that is, more than 34 days.

Once we have described how to perform a naive attack to a single channel, we now describe how the adversary improves the effectiveness of the attack, both minimizing AER and maximizing TBT . We focus the attack goal to block the victim A as a middle node in multipath payments. In that case, the value $C_{blocked}$ is the total capacity of node A in the LN, that is $C_{blocked} = C_A = \sum C_{AB_i}$. Notice that regarding the attack goal, in order to block a middle node in a payment route it suffices to block all incoming balances to A or all outgoing balances from A . In either of these cases, A cannot route any payment. Then, the naive attack over a single node A achieves $C_{attack} = \min\{\sum_{i=1}^n balance_{AB_i}, \sum_{i=1}^n balance_{B_iA}\}$. Clearly, $C_{attack} \leq C_A$. The AER for such an attack is reduced with respect to the naive attack of a single channel. Notice that, with this approach, the AER reduction cannot be determined by the adversary, i.e., the adversary cannot directly control the balances. However, the AER can also be reduced when the same payment is used more than once to block different channels. In fact, in a multihop payment, a single payment p blocks up to $m \cdot p$ capacity being m the number of hops of the payment route. Another strategy to reduce AER is to construct the largest possible route. However, if the attack is focused on a victim A , not only has the length of the route to be computed, but it also should be kept close to A in order to ensure all blocking capacity obtained for

that route is able to block channels belonging to A . As we will see, the best strategy to keep the payment route close to the victim is to perform routes through A with loops as short as possible that return to A . This possibility will depend on the topology of the payment network in which A is connected.

The improvement of the attack can also be measured regarding the time during which the attack takes place. The objective is to maximize TBT . To this end, the adversary can be placed at the end of the route, to hold the payment as much time as possible before the funds of the route are unlocked. As we will see in detail in the next section, this strategy increases the capacity C_{attack} needed for the adversary.

A.1.1 Adversarial knowledge

To accurately perform the Lockdown attack, the adversary needs precise knowledge of the network. To construct payments routes which pass through victim A , the adversary needs to know the topology of the network to construct such paths. This information is available using any LN implementation, since it is needed to perform standard payments. Additionally to the topology of the network, the detailed information about balances of every channel are needed to perform the attack. This information can be derived from existing attacks in the literature [91].

Furthermore, to minimize AER , the number of hops of a payment route has to be maximized. Although payment routes in the LN are bounded to 20 hops [209], the exact number of hops that a route may contain is also limited by values T_{max} and δ of each node of that route. Notice that a node does not accept a payment that locks its funds more than T_{max} time, and is fixed by the adversary but decreased in each hop by the δ of each node. Then, depending on T_{max} and δ at each node of the route, the total number of hops in a route could be lower than 20. For that reason, the adversary also needs to know the values T_{max} and δ of each node of the network.

A.1.2 AER minimization

The Lockdown attack can be improved in terms of AER . For instance, regarding the *shorter loop* case example (Figure 3.3a), the AER depends on the difference between $o_1 + o_3$ and $o_2 + o_4$. In the extreme case in which $o_1 + o_3 = o_2 + o_4$ such attack has the worst possible AER , as $C_A = (o_1 + o_3) + (o_2 + o_4) = 2(o_1 + o_3)$ and $C_{attack} = C_{MA} = 2 \cdot (o_1 + o_3)$, so $AER = 1$. However, the adversary can reduce this value by relooping the nearer part of each route next to A . Then, if each payment route contains m hops, each original path can be transformed into $\mathcal{M} \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow \dots \rightarrow \mathcal{M}$ and $\mathcal{M} \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow \dots \rightarrow \mathcal{M}$. With those loops, the total amount that has to be routed is reduced to $\frac{2o_1}{m-2}$ and $\frac{2o_3}{m-2}$ respectively, so $C_{attack} = \frac{4(o_1+o_3)}{m-2}$ and $AER = \frac{2}{m-2}$. Notice this relooping strategy can also be implemented in the *longer loop* scenario (Figure 3.3b).

A.1.3 TBT maximization

Recall that the adversary should maximize TBT to make the attack more effective. To this end, the adversary takes advantage of being at the beginning and end of each payment.

As a first node, the adversary can determine the maximum χ_i for a particular payment i , since such value depends on values δ and T_{max} of each node and the node position in the route. The first one, T_{max} , is the maximum amount that a node allows an outgoing payment in a channel to be locked. The second value, δ , indicates the difference, in blocks, that each hop in the route requires. When a node receives a payment, he sets an expiration time¹, θ , for the payment, and subtracts his δ . If the resulting value is lower than his T_{max} , then he will keep forwarding the payment. Otherwise, the node will refuse the payment, the route will be discarded and the payer will need to find another route. Then, the best strategy for an adversary to maximize χ_i is to simulate the route assuming that each node, instead of discarding the payment, will set the new θ as his T_{max} (see Section A.3 for a detailed example).

As a last node of the payment, the adversary can hold the payment during the received $\theta = \chi_i$ value, being sure that the previous node does not cancel the payment before that time — it fits the proper waiting values of the implementation.

A.2 Experimental results

To analyze the feasibility of the proposed attack and provide a proof-of-concept, we need to ensure that nodes in the LN behave in a particular way. Firstly, to minimize *AER* we test if the type of routes with cycles used in our attack can be routed through the nodes of the LN. Secondly, to maximize *TBT*, we verify if the recipient of a multihop route is able to retain a payment during a certain period of time before the payment is finally processed locking channels involved in the payment route. Furthermore, we are also interested in implementing a mechanism for which the recipient can cancel the payment without paying any fee to the routing nodes.

We perform a test in a simnet controlled environment to validate that our claims are correct and that the routes generated in our attack containing loops can effectively be deployed in the three most relevant available implementations of the Bitcoin LN, namely lnd, c-lightning and eclair. Results can be found in Section A.3.

Once the feasibility of the attack has been proven from an implementation point of view, we performed some attack simulations for the LN of the Bitcoin mainnet in order to measure the *AER* of the attack, the function $\chi(t)$, and its economic cost. Notice that there is no technical reason that stops us from effectively executing the simulated attacks in the Bitcoin mainnet. However, for ethical reasons, we have not performed the attack on the mainnet and, instead, we have performed a responsible disclosure to the developers of the LN implementations.

