Efficient Synchronous Byzantine Consensus

by

Nibesh Shrestha
B.Eng., Tribhuvan University (2013)

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
July 17, 2023

Efficient Synchronous Byzantine Consensus

by

Nibesh Shrestha

Committee Approval:
We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Dr. Pencheng Shi Dissertation Advisor	Date
Dr. Kartik Nayak (Duke University) Dissertation Co-Advisor	Date
Dr. Stanisław Radziszowski Dissertation Committee Member	Date
Dr. Matthew Fluet Dissertation Committee Member	Date
Dr. Ricardo Figueroa Dissertation Defense Chairperson	Date

Certified by:

Dr. Pencheng Shi Ph.D. Program Director, Computing and Information Sciences	Date

Efficient Synchronous Byzantine Consensus

by

Nibesh Shrestha

Submitted to the
B. Thomas Golisano College of Computing and Information Sciences Ph.D. Program in Computing and Information Sciences
in partial fulfillment of the requirements for the
Doctor of Philosophy Degree
at the Rochester Institute of Technology

Abstract

With the emergence of decentralized technologies such as Blockchains, Byzantine consensus protocols have become a fundamental building block as they provide a consistent service despite some malicious and arbitrary process failures. While the Byzantine consensus problem has been extensively studied for over four decades under various settings, many challenges and open problems still exist. Improving the communication complexity and the latency or round complexity are the two key challenges in the design of efficient and scalable solutions for the Byzantine consensus problem. This thesis focuses on improving the communication complexity and the round complexity of the synchronous Byzantine consensus problem under various setup assumptions.

In this thesis, I will first present OptSync, a new paradigm to achieve optimistic responsiveness that allows a consensus protocol to commit with the best-possible latency under all conditions. A lower bound that relates to the commit latencies for an optimistically responsive protocol and matching upper bound protocols with optimal commit latency under all conditions will be presented.

Then, I will discuss consensus protocols in the absence of threshold setup; this setting supports efficient reconfiguration of participating parties. In this setting, I will present two efficient consensus protocols that incur quadratic communication per decision and optimistically responsive latency during optimistic conditions.

Next, I will discuss the design of communication and round efficient protocols for distributed key generation (DKG). I will present a new framework to solve the DKG problem and present two new constructions following the framework. The first protocol incurs cubic communication in expectation and expected constant rounds, while the second protocol incurs cubic communication in the worst-case and linear round complexity. Improved constructions for several useful primitives such as gradecast and multi-valued validated Byzantine agreement will also be presented.

Finally, I will present communication and round efficient protocols for parallel broadcast where all parties wish to broadcast their input. A generic reduction from parallel broadcast to graded parallel broadcast and validated Byzantine consensus will be presented along with improved constructions for gradecast with multiple grades and multi-valued validated Byzantine agreement.

Acknowledgments

I am immensely grateful to the many people who have supported me throughout my PhD journey.

First and foremost, I would like to thank my advisors, Professor Pengcheng Shi and Professor Kartik Nayak for their unwavering support and guidance throughout my doctoral studies. Professor Shi always provided me with valuable suggestions despite me not working in the area of his expertise. He has always been a kind and amazing person to interact with. I spent most of my time as a graduate student working with Kartik. I started working with Kartik when I knew very little about research. But, he always showed faith in my ability and supported me throughout. I will forever remember a quote he stated "the devil is in the details" to emphasize the importance of thinking rigorously about research problems, an art I am still learning to master. His ability to clearly articulate ideas and relentlessly work on challenging research problems is something I wish to inherit. His mentorship has been instrumental in my academic growth, and I could not have completed this work without his guidance. He also introduced me to an amazing group of researchers who helped me broaden my horizons and expand my knowledge. I also very much appreciate his guidance during my job search and his advice on many other aspects of life.

I would also like to extend my gratitude to my mentors, including Ittai Abraham, Aniket Kate and Dahlia Malkhi. Numerous research discussions we had, has helped me explore the field of consensus and applied cryptography, and deepen my understanding. Your contributions to this work have been significant, and I am grateful for the opportunity to work with you. Your insights, feedback, and support have been critical in shaping my research, and I am fortunate to have had you as mentors.

I am thankful to my student collaborator Adithya Bhat, specially for his mentorship on various secret sharing schemes. I also benefit from numerous technical discussions with Aditya Asgaonkar, Francesco D'Amato, Julian Loss, Atsuki Momose, Ling Ren, Sarisht Wadhwa, Sravya Yandamuri and Luca Zanolini. I would like to thank them for their valuable time. I would also like to thank Maofan "Ted" Yin for his wonderful consensus libraries.

I am grateful to Professors Matthew Fluet and Stanisław Radziszowski for their invaluable contributions as members of my doctoral committee, offering insightful suggestions that greatly enhanced the quality of this thesis. I would also like to extend my gratitude to Professors Mohan Kumar and Peizhao Hu for their guidance during the early part of my doctoral studies. I am also thankful to Professor Arthur Nunes Harwitt, with whom I had the opportunity to work as a teaching assistant during my PhD study.

I would also like to express my appreciation to my friends: Robik, Binyul, Krishna, Garegin, and Avinash for their friendship and support throughout my PhD journey. Your presence, humor, and encouragement have been a source of comfort and inspiration during challenging times. Thank you for being a part of my life and for making my graduate school experience more enjoyable.

Finally, I would like to thank my parents, my sisters, my brother and my wife, Pranisha who have been an endless source of love and understanding throughout this period. Your encouragement has been instrumental in my academic achievements, and I am forever grateful for your presence in my life.

Contents

Chapter 1

Introduction

Byzantine fault tolerant (BFT) consensus protocols provide a consistent service despite some malicious and arbitrary process failures. These protocols have been particularly useful in developing infrastructures that span across multiple entities some of which may be incentivized to act maliciously. As a result, Byzantine consensus protocols have been widely adopted to build decentralized blockchain technologies that promise tamper-proof ledger services without a central authority. While the Byzantine cosensus problem has been extensively studied for over 40 years under various models and assumptions with numerous protocols and impossibility results, many challenges and open problems still exist. Improving the communication complexity (i.e., reducing the number of bits honest parties exchange) and the latency or round complexity (i.e., the time required to reach a decision) are the two key challenges in the design of efficient and scalable solutions for the Byzantine consensus problem.

The Byzantine consensus problem has been studied under various network models. Widely studied network models include asynchronous model [17, 8], partially synchronous model [45, 32, 104, 27] and synchronous model [7, 71]. Protocols designed under asynchronous and partially synchronous model are tolerant to arbitrary delays in the network while protocols designed under synchrony assumption require messages to arrive at the specified time. However, asynchronous or partially synchronous Byzantine consensus protocols can tolerate only up to one-third Byzantine failures [45] while synchronous Byzantine consensus typically tolerate one-half Byzantine failures [52, 54, 71] or even 99% Byzantine failures [43]. This thesis focuses on improving the communication complexity and the round complexity of synchronous Byzantine consensus protocols under various setup assumptions.

The first chapter (Chapter 3) addresses the latency concern of the synchronous Byzantine consensus. In general, synchronous Byzantine consensus protocols tolerate up to one-half Byzantine failures. However, the latency to commit a consensus decision inherently depends on the prior-known pessimistic network delay Δ. This is in constrast to the partially synchronous or asynchronous consensus protocols which can commit *responsively* at actual network speed δ ($\delta << \Delta$). A recent work Thunderella [90] introduced the notion of *optimistic responsiveness* to allow a synchronous consensus protocol to commit at network speed in $O(\delta)$ time when certain optimistic conditions are met. In particular, the protocol can commit responsively when the leader and $> 3n/4$ replicas behave honestly, where n is the total number of replicas in the system. However, the Thunderella paradigm of optimistic responsiveness required explicit back-and-forth switching between two modes–*fast mode* where replicas commit in $O(\delta)$ time during optimistic conditions, and *slow mode* where the commit latency depends on Δ during non-optimistic conditions with some intermediary transition phase which incurs additional latency. Their paradigm has two major drawbacks: (i) it is hard to know whether optimistic conditions are met or not; switching to the fast path incorrectly will cause undue switching latency, and (ii) the adversary can worsen the overall latency by behaving honestly while on the slow path (thereby triggering a switch to the fast path) and not responding when on the fast path; forcing a switch to the slow path which incurs additional switching latency in between. To address these concerns, this thesis presents OptSync [100], a new paradigm of optimistic responsiveness where both slow and fast mode exist simultaneously and allows the protocol to commit with the best-possible commit latency under all circumstances. In the process, the paradigm also removes the need to perform explicit back-and-forth switching and its associated switching latency. A lower bound on the latency for such an optimistically responsive protocol and matching upper bound protocols that achieve the optimal latency are presented along with experimental evaluations that show significant latency improvement over the prior art.

The second and third chapters (Chapters 4 and 5) address the communication complexity of the Byzantine consensus protocol in the absence of threshold setup. All prior known synchronous BFT protocols [71, 50, 4, 104, 27, 8, 84] tolerating $t < n/2$ Byzantine faults assume threshold setup and make use of threshold signatures to reduce their communication complexity. However, threshold signatures require an initial setup phase called distributed key generation (DKG) to establish threshold keys among all participating replicas and hence, are not suitable in a setting where the participating replicas can change over time. In essence, the protocols relying on threshold signatures are not *reconfiguration-friendly*. In the absence of threshold signatures, all of the protocols incur cubic-communication; and hence are not scalable. This thesis explores design of communication efficient synchronous BFT protocols in the absence of threshold setup and threshold signatures. Two new BFT state machine replication (SMR) protocols with $O(\kappa n^2)$ communication per consensus

decision even in the absence of threshold signatures (κ is the security parameter) and varying commit latency are presented. Getting rid of threshold signatures allows for efficient reconfiguration of the participating replicas and does not require generating threshold keys each time a new replica joins the system. Efficient reconfiguration-friendly BFT SMR plays an important role in the design of several applications. Randpiper [21] and OptRand [22] uses these BFT SMR protocols to design efficient random beacon protocols [23] with the same complexity metric.

While chapters 4 and 5 design communication efficient BFT consensus protocols in the absence of threshold setup i.e., without DKG setup, DKG protocols are useful in many other cryptographic protocols such as threshold signatures [24, 99] and threshold encryption schemes [39]. The fourth chapter (Chapter 6) explores the design of communication and round efficient protocols for the distributed key generation problem. All prior works on synchronous DKG protocols [93, 87, 59, 30] assume a "broadcast channel" to abstract consensus mechanism and incur $O(\kappa n^4)$ communication and linear round complexity. This thesis presents a new framework to achieve communication and round efficient protocols to solve the DKG problem. In this framework, a broadcast channel is replaced with weaker consensus primitives such as gradecast which can be achieved in a communication efficient manner. By making use of the framework, two efficient DKG protocols are presented that incur $O(\kappa n^3)$ communication with either linear latency or expected constant round complexity. In the process, improved construction for gradecast and multi-valued validated Byzantine agreement are also obtained.

Finally, this thesis presents efficient constructions for parallel broadcast primitive, where all n parties wish to broadcast ℓ bit messages in parallel. Prior approaches for parallel broadcast naïvely run n instances of Byzantine agreement (or Byzantine broadcast) primitives increasing the communication complexity by the undesirable factor of n along with linear round complexity. This thesis shows a reduction from parallel broadcast tolerating $t < n/2$ Byzantine faults to graded parallel broadcast (a new primitive we introduce) and use a single instance of validated Byzantine consensus to achieve parallel broadcast protocols with $O(n^2\ell + \kappa n^3)$ communication and expected constant number of rounds. In the process, improved constructions for gradecast protocol with multiple grades with asymptotically optimal communication complexity and a multi-valued validated Byzantine agreement protocol with asymptotically optimal communication complexity are also obtained.

1.1 Overview of Contributions

This section presents an overview of the research conducted for this thesis.

Chapter 3: On the Optimality of Optimistic Responsiveness. Synchronous consensus protocols, by definition, have a worst-case commit latency that depends on the bounded network delay. The notion of optimistic responsiveness was recently introduced to allow synchronous protocols to commit instantaneously when some optimistic conditions are met. In this work [100], we revisit this notion of optimistic responsiveness and present optimal latency results.

We present a lower bound for Byzantine Broadcast that relates the latency of optimistic and synchronous commits when the designated sender is honest and while the optimistic commit can tolerate some faults. We then present two matching upper bounds for tolerating t faults out of $n = 2t+1$ parties. Our first upper bound result achieves optimal optimistic and synchronous commit latency when the designated sender is honest and the optimistic commit can tolerate at least one fault. We experimentally evaluate this protocol and show that it achieves throughput comparable to state-of-the-art synchronous and partially synchronous protocols and under optimistic conditions achieves latency better than the state-of-the-art. Our second upper bound result achieves optimal optimistic and synchronous commit latency when the designated sender is honest but the optimistic commit does not tolerate any faults. The presence of matching lower and upper bound results make both of the results tight for $n = 2t + 1$. Our upper bound results are presented in a state machine replication setting with a steady-state leader who is replaced with a view-change protocol when they do not make progress. For this setting, we also present an optimistically responsive protocol where the view-change protocol is optimistically responsive too.