Our simulations will assess the effectiveness of the attack given the actual topology of the network. We base our simulations on the attack algorithm described in Section 3.1.2.1, but, in order to provide accurate results, we have taken into account different restrictions that actual LN implementations take over their parameters.

Firstly, we bound to 20 the maximum hops that a payment route may have in the LN [210]. Regarding the routes' length, we assume that the expiration time for a route θ at each hop

¹For simplicity, we assume θ as a relative blockheight value.

cannot be lower than zero.

Secondly, all existing LN implementations fix the maximum value of a channel at 16777215 satoshis². This value may impact the channel that \mathcal{M} has to open with the victim A . Since such payment channel needs to have a total capacity of C_{attack} , if $C_{attack} > 16777215$ then \mathcal{M} needs to open more than one channel with A .

Once these values have been taken into account, to perform our simulations, we take a snapshot of the topology of the LN³ of the bitcoin mainnet on July, 9th, 2019 at 12:00.

A.2.1 Simulation assumptions

To execute the attack algorithm described in Section 3.1.2.1, the adversary needs to complement the information of the network graph with further data. The information needed is: the balance of each channel and the values T_{max} for each node of the network.

Regarding the balances, they can be obtained from previous attacks [91]. However, instead of performing the attack, we have assigned the balances of each channel using different statistical distributions, trying to reproduce the different scenarios that could be found in the network. In order to assign balances to channels, we proceed in the following way: for each channel, first the balance of one of the nodes is randomly selected using one of the selected distributions, and taking the capacity of the channel as the maximum possible value to generate. Then, the balance of the other node in the channel is set as the remaining balance (that is, the capacity minus the balance). Five different distributions are used to assign balances to channels: *deterministic*, *uniform*, *normal*, *exponential*, and *beta*. The *deterministic* distribution always assigns half of the capacity of the channel to each of the nodes; the *normal* distribution is used with $\mu = 0.5$ and $\sigma = 0.2$; the *exponential* distribution uses $\lambda = 1$; and the *beta* distribution $\alpha = \beta = 0.25$.

The value T_{max} is a network node parameter that is not publicly available, as it is not advertised by the nodes. However, this value is implementation-dependant⁴. Hence, by inferring the LN implementation of each node, we can obtain the values of T_{max} for that node. To infer the LN client implementation run by each node, we take into account the fee rate, the fee base rate, and the δ values announced in the nodes' channels policies. As shown in Table A.2.1, default values for the fee rate and δ parameters allow to uniquely identify the LN implementation. We use those values to infer node implementation. Moreover, the default value for the fee base rate is always 1000. We use this third value to further validate that the node is using default values in its policies.

However, users may indeed change channel policies, or even use different policies for different channels. On the one hand, if a node is not announcing any policy with the fee rate, fee base rate, and δ values corresponding to any of the described implementations, we assume the

²This bound is just an implementation parameter. There are already channels in the LN with larger values. The availability of larger channels reduces the number of channels for the attack, as well as total fees to pay for every open channel and the total cost of the attack.

³This information can be obtained, for instance, with the instruction `describegraph` of the `lnd` implementation.

⁴One may assume users changing some LN implementation parameters. However, T_{max} is hardcoded in each LN implementation.

	lnd (old)	lnd (new)	c-lightning	eclair
Fee rate	1	1	10	100
Fee base rate	1000	1000	1000	1000
δ	144	40	14	144

Table A.2.1: Parameters that help infer the Lightning Network implementation of a node.

implementation of that node is unknown. On the other hand, whenever a node announces different policies in its channels but only one of them corresponds to a default behavior, the node is tagged with this implementation. Finally, if multiple policies are announced and multiple default policies are identified, then again the node is tagged as unknown.

Taking this approach, using the selected snapshot of the network, we end up with a small percentage of unknown nodes (11.3%), for which we are not able to properly infer the implementation. In that case, we randomly tag those nodes with one of the three main implementations, with the same percentage distribution than those nodes already tagged. Using this approach, network nodes have been classified as shown in Table A.2.2.

	nodes (number)	(percentage)
lnd	2196	91.04%
c-lightning	183	7.59%
eclair	33	1.37%
<i>Total</i>	2412	100%

Table A.2.2: Number of nodes, with at least one channel, classified in one of the main implementations for the snapshot graph used in our analysis.

A.2.2 Attack simulation results

To perform the simulation, the attack focuses on one of the most relevant nodes in the network. This node has 600 open active channels with a total capacity slightly above 43 BTC. Then, we test the effectiveness of the attack considering that this node runs one of the three main implementations, lnd, c-lightning or eclair. For each implementation, we also test each of the balance distributions. In order to present more representative results, being the balance distribution a probability distribution, we execute the experiments 10 times and take the mean values.

For each implementation and for each balance distribution, we performed the attack and measured the *AER* of the attack, the percentage of the capacity of the victim that has been blocked, the total channels needed to perform the attack, and the normalized Total Blocked Time, \widetilde{TBT} (cf. Table 3.1). Furthermore, we also have analyzed the χ function of the attack (see Figure A.2.1).

Table 3.1 shows that the attack is effective in all scenarios (implementations and balance

distribution), because the AER is lower than 2 which is the value for a naive attack. Notice that in the worst attack, for a Beta distribution in which the node runs an eclair implementation, the AER is 0.584, which is half of the capacity of the victim to block 86.30% of its capacity. In fact, the percentage of the victim's capacity blocked is large for all the scenarios, never below the 80%. Moreover, the \widetilde{TBT} also shows that lnd implementations are the ones allowing the adversary to block more capacity over time (as can also be observed in Figure A.2.1).

Figure A.2.1 plots the χ function which shows the amount of time locking the funds. As expected, our results show that the value of T_{max} of each implementation determines the time that the attack lasts for. When the victim runs an lnd implementation, 80% of the capacity of the victim can be locked during 287 blocks (almost two days) in any balance distribution tested. But if we look at the 50%, this value is increased to 2407 blocks (more than 16 days). Even for the eclair implementation with the lowest T_{max} value of all three implementations ($T_{max} = 1008$), 50% of the capacity can be blocked during 287 blocks (almost two days) for all tested balance distributions.

Besides the effectiveness of the attack showed so far, we also measure the economic cost of the attack. For this measure, we take the same methodology as that of our previous work [91], in which the total cost of the attack can be divided between the entrance barrier cost and the economic cost. On the one hand, the entrance barrier cost takes into account the economic resources that the adversary has to control to be able to perform the attack. These resources will be completely recovered after the attack has been finished. On the other hand, the economic cost of the attack is the amount of money that the adversary will lose due to the execution of the attack.

Regarding **the entrance barrier cost**, the proposed attack needs to fund one or multiple LN channels with the capacity C_{attack} . Such amount is represented by the channels needed value of Table 3.1. For instance, the attack for the uniform distribution over c-lightning has an entrance barrier cost of 46 channels (or 7.71753546 BTC) to block 84.25% of the capacity of the node.

As for the **economic cost** of the attack, two values have to be taken into account: (i) the fee corresponding to the funding transaction of the channel; (ii) the fee corresponding to the transaction that closes the channel. Regarding the fees of the funding transactions, the cost depends on the number of channels needed to perform the attacks. The cost in fees for each channel depends on the size in bytes of the funding transaction. However, the size mostly depends on its inputs that will vary for each particular transaction, but a funding transaction with a single input can cost as low as 0.00001527 BTC⁵. Secondly, and regarding the closing transaction, it is also difficult to estimate the exact fee for a generic closing transaction, since again multiple parameters may affect this value. A cost rounding 0.00000909 BTC can be

⁵See, for instance, transaction:

[11b68b276453ac54c23ee49186df78d9895fbfd47071ced6371364abdddcfc6f](#). It is the funding transaction corresponding to the Channel Id [645513381196136448](#) opened on July 26, 2019, by node [021607cfce19a4c5e7e6e738663dfafbbac262e4ff76c2c9b30dbeefc35c00643](#)

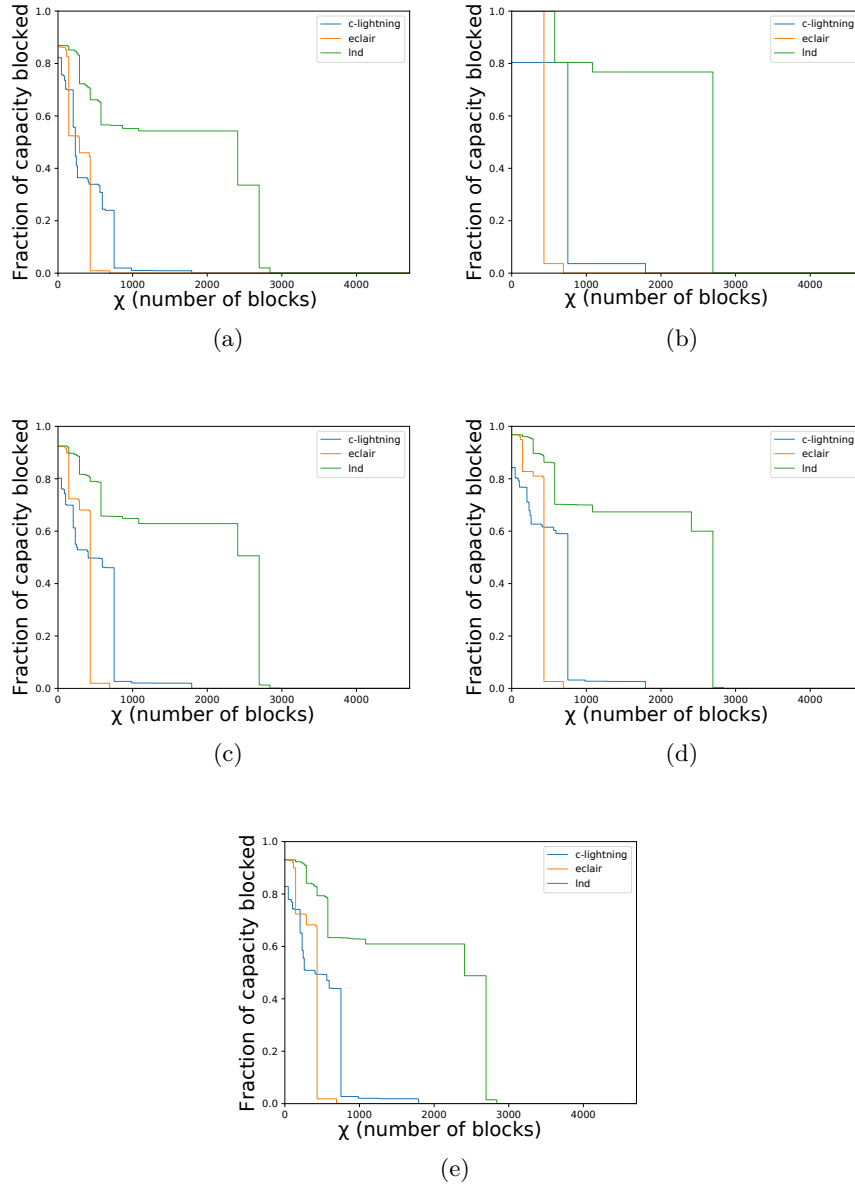


Figure A.2.1: Number of blocks for which the Lockdown attack locks balances, i.e. $\chi(t)$ function results, for every tested distribution: (a) Beta, (b) Deterministic, (c) Exponential, (d) Normal, (e) Uniform.

achieved, as can be seen in different existing closing transactions⁶.

Notice that we have not included the Lightning fees as a cost because they are never applied (the payments never succeed). For that reason, the total number of payments needed to perform the attack does not affect the economic cost of the attack.

With these values, we can estimate the economic cost of an attack. For instance, an attack based on a normal distribution assuming an lnd node blocks the 96.79% of the capacity of the node with 63 channels, that means 0.00153468 BTC in fees for opening and closing the channels, or around 15 Euros.

A.3 Simnet network

To perform our experiments, we create a Lightning simnet network with eleven nodes, $\mathcal{M}, A, B_1, \dots, B_9$. Node \mathcal{M} will be the adversary and A the victim. Nodes B_1, \dots, B_9 will represent victim's neighbors. To test all implementations in our simnet, we run different implementations for different nodes. More precisely, the following configuration has been taken. Nodes $\mathcal{M}, A, B_1, B_2, B_3$ run the lnd implementation with version 0.5.2-99-beta, nodes B_4, B_5, B_6 the c-lightning with version v0.7.0 and nodes B_7, B_8, B_9 run eclair with version *version=0.2-SNAPSHOT*. Over this configuration, we have created 10 payment channels, as shown in Figure A.3.2a.