Material in this chapter first appeared as: Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020

Chapters 4 and 5: Communication Efficient State Machine Replication without Threshold Signatures. Byzantine consensus protocols, in general, assume threshold setup and rely on threshold signatures to achieve good communication complexity. However, protocols relying on threshold setup do not allow efficient reconfiguration of participating parties. In the absence of threshold setup, prior Byzantine consensus protocols incur cubic communication per decision. Avoiding threshold setup allows for efficient reconfiguration of participating parties. In this work [21], we design a communication efficient BFT consensus protocol that incurs $O(\kappa n^2)$

communication per decision without threshold signatures and tolerates optimal $t < n/2$ Byzantine failures. The resulting construction has been useful to achieve consensus in widespread applications such as random beacons [21] and distributed key generation [101].

While the above BFT consensus protocol achieve a communication efficient solution to consensus, it incurs a large latency to reach a decision. In this work [22], we design OptRand, an optimistically responsive consensus protocol that commits decisions at actual network speed during optimistic conditions. This protocol incurs $O(\kappa n^2)$ communication per decision in the absence of threshold setup and commits decisions in an optimistically responsive manner. This protocol closely follows the optimistic responsiveness paradigm introduced in Chapter 3.

Material in Chapter 4 first appeared as: Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3502–3524, 2021.

Material in Chapter 5 first appeared as: Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. OptRand: Optimistically responsive distributed random beacons. In *Proceedings of the 30th Network and Distributed System Security Symposium (NDSS)*, 2023.

Chapter 6: Synchronous Distributed key generation without Broadcasts. Distributed key generation (DKG) is a key building block in developing many efficient threshold cryptosystems. This work [101] initiates the study of communication complexity and round complexity of DKG protocols over a point-to-point (bounded) synchronous network. Our key result is the first synchronous DKG protocol for discrete log-based cryptosystems with $O(\kappa n^3)$ communication complexity (κ denotes a security parameter) that tolerates any $t < n/2$ Byzantine faults among n parties. We present two variants of the protocol: (i) a protocol with worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds, and (ii) a protocol with expected $O(\kappa n^3)$ communication and expected constant rounds. In the process of achieving our results, we design (1) a novel weak gradecast protocol with a communication complexity of $O(\kappa n^2)$ for linear-sized inputs and constant rounds, (2) a primitive called "recoverable set of shares" for ensuring recovery of shared secrets, (3) an oblivious leader election protocol with $O(\kappa n^3)$ communication and constant rounds, and (4) a multi-valued validated Byzantine agreement (MVBA) protocol with $O(\kappa n^3)$ communication complexity for linear-sized inputs and expected constant rounds. Each of these primitives is of independent interest.

Material in Chapter 6 is currently under submission.

Chapter 7: Communication and Round Efficient Parallel Broadcast protocols. This work [11] focuses on the *parallel broadcast* primitive, where each of the n parties wish to broadcast their ℓ-bit input in parallel. We consider the *authenticated* model with PKI and digital signatures that is secure against $t < n/2$ Byzantine faults under a *synchronous* network.

We show a generic reduction from parallel broadcast to a new primitive called graded parallel broadcast and a single instance of validated Byzantine agreement. Using our reduction, we obtain parallel broadcast protocols with $O(n^2\ell+\kappa n^3)$ communication (κ denotes a security parameter) and expected constant rounds. Thus, for inputs of size $\ell = \Omega(n)$ bits, our protocols are asymptotically free.

Our graded parallel broadcast uses a novel gradecast protocol with multiple grades with asymptotically optimal communication complexity of $O(n\ell + \kappa n^2)$ for inputs of size ℓ bits. We also present a multi-valued validated Byzantine agreement protocol with asymptotically optimal communication complexity of $O(n\ell + \kappa n^2)$ for inputs of size ℓ bits in expectation and expected constant rounds; this protocol may be of independent interest.

Material in Chapter 7 is currently under submission.

Chapter 2

Background

There are various formulations of the Byzantine consensus problem in the literature. In this chapter, we review a few widely studied formulations. The term "consensus" is used as a collective term for all such variations.

2.1 Byzantine Broadcast and Byzantine Agreement

Two well-studied formulations are Byzantine broadcast and Byzantine agreement, as introduced by Pease, Shostak and Lamport [91, 77]. In Byzantine broadcast, there is a designated *sender* who tries to broadcast a value to all parties. Up to t out of n participating parties may be Byzantine faulty and perform arbitrary actions. The non-faulty parties are said to be *honest* and execute the protocol as specified. At the end of the protocol execution, all honest parties are required to agree on a common value. To rule out trivial outputs (such as all honest parties outputs $\perp$), a validity requirement is placed to ensure all honest parties output the sender's value when the sender is honest. More formally, the Byzantine broadcast problem is defined as as follows.

Definition 2.1.1 (Byzantine Broadcast [43])**.** *A Byzantine broadcast protocol provides the following three guarantees.*

- ***Agreement.*** *If two honest replicas commit values b and b' respectively, then $b = b'$.*
- ***Termination.*** *All honest replicas eventually commit.*

- ***Validity.*** *If the designated sender is honest, then all honest replicas commit on the value it proposes.*

In Byzantine agreement problem, all parties provide an input value; there is no designated sender. Up to t out of n participating parties are allowed to be Byzantine faulty. The requirements for agreement and termination are similar to the Byzantine broadcast. A requirement for validity is placed to rule to trivial solutions and requires all honest parties output a common input value b which is the input of all honest parties. More formally, the Byzantine agreement problem is defined as follows.

Definition 2.1.2 (Byzantine Agreement [77])**.** *A Byzantine agreement protocol provides the following three guarantees.*

- ***Agreement.*** *If two honest replicas commit values b and b' respectively, then $b = b'$.*
- ***Termination.*** *All honest replicas eventually commit.*
- ***Validity.*** *If all honest parties hold the same input value b, then they all commit on b.*

Lamport, Shostak and Pease [91, 77] studied lower bounds on the number of parties required to tolerate t Byzantine faults assuming synchronous communication model. In synchronous communication model, the protocol is executed in synchronized rounds and messages sent by an honest party at the beginning of a round is guaranteed to be received by the end of the round. In the *plain authenticated* model (without public key infrastructure (PKI) and digital signatures), they showed Byzantine broadcast and agreement can be solved if and only if $t < n/3$. The plain authenticated model is also called *unauthenticated* model. In the *authenticated* model, assuming PKI and digital signatures, Byzantine broadcast can be solved if $t < n - 1$ and Byzantine agreement can be solved if $t < n/2$. They also presented protocols with optimal fault-tolerance; albeit their protocol had exponential communication complexity.