With this settlement, \mathcal{M} performs a payment to himself, following the route $\mathcal{M} \rightarrow A \rightarrow B_1 \rightarrow A \rightarrow B_2 \rightarrow A \rightarrow B_3 \rightarrow A \rightarrow B_4 \rightarrow A \rightarrow B_5 \rightarrow A \rightarrow B_6 \rightarrow A \rightarrow B_7 \rightarrow A \rightarrow B_7 \rightarrow A \rightarrow B_9 \rightarrow A \rightarrow \mathcal{M}$.

The correct execution of the experiment proves that the payment has been processed by all nodes and that routes can effectively contain loops. Notice that the loops tested in this experiment are the shortest possible, which validates the *shorter loop* case of our attack (see Section 3.1.2.1). Also, the implementation selected for each node ensures that this behavior is equivalent in all implementations.

Figure A.3.2b shows a new scenario where we have added a payment channel between nodes B_6 and B_9 . With this scenario, \mathcal{M} performs a payment to himself, following the route $\mathcal{M} - A - B_6 - B_9 - A - B_6 - B_9 - A - \mathcal{M}$.

Again, the test shows that the payment is correctly processed by all nodes and it proves that all implementations can also accept the *longer loop* case, because we have chosen A, B_6 and B_9 all with different implementations.

Once we have ensured that routes with cycles are possible to execute in any implementation, we would like to study how to maximize χ_p , the time for which a recipient can lock a payment p . This value can be estimated using information of the nodes that are included in a route. More precisely, values δ and T_{max} of each node and the node position in the route determines the maximum time a payment can be locked for.

In our scenario, the adversary controls both the first and the last node of the route. We first describe how, as a first node, the adversary can determine the maximum χ_p for a particular

⁶For instance, Channel Id [624629257244573696](#) with total capacity 0.05 BTC has been closed with the following close transaction [362235c844533ff7ae0e2fca078b956e82093b92f86010bed51e990d52af6679](#)

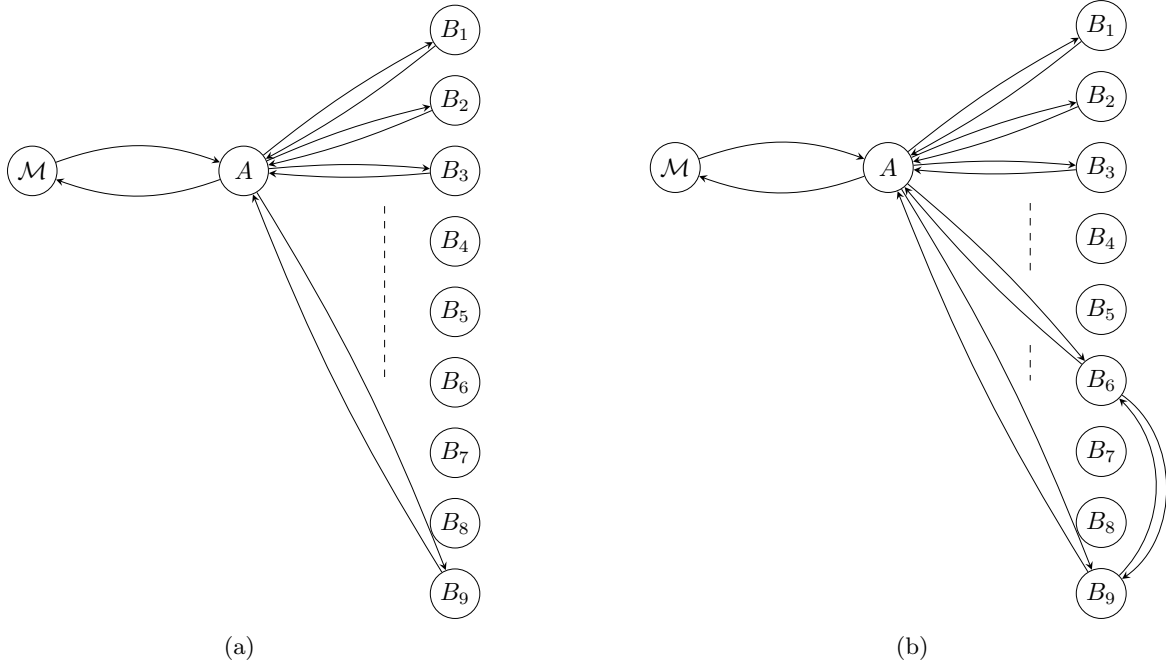


Figure A.3.2: Simnet scenarios maximizing A 's balance to lock in one payment of the attacker, by routing through B_i (a) and allowing loops (b).

route. Then, we will detail how the adversary, as the last node of the route, may block the payment during χ_p and how, after that time, he can cancel the payment without paying any fee to the routing nodes and, furthermore, restating all channels to the initial state to perform a second attack.

As pointed out in Section 2.1, the parameters that determine the actions of each node of the route are T_{max} and δ . T_{max} is the maximum amount that a node allows an outgoing payment in a channel to be locked for, while δ indicates the difference, in blocks, that each hop in the route requires. These parameters are different for each implementation, as Table A.3.3 shows. When a node receives a payment, he sets an expiration time⁷, θ , for the payment, and subtracts his δ . If the resulting value is lower than his T_{max} , then he will keep forwarding the payment, otherwise, the node will refuse the payment, the route will be discarded and the payer will need to find another route. Then, the best strategy for an adversary to maximize χ_p is to simulate the route assuming that each node, instead of discarding the payment, will set T_{max} to this new θ . For instance, suppose the following route $\mathcal{M} - B_i - B_j - B_k - \mathcal{M}$ and assume that B_i runs an lnd node, B_j runs an eclair node and B_k runs a c-lightning node. Assuming the default values of Table A.3.3, the simulation performed by \mathcal{M} will start with $\theta = \infty$. When processing the first hop, B_i runs an lnd node which translates into $T_{max} = 5000$ and $\delta = 144$, meaning that, for that hop, we can compute $\theta = 5000 - 144 = 4856$. In the next hop, B_j runs an eclair implementation, hence $T_{max} = 1008$ and $\delta = 144$. In that case, since the received $\theta = 4856$ is greater than 1008, we will set $\theta = 1008 - 144 = 864$. Then B_k runs a c-lightning