Fully polynomial protocols were later given by Dolev and Strong [43] for $t < n/2$ case in the authenticated setting and by Garay and Moses [58] for $t < n/3$ case in the unauthenticated setting. Dolev and Reischuk [41] showed a communication complexity lower bound of $\Omega(n^2)$ for Byzantine broadcast, which also applies to Byzantine agreement. Berman, Garay and Perry [20] provided the first protocol with optimal communication complexity for $t < n/3$ in the unauthenticated setting. Very recently, Momose and Ren [84] gave Byzantine agreement protocol with $O(\kappa n^2)$ communication for $t < n/2$ in the authenticated setting.

A lower bound on round complexity for deterministic Byzantine consensus problem has been given in several works [51, 12, 43]. In particular, any deterministic Byzantine consensus problem must incur at least $t + 1$ rounds. To circumvent the lower bound, randomization has been used to achieve consensus in constant expected rounds in several works [49, 50, 71]. In the asynchronous communication model, where messages are eventually delivered, a well-known FLP impossibility result [53] rules out any deterministic solutions. Randomization [17, 26] and partial synchrony [45] has been proposed to circumvent the lower bound.

2.2 Byzantine Fault Tolerant State Machine Replication

The Byzantine broadcast and Byzantine agreement formulations are interesting formulations to study theoretical feasiblity results. However, they consider a single consensus instance i.e., decision on a single value. In a practical setting, it is desirable to keep committing new decisions and build a *common log* of committed decisions. This notion is captured by state machine replication [97] formulation. In state machine replication (SMR) formulation, there are designated parties called *clients* who supply commands/requests to a set of parties who execute the consensus protocol. The set of parties who execute the consensus protocol are called *replicas*. Replicas provide a consistent service despite some replicas failing arbitrarily i.e., Byzantine failures. The consistent service provides two guarantees–*safety* and *liveness*. Safety requires honest replicas to not commit different values at the same log position while liveness requires client request to be eventually committed. More formally, Byzantine fault tolerant state machine replication is defined as follows.

Definition 2.2.1 (Byzantine Fault-tolerant State Machine Replication [97])**.** *A Byzantine fault-tolerant state machine replication protocol commits client requests as a linearizable log to provide a consistent view of the log akin to a single non-faulty server, providing the following two guarantees.*

- ***Safety.*** *Honest replicas do not commit different values at the same log position.*
- ***Liveness.*** *Each client request is eventually committed by all honest replicas.*

In general, a BFT SMR protocol consists of a designated replica called the *leader* who proposes client requests to other replicas. In this regard, the Byzantine broadcast and BFT SMR formulation may appear as the same. However, BFT SMR protocol requires at least $n \geq 2t + 1$ [97] while Byzantine broadcast can be solved for $t < n - 1$ in the authenticated synchronous setting. The key difference is that Byzantine broadcast requires honest parties to stay in agreement while BFT

SMR additionally requires public verifiability (i.e., the external clients should be able to verify a committed decision) which requires $t < n/2$. In the asynchronous communication model, both the problems are equivalent as both the formulations require $t < n/3$.

A well-known BFT SMR protocol called Practical Byzantine Fault Tolerance (PBFT) [32] was given by Castro and Liskov that works in the partial synchrony model and has optimal resilience (i.e., $t < n/3$). A number of follow-up works [9, 80, 33] have been proposed to improve its efficiency. Protocols in partial synchrony model provide safety during the periods of asynchrony and liveness during the period of synchrony. All of these solutions have $O(\kappa n^2)$ communication per consensus decision. A recent work HotStuff [104] provided the first SMR protocol with linear communication complexity assuming threshold signatures.

In synchronous setting, the protocols [3, 43, 71] were mostly designed assuming *lock-step* synchrony model where honest parties are assumed to have access to synchronized rounds which is not practical. Later, Dfinity [65, 6] introduced *non lock-step* synchrony model where the local measured Δ time is assumed to be a correct upper bound for the network delay. This allowed a synchronous protocol to make progress at actual network speed while commit depends on the pessimistic network delay Δ. In their protocol, the latency to commit a decision required 17Δ time. In this model, a recent work [9] reduced the commit latency to only Δ and showed its optimality. A series of recent works [90, 89, 7] explored the notion of optimistic responsiveness to allow a synchronous consensus protocol to commit at actual network delay when certain optimistic conditions are met. Additionally, we note that all known BFT SMR protocols in the synchronous model have $O(\kappa n^2)$ communication per consensus decision in the presence of threshold signatures, and cubic communication without it.

2.3 Multi-valued Validated Byzantine Agreement

Most formulations of Byzantine consensus require the output value be the input of at least one honest party. In multi-valued validated Byzantine agreement (MVBA) formulation, the output value can be the input of any party including a Byzantine party; as long as it is externally valid. In an MVBA protocol, there is an external valid function ex-validation that every party has access to. Every honest parties start with some externally valid input v_i, and on termination must output a value. An MVBA protocol has the following properties:

Definition 2.3.1 (Multi-valued Validated Byzantine Agreement [8, 79, 101])**.** *A protocol solves multi-valued validated Byzantine agreement if it satisfies following properties except with negligible*

probability in the security parameter κ:

- ***Validity.*** *If an honest party decides a value v, then* ex-validation$(v) = true$.
- ***Agreement.*** *No two honest parties decide on different values.*
- ***Termination.*** *If all honest parties start with externally valid values, all honest parties eventually decide.*

Multi-valued validated Byzantine agreement was first introduced by Cachin et al. [28] to allow honest parties to agree on any externally valid values. Their protocol works in asynchronous communication model and has optimal resilience ($t < n/3$) with $O(n^2\ell + \kappa n^2 + n^3)$ communication for input of size ℓ. Abraham et al. [8] gave an MVBA protocol with optimal resilience and $O(n^2\ell + \kappa n^2)$ communication in the same asynchronous setting. Lu et al. [79] extended the work of Abraham et al. [8] to handle long messages of size ℓ with a communication complexity of $O(n\ell + \kappa n^2)$. To the best of our knowledge, prior to our work, MVBA has not been explored in the synchronous communication model.

2.4 Parallel Broadcast

In parallel broadcast primitive (aka, *interactive consistency* [91]), all parties wish to broadcast their input in parallel. The parallel broadcast problem is formally defined as follows:

Definition 2.4.1 (Parallel Broadcast [91])**.** *In a parallel broadcast protocol, each party P_i has its input value v_i and each party P_i outputs a n-element vector $\mathcal{V}_i$ of values. A parallel broadcast protocol tolerating t Byzantine failures has the following properties:*

- ***Agreement.*** *All honest parties must agree on the same vector of values $\mathcal{V} = [v_1, \ldots, v_n]$.*
- ***Validity.*** *If the input of an honest party P_j is v_j, then $\mathcal{V}_i[j] = v_j$.*
- ***Termination.*** *All honest parties must eventually decide on a vector $\mathcal{V}$.*

The notion of parallel-broadcast was originally introduced by Pease, Shostak and Lamport [91]. They studied the problem in both authenticated and unauthenticated model and showed that the problem can be solved for $t < n/3$ in the unauthenticated model and for $t < n - 1$ in the

authenticated model. They also gave upper bound protocols with optimal fault tolerance, but with exponential communication complexity. Efficient solutions for parallel broadcast has been explored in several works [18, 1, 103] under various models and setup assumptions.

2.5 Primitives

In this work, we use several primitives which are explained below:

Linear erasure and error correcting codes. We use standard (n, b) Reed-Solomon (RS) codes [95]. This code encodes b data symbols into codewords of n symbols and can decode b elements of the codewords to recover the original data.

- ENC. Given inputs $m_1, \ldots, m_b$, an encoding function ENC computes $(s_1, \ldots, s_n) = \mathsf{ENC}(m_1, \ldots, m_b)$, where $(s_1, \ldots, s_n)$ are codewords of length n. A combination of any b elements of the codeword uniquely determines the input message and the remaining of the codeword.
- DEC. The function DEC computes $(m_1, \ldots, m_b) = \mathsf{DEC}(s_1, ..., s_n)$, and is capable of tolerating up to c errors and d erasures in codewords $(s_1, \ldots, s_n)$, if and only if $n - b \geq 2c + d$.

Cryptographic accumulators. The cryptographic accumulator constructs an accumulation value for a set of values and produces witness for each value in the set. Given accumulation value and a witness, any party can verify if a value is in the set. Formally, given a parameter κ, and a set D of n values $d_1, \ldots, d_n$, an accumulator has the following interface:

- $\mathsf{Gen}(1^\kappa, n)$: takes a parameter κ and an accumulation threshold n (an upper bound on the number of values that can be accumulated securely), returns an accumulator key ak. The accumulator key ak is part of the trusted setup and therefore is public to all parties.
- $\mathsf{Eval}(ak, \mathcal{D})$: takes an accumulator key ak and a set $\mathcal{D}$ of values to be accumulated, returns an accumulation value z for the value set $\mathcal{D}$.
- $\mathsf{CreateWit}(ak, z, d_i, \mathcal{D})$: takes an accumulator key ak, an accumulation value z for $\mathcal{D}$ and a value d_i, returns $\perp$ if $d_i \notin \mathcal{D}$,and a witness w_i if $d_i \in \mathcal{D}$.
- $\mathsf{Verify}(ak, z, w_i, d_i)$: takes an accumulator key ak, an accumulation value z for $\mathcal{D}$, a witness w_i and a value d_i, returns true if w_i is the witness for $d_i \in \mathcal{D}$, and false otherwise.

Chapter 3

On the Optimality of Optimistic responsiveness

3.1 Introduction

Byzantine fault-tolerant (BFT) protocols based on a synchronous network have a high resilience of up to one-half Byzantine faults. In comparison, BFT protocols under asynchronous or partially synchronous networks can tolerate only one-third Byzantine faults. Although partially synchronous protocols have a lower tolerance for Byzantine faults, they have an advantage in terms of the latency to commit – they can commit in $O(\delta)$ time where δ is the actual latency of the network. On the other hand, the latency for synchronous protocols depends on Δ, where Δ is a pessimistic bound on the network delay.

A recent work, Hybrid Consensus [89], formalized this difference by introducing a notion called *responsiveness*. A protocol is responsive if its commit latency depends only on the actual network delay δ, but not the pessimistic upper bound Δ. In this regard, asynchronous and partially synchronous protocols are responsive by design, whereas synchronous protocols are not.

For synchronous protocols, a notion called *optimistic responsiveness* was introduced by Thunderella [90]; this allows synchronous protocols to commit responsively when some *optimistic* conditions are met. Thunderella is safe against up to one-half Byzantine faults. Moreover, if a "leader" and $> 3n/4$ replicas are honest, and if they are on a "fast-path", then replicas can commit responsively in $O(\delta)$ time; otherwise, the protocol falls back to a "slow-path", which has a commit latency

that depends on Δ.

The Thunderella paradigm of optimistic responsiveness requires replicas to know which of the two paths they are on, and explicitly switch between them. If, at some point, the optimistic conditions cease to be met, the replicas switch to the slow-path. When they believe the optimistic conditions start to hold again, they switch back to the fast-path. Thunderella uses Nakamoto's protocol [85] or the Dolev-Strong protocol [43] as their slow-path. Thus, the slow-path, as well as the switch between the two paths, is extremely slow, requiring $O(\kappa\Delta)$ and $O(n\Delta)$ latency respectively (where κ is a security parameter). The slow-path latency can be improved to 2Δ using state-of-the-art synchronous protocols [7].

Can we further improve the latency of optimistically responsive synchronous protocols? Before answering the question, let us emphasize an important point in the study of optimistic responsiveness: replicas do not know whether the optimistic conditions are met. If all the replicas know, in the case of Thunderella, whether or not fewer than $\frac{1}{4}$ replicas are Byzantine, then we can use a protocol with optimal latency for that setting. Under optimistic conditions, we can use partially synchronous protocols [104, 75, 34, 32] to commit responsively; otherwise, we can use a state-of-the-art synchronous protocol tolerating a minority faults to commit in $\Delta + O(\delta)$ time [7, 9]. In contrast, the slow-path–fast-path switching paradigm, even if it uses optimal protocols in the two respective paths, still leaves a lot to be desired. If we start off in the wrong path, then we incur an additional switching delay, making the latency worse than either of the competing options under their respective conditions. More importantly, since there is no way to verify whether the optimistic conditions hold, such a protocol cannot tell when to switch to the fast-path, and hence will likely "miss out" on some periods with optimistic conditions.

This chapter explores optimality of optimistic responsiveness with the above restriction in mind. Specifically, we ask,

What is the optimal latency of an optimistically responsive synchronous protocol?

To answer this question, we obtain upper and lower bounds for the latency of such protocols. We also show that our protocol has better latency and comparable throughput in practice compared to state-of-the-art synchronous and partially synchronous protocols.

A lower bound on the latency of an optimistically responsive synchronous protocol. Our first result presents a lower bound on the latency of such optimistically responsive synchronous protocols. Specifically, we show the following result:

Theorem 1 (Lower bound on the latency of an optimistically responsive synchronous protocol, informal)**.** *There does not exist a Byzantine Broadcast protocol in an unsynchronized start model that can tolerate $t \geq n/3$ faults and achieve the following simultaneously when the designated sender is honest, messages sent by non-faulty parties arrive instantaneously, and all honest parties start at time 0:*

(i) (optimistic commit) all honest parties commit before time $O(\delta)$ when there are $\max(1, n-2t)$ crash faults, and

(ii) (synchronous commit) all honest parties commit before time $2\Delta - O(\delta)$ when there are t crash faults.