⁷Although the θ is an absolute blockheight value, here we will refer as a relative value to simplify the explanation.

with $T_{max} = 2016$ and $\delta = 14$ and the received $\theta = 864$ is lower than 2016, we can calculate $\theta = 864 - 14 = 850$. As this is the last hop, $\theta = 850$ is the time during which the channel can be blocked. With this procedure, \mathcal{M} can compute the optimal θ value that he will include in the first hop to maximize χ_p . In that case $\theta = 850 + 14 + 144 + 144 = 1152$ will provide a maximum χ_p .

	lnd	c-lightning	eclair
δ	144	14	144
T_{max}	5000	2016	1008

Table A.3.3: Default parameters for different implementations.

A.4 Countermeasures to handle the attack

The main countermeasures focus on increasing the *AER* in order to make the attack less profitable. As discussed in Section 3.1.2.1, *AER* is reduced thanks to the possibility that a single payment performs a route with multiple hops. Furthermore, if the adversary may maintain the route near the victim, the *AER* is even further diminished. As a result, different countermeasures can be adopted.

First of all, loops in a payment route should be minimized or forbidden. In particular, cycles of length two (the ones of the form $A \rightarrow B \rightarrow A$) should be completely forbidden, since these minimize *AER* and keep the route close to a potential victim. We argue that imposing this restriction does not damage any possible functionality of the LN. Notice that lightning payments, even those in a multihop route, are designed to be performed atomically in the sense that they are executed completely or not executed at all. A payment with a subpath of the form $A \rightarrow B \rightarrow A$, once executed, leaves the state between A and B exactly as it was previous to the payment. The implementation of this measure is straightforward even assuming that routing in the LN is performed through onion routing. Note that in the onion routing approach, every node is aware of the previous and next node in the route, meaning they can reject a route if both nodes are the same.

Regarding cycles of length greater than 2, it is clear that its restriction also increases *AER* and hinders the attack. Again, although the LN currently routes using onion packets and nodes are only aware of the previous and next hop in the route, additional information transferred between routes and shared by all nodes, such as the hash used in the HTLC, can be used to detect that a cycle is passing through a node and reject cycles. However, in contrast with cycles of length two, longer cycles do not keep the same state of the channel and it can be used for legitimate purposes, like spontaneous payments⁸, whose restrictions could impact future LN features.

⁸SPSP, Simple Protocol for Spontaneous Payments, <https://lists.linuxfoundation.org/pipermail/lightning-dev/2018-June/001327.html>

Besides cycles, increasing the length of a payment route also reduces *AER*. As such, a possible countermeasure for the proposed attack is the reduction of the maximum length of a route. This value is set to 20 in the LN specification and it could be reduced to increase the *AER* of an attack. However, this directly impacts in the performance of the network, given that its reduction could potentially discard possible routes for legitimate payments. More testing should be performed before implementing this type of countermeasure.

Another straightforward countermeasure that can be performed to reduce the effectiveness of the attack is the fine-tuning of some lightning parameters that, until now, have not been properly addressed. These parameters are T_{max} and δ , which have two different implications for our attack. In spite of the maximum hop value (set to 20), T_{max} and δ can effectively determine a lower bound for the number of hops in a route.⁹ Given that reducing the maximum number of loops increases *AER*, setting proper values could potentially hinder attacks. On the other hand, the time value during which a channel or victim can be blocked without the adversary needing to perform any action is also dependent on those two parameters. As a result, reducing the actual values of T_{max} and δ is a countermeasure for our attack, since it reduces the time during which the adversary may lock the funds. However, assessing the correct values for T_{max} and δ deserves a detailed and exhaustive analysis and testing.

⁹For instance, a payment route in which all nodes run an eclair implementation can be of at most 7 hops.

Appendix B

TRAP protocol: discussions

In this chapter, we revisit and extend Figure 4.1 after explaining the TRAP protocol. Then, we discuss the expected behavior of a coalition provided the correctness of the TRAP protocol.

B.1 Extended example figure

Figure B.1.1 depicts a slightly extended version of the execution example of Figure 4.1. Similarly to Figure 4.1, the execution starts with $k + t$ Byzantine and rational players causing a disagreement on predecisions. However, now we detail further how the e baiters prevent termination of the BFTCR protocol. In particular, by not committing to a value in the first reliable broadcast of BFTCR, the e baiters can prevent players in A and in B from terminating in any of the two partitions. Thus, the e baiting players wait till they receive certificates from players in A and in B in order to construct PoFs. Then, they wait till they deliver enough values from the second group reliable broadcasts from players in partitions A and B that guarantee that no other Byzantine or rational player can become a valid candidate once they reveal that they are baiting (as we showed in the proof of Lemma 4.4). At this point, the e players reveal their PoFs by sending the decryption key to their commitment. Then, players in A and B can resolve their disagreement on predecisions, choose a winner of the reward from among the e valid candidates at random, and punish the rest of deviating players.

B.2 Paying a reward at no cost to non-deviants

One might think that implementing a baiting strategy with a reward and deposits might not be enough: we need to discourage coalitions from actually playing the baiting strategy, since the system would have to pay the reward \mathfrak{R} , and thus the coalition can effectively steal some funds from the system. However, if the system can use the deposited amount \mathfrak{L} from at least t_ℓ certified fraudsters in the coalition to pay for the baiting reward \mathfrak{R} , then the system does not lose any funds (lossfree reward), while obtaining agreement (baiting agreement).