Thus, if a Byzantine Broadcast protocol tolerating $t \geq n/3$ corruption has an optimistic (fast) commit with latency $O(\delta)$ while still being able to tolerate $\max(1, n-2t)$ faults, then the synchronous (slow) commit should have a latency $\geq 2\Delta - O(\delta)$ when tolerating t faults. This lower bound applies to protocols in an *unsynchronized start* model where parties do not all start the protocol at the same time (explained later).

Our next two results present matching upper bounds for $n = 2t + 1$. In our protocols, when the conditions for an *optimistic commit* are met, replicas commit optimistically. Otherwise, they commit using the *synchronous commit* rule. Thus, intuitively, they exist on both paths *simultaneously* without requiring an explicit switch. Since all of our upper bounds require $O(\delta)$ time for the optimistic commit, whenever appropriate, we also call it a *responsive commit*.

Optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized responsive quorum. Our first protocol obtains optimistic responsiveness where the synchronous commit has a commit latency of 2Δ, while the responsive commit has a latency of 2δ using quorums of size $> 3n/4$. Specifically, we show the following:

Theorem 2 (Optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized responsive quorum, informal)**.** *There exists a Byzantine Broadcast protocol tolerating $< n/2$ faults, and under an honest sender achieves the following simultaneously:*

(i) (responsive commit) a commit latency of 2δ when $> 3n/4$ replicas are honest, and

(ii) (synchronous commit) a commit latency of $2\Delta + O(\delta)$ otherwise.

Intuitively, the fundamental property that this upper bound provides in comparison to Thunderella

or Sync HotStuff is *simultaneity*, i.e., replicas do not need to agree on specific paths for performing a responsive commit or a synchronous commit. Moreover, the parameters obtained in this result are optimal. First, the early stopping lower bound due to Dolev-Reischuk-Strong [42] states that when the number of faults is f, and the maximum number of faults is t, each execution of Byzantine Broadcast requires $\min(t+1, f+2)$ rounds. Hence, no protocol tolerating a fault can have latency less than 2δ. Second, the $> 3n/4$ quorum size is tight due to a lower bound in Thunderella [90]; the bound says that no protocol can have a worst-case resilience of one-half Byzantine replicas while being optimistically responsive for more than $n/4$ Byzantine replicas. Finally, latency for the synchronous commit is optimal (ignoring $O(\delta)$ delays) due to our first result.

Optimal optimistic responsiveness with Δ-synchronous latency and n-sized responsive quorum. The $2\Delta - O(\delta)$ latency bound for a synchronous commit is applicable when the optimistic commit can tolerate $\max(1, n-2t)$ faults. In this result, we show that the synchronous latency can be improved if the optimistic commit guarantees hold only when all $n = 2t+1$ replicas are honest.

Theorem 3 (Optimistic responsiveness with Δ-synchronous latency and n-sized responsive quorum, informal)**.** *There exists a Byzantine Broadcast protocol tolerating $< n/2$ faults, and under an honest sender achieves the following simultaneously:*

(i) (responsive commit) a commit latency of 2δ when all n replicas are honest, and

(ii) (synchronous commit) a commit latency of $\Delta + O(\delta)$ otherwise.

The responsive commit latency is optimal due to Dolev et al. [42] while the synchronous commit latency Δ is optimal (ignoring $O(\delta)$ delays) due to the lower bound in Sync HotStuff [7].

Implementation and evaluation. We implement and evaluate the performance of our first protocol and compare it with state-of-the-art synchronous and partially synchronous protocols. We note that although the upper bounds were presented for Byzantine Broadcast in the theorem statements, in practice, such protocols will be useful in a state machine replication (SMR) setting for consensus on a sequence of values. Hence, we describe as well as implement our protocols in an SMR setting. In the SMR setting, our protocols assume a steady state leader proposing a sequence of values. Whenever the leader does not make progress, it is replaced using a view-change protocol. An honest designated sender in Byzantine Broadcast is thus equivalent to having an honest leader in a state machine replication setting. Thus, when the leader is honest, our protocol from Theorem 2 can commit every value optimistically in 2δ time and synchronously in $2\Delta + O(\delta)$. Moreover, the

honest leader can propose consecutive values as fast as 2δ time independent of whether commits are performed responsively or synchronously.

In our evaluation, we observe that under optimistic conditions, our latency is better than even a partially synchronous protocol such as HotStuff [104] since HotStuff requires more rounds of communication. Our protocol also obtains a throughput comparable to these protocols.

Optimistic responsiveness with responsive view-change. Our upper bound protocols can commit responsively when the leader is honest and optimistic conditions are met. However, when executing on a sequence of values, for reasons such as fairness or distribution of work, we may want to change leaders every block, or every few blocks. Indeed, several recent protocols have been designed with this goal in mind [7, 34, 35, 60, 65, 98]. For the upper bounds described earlier, the view-change protocols, although efficient, still require $4\Delta + O(\delta)$ time. Such a latency is reasonable if a view-change happens only occasionally. However, the incurred latency maybe high if we need to change views after every block. Moreover, the latency is incurred even when the optimistic conditions are met.

Our final result addresses this concern and presents a protocol which has an optimistically responsive view-change as well. Thus, when rotating among honest leaders and if $> 3n/4$ replicas are honest, the steady state commit and view change can both finish in $O(\delta)$ time. On the other hand, even if the optimistic conditions are not met, the protocol requires 2Δ time to do a view change and $3\Delta + O(\delta)$ time to commit a block in the steady state.

Summary of contributions. To summarize, we make the following contributions in this work:

1. We present a lower bound on the latency for optimistic responsiveness (Section 3.3).
2. We then present two upper bound results. Section 3.4 presents an optimal optimistically responsive protocol with 2Δ-synchronous latency tolerating at least 1 fault in the responsive commit. We present an optimal optimistically responsive protocol with Δ-synchronous latency tolerating no crash faults in the responsive commit in Section 3.5.
3. We present an optimistically responsive protocol that includes an optimistically responsive view-change (Section 3.6).
4. We evaluate our 2Δ-synchronous protocol (Section 3.7).

3.2 Model and Definitions

We consider a standard State Machine Replication (SMR) problem used for building a fault tolerant service to process client requests. The system consists n replicas out of which $t < n/2$ replicas are Byzantine faulty. Byzantine replicas may behave arbitrarily. The aim is to build a consistent linearizable log across all non-faulty (honest) replicas such that the system behaves like a single non-faulty server in the presence of $t < n/2$ Byzantine replicas.

We assume the network between replicas includes a standard synchronous communication channel with point-to-point, authenticated links between them. Messages between replicas may take at most Δ time before they arrive, where Δ is a known maximum network delay. To provide safety under adversarial conditions, we assume that the adversary is capable of delaying the message for an arbitrary time upper bounded by Δ. The actual message delay in the network is denoted by δ. We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a replica p is digitally signed by p's private key and is denoted by $\langle x \rangle_p$.

3.3 A Lower Bound on the Latency of Optimistic Responsiveness

An optimistically responsive synchronous protocol has two commit rules – an optimistic commit rule and a synchronous commit rule. This section presents a lower bound that captures the relationship between the latencies of the two commit rules. Essentially, it says that if the optimistic commit rule is too fast, then the synchronous commit rule has to be correspondingly slower. Specifically, the sum of the latencies of the two commit rules should be at least 2Δ time.

In a bit more detail, suppose that there exists a protocol with an optimistic commit rule tolerating $\max(1, n - 2t)$ faults with a commit latency of $< \alpha$ time for some $0 < \alpha < \Delta$ when all messages arrive instantaneously. The lower bound then proves that if the optimistically responsive protocol can tolerate $t \geq n/3$ Byzantine faults, then its synchronous commit rule cannot have a latency $< 2\Delta - \alpha$. The converse is also true: if there exists a protocol tolerating $t \geq n/3$ faults and committing with a latency of $< 2\Delta - \alpha$, then it cannot commit with $< \alpha$ latency in an optimistic case that tolerates $\max(1, n - 2t)$ faults even when messages arrive instantaneously.

Unsynchronized starts. Often, in protocols involving multiple parties, not all parties start the protocol at the same time due to network delays. We refer to this model as the unsynchronized-

start model. Such a model captures state machine replication protocols where replicas move to the next view/slot at different times.

To formalize this model, we assume that honest parties start the protocol execution at different times decided by an adversary such that the following conditions hold: (i) each honest party starts the protocol at time $< \Delta$ and (ii) an honest party starts the protocol before receiving a message from any other party. Byzantine parties, on the other hand, are assumed to start the protocol execution at time 0. The parties start the protocol with a fixed state independent of when the protocol execution started; in particular, they do not have access to the execution start time.

Intuition and proof. The intuition behind the lower bound is to show a *split-brain* attack that can be performed by a minority of Byzantine replicas if a protocol has a sum of latencies for the two commit rules to be less than 2Δ. For simplicity, we present the intuition with $n = 2t+1$. First, observe that any protocol tolerating minority Byzantine faults cannot use quorum sizes larger than $n - t = t + 1$ in the worst case. Hence, it is always possible that a single honest replica R commits to a value due to a quorum of messages received from only the Byzantine replicas if it does not wait long enough before committing. Second, since the optimistic commit rule can tolerate at least one crash fault, replicas (set P) committing through the optimistic rule may commit without receiving any messages from replica R. Thus, to avoid a safety violation through a split-brain attack, replicas in P and R must communicate protocol instance specific messages with each other. Using the fact the start message from the trusted environment may be delayed by $\beta < \Delta$, replicas in P may start the protocol only at β time after which they receive messages instantaneously. Moreover, it takes Δ time for messages from P to arrive at R. For a specific value of $\beta = \Delta - \alpha$, R may commit a different value if it commits within $2\Delta - \alpha$ time. On the other hand, since P is performing an optimistic commit, it may not wait for more than α time after receiving its start message (at time β) before committing. This is not sufficient to receive any message from R. Thus, the sum of the latencies of the two commit rules should be at least 2Δ time. We now present the formal lower bound below.

Theorem 4 (Lower bound on the latency of an optimistically responsive synchronous protocol)**.** *For $0 < \alpha < \Delta$, there does not exist a Byzantine Broadcast protocol in an unsynchronized start model that can tolerate $t \geq n/3$ faults and achieve the following simultaneously when the designated sender is honest, messages sent by non-faulty parties arrive instantaneously, and all honest parties start at time 0:*

(i) (optimistic commit) all honest parties commit before time α when there are $\max(1, n-2t)$ crash faults, and

(ii) (synchronous commit) all honest parties commit before time $2\Delta - \alpha$ when there are t crash faults.

Proof. Suppose there exists a protocol that simultaneously achieves both properties above. We will show a sequence of worlds, and through an indistinguishability argument prove a violation in the agreement property of such a protocol. Consider parties being split into three groups P, Q, and R such that $|P| \leq t$, $|Q| \leq t$, and $|R| = \max(1, n - 2t)$. We suppose the designated sender is in Q.

We set $\beta = \Delta - \alpha$. Recall that under a synchrony assumption, each message can take anywhere from 0 to Δ time to arrive at its destination. We consider four worlds as follows.

World 0.

Setup. Parties in $P \cup Q$ are honest while parties in R have crashed. The honest sender sends input value b. All parties start at 0.

Message schedule. All messages sent among parties in $P \cup Q$ are delivered instantaneously.

Execution and views of honest players. This execution satisfies (i), so all honest parties commit before time α. By the validity property of Byzantine Broadcast, all parties in $P \cup Q$ commit b before time α.

World 1.

Setup. Parties in $P \cup Q$ are honest while parties in R have crashed. The honest sender sends input value b. All parties start at β.

Message schedule. All messages sent among parties in $P \cup Q$ are delivered instantaneously.

Execution and views of honest players. In an unsynchronized start model, since the starting states of parties do not depend on when they start, this execution is indistinguishable to World 0. Hence, all parties in $P \cup Q$ commit b before time $\alpha + \beta$.

World 2.

Setup. Parties in $Q \cup R$ are honest while parties in P have crashed. The honest sender sends input value $b' \neq b$. All parties start at 0.

Message schedule. All the messages sent among parties in $Q \cup R$ are delivered instantaneously.

Execution and views of honest players. This execution satisfies (ii), so all honest parties commit before time $\beta + \Delta = 2\Delta - \alpha$. By the validity property, all parties in $Q \cup R$ commit b' before time $\beta + \Delta$.

World 3.

Setup. Parties in $P \cup R$ are honest while parties in Q (which includes the designated sender) are Byzantine. Parties in P start at time β while parties in R start at time 0.

Message schedule. The parties in Q perform a split-brain attack where they behave like in World 2 towards parties in R, and behave like in World 1 towards parties in P. Hence, we will denote each brain of Q as Q_1 and Q_2 such that Q_1 only communicates with P and Q_2 only communicates with R.

Parties in Q_1 send messages to parties in P only after time β. Messages sent between $P \cup Q_1$ are delivered instantaneously (like in World 1). Messages between $Q_2 \cup R$ are delivered instantaneously (like in World 2). In addition, all messages sent across R and P are delayed by Δ. Messages received by a party in Q_1 from P are forwarded to its other brain replica in Q_2.

Execution and views of honest parties. Since messages from R to P are delayed to the maximum Δ time and Q_1 behaves exactly as in World 1, the views of parties in P are exactly the same as in World 1 until time Δ. Hence, parties in P commit b before time $\alpha + \beta = \Delta$.