Furthermore, notice that if the coalition consists entirely of rational players then they do not actually play this strong baiting strategy since, by the definition of strong baiting strategy, they all individually lose more than they can gain from deviating. Even if the presence of

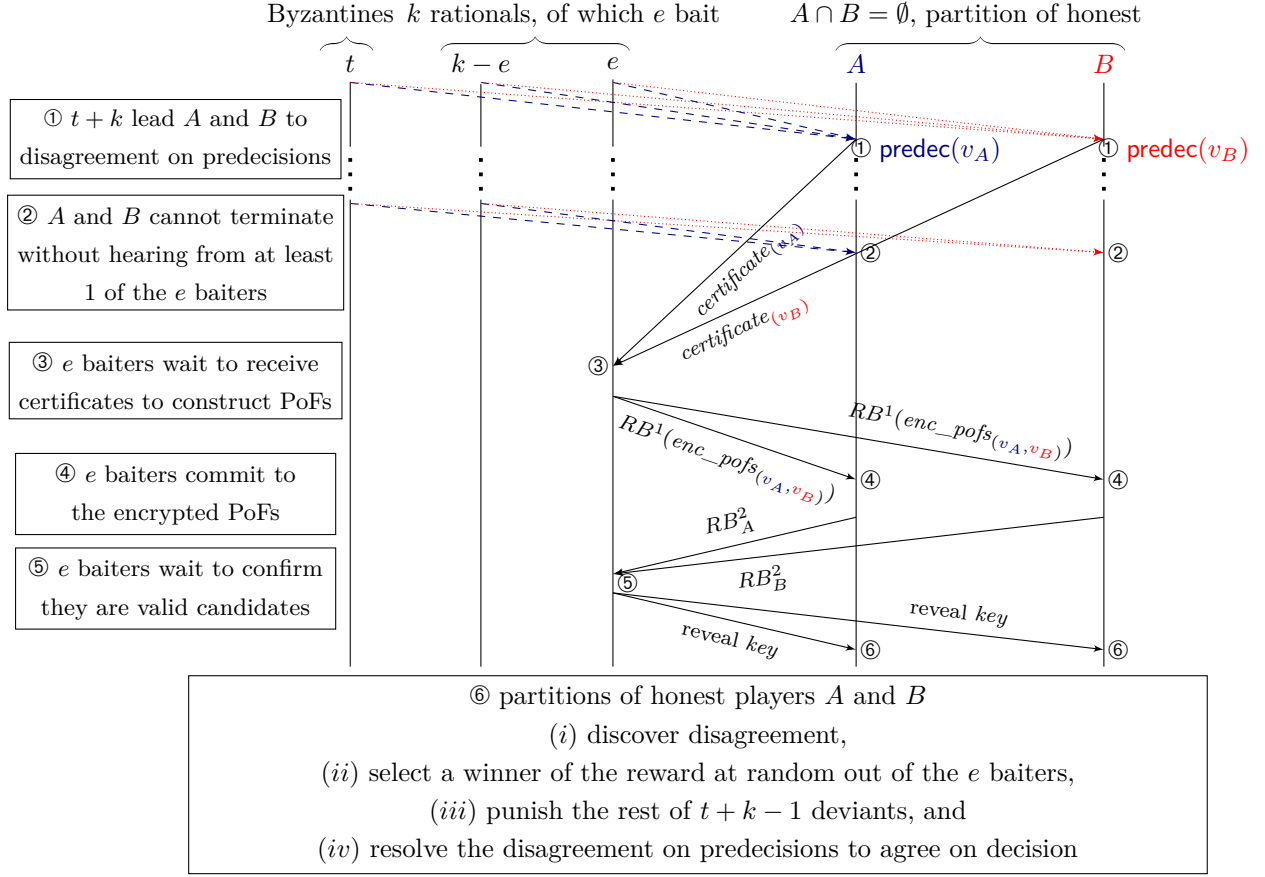


Figure B.1.1: Extended example execution of the TRAP protocol. First, ① all t Byzantine and k rational players collude to cause a disagreement on the output of the accountable consensus protocol, resulting in A and B predeciding different outputs. Then, ② e of the k rational players decide to bait while executing the BFTCR protocol, preventing A and B from deciding their disagreeing predecisions. As such, ③ the e baiters wait until they receive proof of the disagreement on predecisions, to then ④ commit to the encrypted PoFs. Finally, ⑤ once they deliver as many second reliable broadcast from A and B as possible confirming that honest players delivered their PoFs encrypted commitment, then ⑥ the e baiters prove the disagreement revealing the proofs-of-fraud in the BFTCR protocol. Hence, neither A nor B decide their conflicting predecisions, but instead reward one of the e baiters, punish the rest of $t + k - 1$ players responsible for the disagreement on predecisions, and resolve the disagreement, deciding one of v_A or v_B , or, depending on the application, merging both.

Byzantine players leads to a baiter being paid, agreement will still be guaranteed at no cost to non-deviating players. This leaves the open question of how likely it is that Byzantine players with unexpected utilities but possibly with the goal to break the system would be interested in giving their funds for free to rational players, if it does not cause some damage on non-deviating players or on the system itself. In other words, with a more refined, realistic modeling of Byzantine players, it is very likely that the very correctness of the TRAP protocol will be enough of a deterrent from deviating, which would lead to agreement directly at predecision level.

Appendix C

Basilic Additional Results

We show in this chapter additional results for Basilic.

C.1 Impossibility of consensus without active accountability

In the proof of Theorem 5.1 we considered that deceitful faults do not prevent termination, that is, that the protocol satisfies active accountability. We show in Corollary C.1 the analogous result in the case where deceitful processes can actually prevent termination, that is, if the protocol does not satisfy active accountability. In this case, since deceitful can have the same impact as Byzantine (in that they can prevent either agreement or termination), then the bounds decrease to $n > 3(t + d) + 2q$. Note that other protocols that use authentication may also be subject to this bound if they do not satisfy active accountability, as it is the case for Polygraph [47].

Corollary C.1. It is impossible for a protocol that solves consensus without satisfying active accountability to tolerate t Byzantine, d deceitful and q benign processes if $n \leq 3(t + d) + 2q$.

Proof. The proof is analogous to Theorem 5.1 with the difference that deceitful processes can actually prevent termination by sending conflicting messages. Thus, we have $n + t + d \leq 2n - 2q - 2t - 2d$, which means $n > 3(t + d) + 2q$. \square

C.2 Extended complexities of Basilic

Before GST and in the presence of an adversary controlling t Byzantine, d deceitful, and q benign processes, let \mathfrak{a} be the number of times the timer is reached before GST (i.e. $\mathfrak{a} \geq \lceil \frac{GST}{\Delta} \rceil$), then the message and bit complexities of AABC increase by a factor of $\mathfrak{a} \cdot n$, thus to $\mathcal{O}(\mathfrak{a}n^3)$ and $\mathcal{O}(\lambda \mathfrak{a}n^4)$, respectively. The same occurs with AARB's complexities. The time complexities are also affected by the time \mathfrak{a} to reach GST thus to $\mathcal{O}(\mathfrak{a}n)$ for AABC and the general Basilic, and $\mathcal{O}(\mathfrak{a})$ for AARB.

Since there are n pairs of reliable broadcasts and binary consensus instances in the Basilic general protocol, the time complexity is $\mathcal{O}(t + q + d)$, message complexity $\mathcal{O}(\mathfrak{a}n^3)$ and bit complexity $\mathcal{O}(\lambda \mathfrak{a}n^4)$. We show in Table C.2.1 the worst-case complexities of the three protocols.

Complexity	AARB	AABC	Basilic
Time	$\mathcal{O}(\mathfrak{a})$	$\mathcal{O}(n)$	$\mathcal{O}(n)$
Message	$\mathcal{O}(\mathfrak{a}n^2)$	$\mathcal{O}(\mathfrak{a}n^3)$	$\mathcal{O}(\mathfrak{a}n^4)$
Bit	$\mathcal{O}(\lambda \mathfrak{a}n^3)$	$\mathcal{O}(\lambda \mathfrak{a}n^4)$	$\mathcal{O}(\lambda \mathfrak{a}n^5)$

Table C.2.1: Complexities of Basilic protocols before GST.

Appendix D

ZLB Additional Results

We show in this chapter additional results for ZLB.

D.1 Number of branches and deceitful ratio

We show in Figure D.1.1 the minimum deceitful ratio (left) and number of deceitful processes (right) for the attackers to be able to cause a disagreement into at least a branches, for a voting threshold of $h = 2n/3$. It is specially interesting observing that a coalition of less than half of the system cannot perform anything else than a double-spending, while a deceitful ratio of $\delta < 11/18$ can at most perform a sextuple-spending, while being significantly close, at only $1/18$ of distance, to the threshold value of $2/3$. Notice also that a can only range from 1 to $n/3 + 1$, a value that is taken when $t + d = t_s = \lceil 2n/3 \rceil - 1$ and each of the $n/3 + 1$ honest processes belong to a different partition.

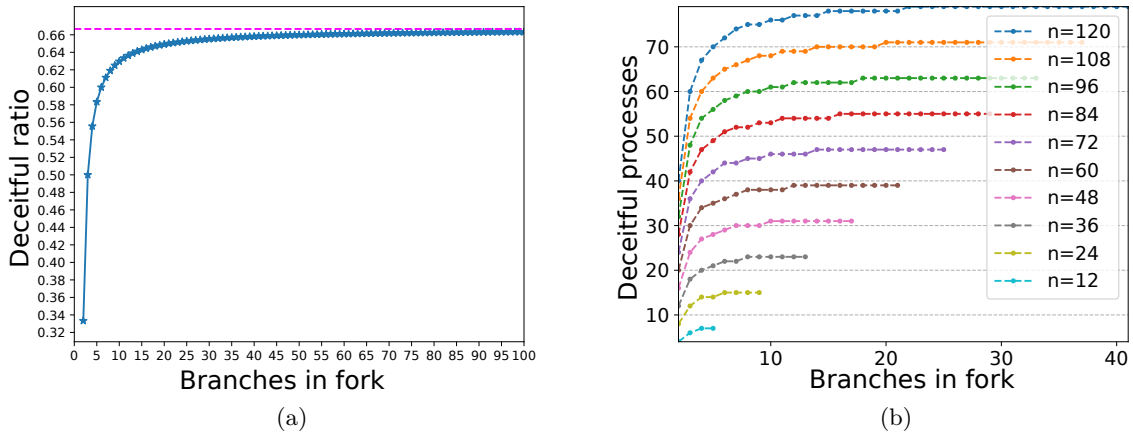


Figure D.1.1: Minimum deceitful ratio δ (left) and number of deceitful processes (right) required for a number of branches a in a blockchain fork for voting threshold $h = 2n/3$.

D.2 Fixed superblock size

Figure D.2.2 shows the throughput of ZLB in a large WAN of up to 300 AWS c5.4xlarge instances, for a total size of all proposals fixed to 200,000 transactions. We include two throughput results of our implementation, one for decisions and one for confirmations, for which we assume the maximum f possible, i.e., all replies must be received before confirming a value. We can see that the throughput of confirmed transactions is slightly lower, given that every process must wait to receive a certificate from every single other process, increasing the impact of slow processes. The performance decreases as the number of processes increases, mainly due to the increase in size of certificates. We omitted the confirmation throughput in Figure 5.6 as it was a negligible amount of time more than decisions for that setting, mainly due to the bottleneck being the validation of transactions, less noticeable for a fixed total size of transactions, as Figure D.2.2 shows.

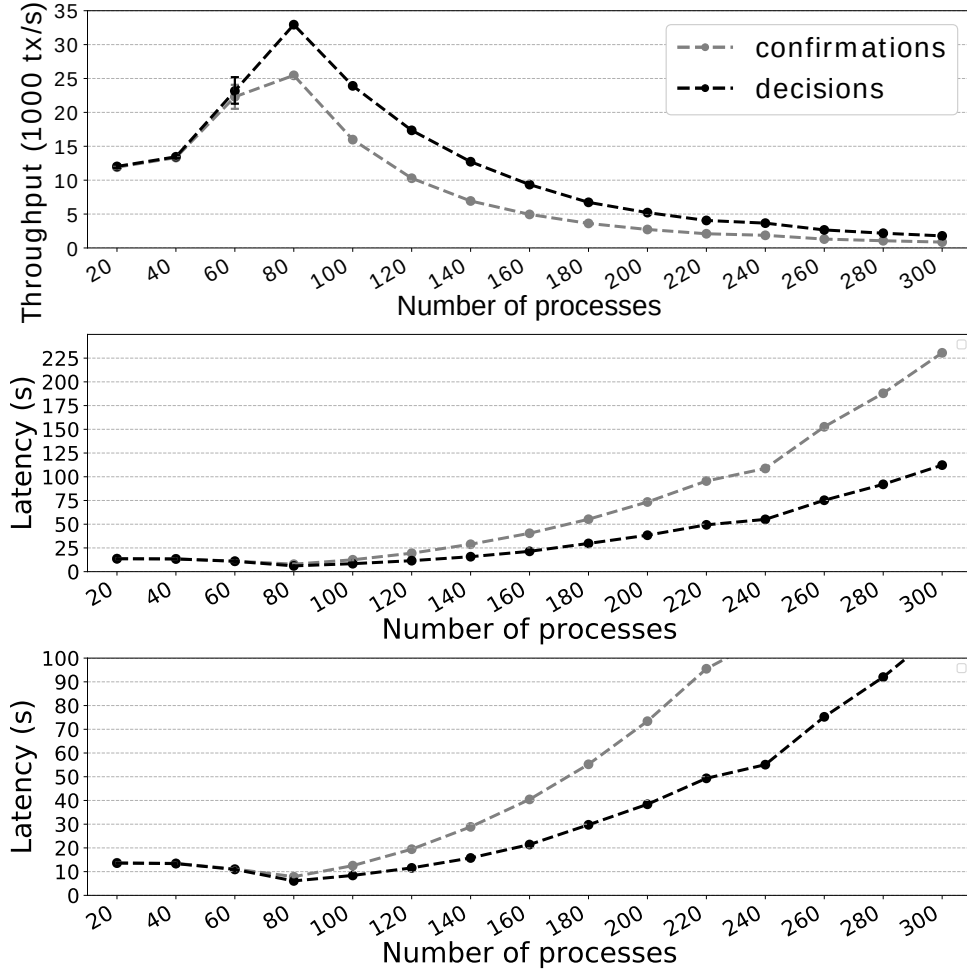


Figure D.2.2: Throughput (top) and latency (center, bottom) of ZLB for decided proposals, compared to confirmed proposals, for a size of the superblock fixed to 200,000 transactions.

D.3 Bitmask of binary consensus attack

The binary consensus attack can maximize disagreements by leading all branches to a different bitmask. However, a disagreement on a bit associated with a proposal broadcast by an honest process might not contribute to the specific attack intended by attackers (e.g. double-spending). We show in Figure D.3.3 however that if attackers do not maximize the disagreeing bits across branches, the number of disagreements decreases, even if they expose themselves in less binary consensus instances, for the binary consensus attack. That is, we test here the number of disagreements for a minimal and maximal Hamming distance between disagreeing bitmasks.

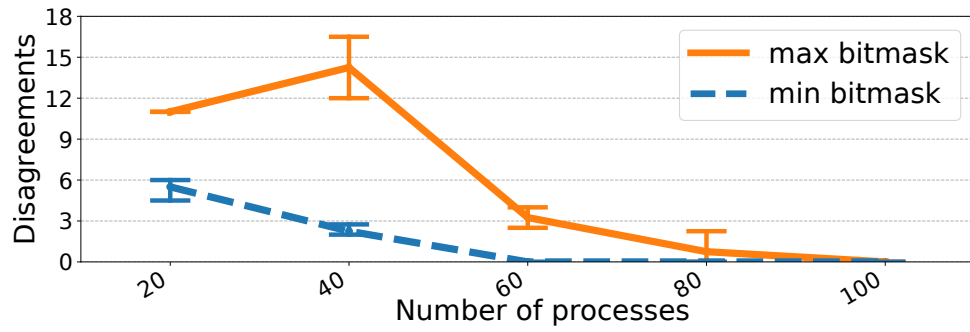


Figure D.3.3: Disagreeing decisions for the binary consensus attack for a minimal disagreement per bitmask of one bit or a maximal of all bits, for $f = d = \lceil 5n/9 \rceil - 1$.

Appendix E

Discussion: safety vs. performance of Kleroterion⁺

We discuss in this section additional changes to the protocol in order to guarantee safety.

Aggregating multiple Kleroterion⁺ random beacons. Kleroterion⁺ allows for $\omega \geq 1$ random beacons to be executed concurrently in ω different committees of size n , and then aggregated into one final random beacon. This way, in order to break the randomness of the beacon, the adversary needs to break the randomness of each of the ω outputs. That is, the probability of the adversary breaking the randomness of the random beacon given ω concurrent executions of Kleroterion⁺, each with a committee size of n , is $Pr(Y > t_s)^\omega$. This means that for $\omega = 5$ we have that $Pr(Y > t_s)^\omega < 10^{-14}$ for $n = 487$, meaning a total of $\omega * n = 2435$ processes in total to produce the final random output. In contrast, for $\omega = 1$ the required committee size for the same level of safety is $n = 3215 > 2435$. This means that this approach may provide better performance. It is important to note however that honest processes of each of the $\omega = 5$ concurrent executions of Kleroterion⁺ should only reconstruct the secrets once they have signed certificates for each of the decided commitments of each Kleroterion⁺ execution. This is to prevent an adversary controlling $t_s + 1$ processes in one of the executions from selecting the random output after learning the outputs of the rest of the executions, producing the desired aggregated output. Note additionally that such approach may decrease the probability of breaking randomness but it maintains the same probability of losing liveness.

Using VRFs to limit adversarial control. Additionally, in order to increase the bias-resistance of Kleroterion⁺, we could make processes choose the output from a Verifiable Random Function (VRF) as their input, instead of any input chosen uniformly at random. This way, even if the adversary controls one iteration $t_s + 1$ of the committee, they can only manipulate the output by choosing one of $\binom{n}{k}$ outputs where k is the number of VRF inputs that Kleroterion⁺ combines, and thus k can be one of $k \in [1, t_s + 1]$. The value chosen for k expresses a trade-off between unpredictability and bias-resistance. This creates an interesting trade-off to be selected by each application. On the one hand, decreasing k reduces the amount of combinations $\binom{n}{k}$ from which the adversary can choose to combine into a final random output, if the adversary controls enough processes of the committee, which increases bias-resistance for lower values of

k . On the other hand, if the adversary controls at least k processes in a committee, it can already compute the output of its own processes, and force the algorithm to select a specific predicted combination, which decreases unpredictability for lower values of k .