Similarly, the view of R until time $\beta + \Delta$ is exactly the same as World 2 since P does not send any message until time β and messages from P to R are delayed by Δ. Q_2 receives (via Q_1) the same set of messages from P as Q in World 2 did. So, Q_2 behaves towards R just like Q in World 2 did. Hence, parties in R commit b' before time $\beta + \Delta = 2\Delta - \alpha$. This leads to a violation of the agreement property between P and R. □

3.4 Optimal Optimistic Responsiveness with 2Δ-synchronous Latency

We first present a simple synchronous consensus protocol that achieves optimal optimistic responsiveness when the optimistic commit does not require a quorum of all replicas. In a *synchronous commit*, a replica commits 2Δ time after voting (recall that Δ is an upper bound on the maximum network delay) if an equivocating proposal has not been detected. In a *responsive commit*, a replica

can commit immediately, i.e., without waiting for the 2Δ time period, if a sufficient number of replicas have voted for the block and no equivocation has been detected. For every block, a replica opportunistically waits to commit using either of the commit rules.

Recall that $\delta \leq \Delta$ is the actual network delay. If a "leader" is honest then no matter what the adversary does, the system can commit a block in time $2\Delta + O(\delta)$. But if there are $> 3n/4$ honest replicas along with an honest leader, then the system can commit in time $O(\delta)$ (in an optimistically responsive manner).

Why does our protocol perform better than protocols in the slow-path–fast-path paradigm? The general strategy employed in the protocols with back-and-forth slow-path–fast-path paradigm is to start on one of the two paths, say, the slow path. When the optimistic conditions are met, an explicit switch is performed to move to the fast path. Similarly, when a lack of progress is detected on the fast path, the protocol makes another switch to the slow path. The explicit switch between the paths incurs a latency of at least Δ in all of these protocols.

Under minority Byzantine faults, the adversary can attack the above strategy to worsen the commit latency compared to a protocol with a single slow path. For example, when the protocol is on the slow path, the adversary responds promptly and the replicas receive $> 3n/4$ responses thereby triggering a switch to fast path. Once on the fast path, the adversary stops responding and prevents progress. This forces an explicit switch to the slow path again. Under this attack, a single decision can incur a latency of 4Δ if the replicas are on the fast path and then switch to the slow path to commit. On the fast path, replicas never commit if the adversary does not respond.

Our protocol avoids this concern by avoiding an explicit switch. Instead, both paths are active simultaneously. As a result, when the leader is honest, the commit latency is 2δ during optimistic executions and 2Δ otherwise.

View-based execution. Like PBFT [32], our protocol progresses through a series of numbered *views* with each *view* coordinated by a distinct leader. Views are represented by non-negative integers with 0 being the first view. The leader of the current view v is determined by (v mod n). Within each view, also called the steady state, the leader is expected to propose values and keep making progress by committing client requests at increasing *heights*. An honest replica participates in any one view at a time and moves to a higher numbered view when the current view fails to make progress. If the replicas detect equivocation or lack of progress in a view, they initiate a *view-change* by blaming the current leader. When a quorum of replicas have blamed the current leader, they perform a view-change and replace the faulty leader.

Blocks and block format. Client requests are batched into *blocks*. Each block references its predecessor with the exception of the genesis block which has no predecessor. We call a block's position in the chain as its height. A block B_k at height k has the format, $B_k := (b_k, H(B_{k-1}))$ where b_k denotes a proposed value at height k, B_{k-1} is the block at height $k-1$ and $H(B_{k-1})$ is the hash digest of B_{k-1}. The predecessor for the genesis block is $\perp$. A block B_k is said to be *valid* if (1) its predecessor block is valid, or if $k = 1$, predecessor is $\perp$, and (2) client requests in the block meet application-level validity conditions and are consistent with its chain of requests in ancestor blocks.

Block extension and equivocation. A block B_k *extends* a block B_l ($k \geq l$) if B_l is an ancestor of B_k. Note that a block B_k extends itself. Two blocks B_k and $B'_{k'}$ proposed in the same view *equivocate* one another if they are not equal to *and* do not extend one another.

Block certificates. A *block certificate* represents a set of signatures on a block by a quorum of replicas. Given a ratio $0 \leq \alpha < 1$, a block B_k and a view v we denote by $\mathcal{C}_v^{\alpha}(B_k)$ a set of $\lfloor \alpha n \rfloor + 1$ signatures from different replicas on block B_k signed in view v. In this section, we will use *synchronous certificate* where $\alpha = 1/2$, and *responsive certificate* where $\alpha = 3/4$. Whenever the distinction between the two is not important, we will represent the certificates by $\mathcal{C}_v(B_k)$ and ignore the superscript α. In the next section, we will also use *full certificates* which require all n replicas to sign.

Chain certificates. We use the notion of *chain certificates* to compare different chains when replicas receive many of them. Most earlier protocols (e.g., HotStuff [104] or Sync HotStuff [7]) compared certified chains using just the views and heights. However, in our protocol, there are two types of certificates, a responsive certificate and a synchronous certificate, and hence, comparing them is subtle. As we will see, the rank of a chain will be completely determined by the block with the highest synchronous certificate from the largest view and the block's ancestors' highest responsive certificate in this view. A chain certificate comprises of a pair of certificates $\mathcal{C}_v^{3/4}(B_k)$ and $\mathcal{C}_v^{1/2}(B_\ell)$. Each element in the pair is either a block certificate or $\perp$ such that (i) if either of them are not $\perp$, both certificates are from the same view, (ii) if not $\perp$'s, the first certificate has threshold 3/4, the second has threshold 1/2, and (iii) block B_ℓ extends block B_k, if $\mathcal{C}_v^{3/4}(B_k)$ is not $\perp$.

Ranking chain certificates. Given two chain certificates $\mathcal{CC} = (\mathcal{C}_v^{3/4}(B_k), \mathcal{C}_v^{1/2}(B_\ell))$ and $\mathcal{CC}' = (\mathcal{C}_{v'}^{3/4}(B_{k'}), \mathcal{C}_{v'}^{1/2}(B_{\ell'}))$, they are first ranked by views, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $v < v'$. While moving from view v to any higher view, our protocol ensures that if a certified block B_k is committed in view

v, then all honest replicas lock on a chain certificate that extends B_k. Hence, a certificate chain produced in a higher view will always include B_k. Said another way, a certificate chain $\mathcal{CC}'$ in a higher view will extend B_k; if it does not, it must be the case that B_k was not committed by any honest replica in view v. Thus, it is safe to extend $\mathcal{CC}'$.

For chain certificates in the same view v, they are first ranked based on the height of the responsive certificate, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $k < k'$. In our protocol, we ensure that if there exists a responsive certificate for a block $B_{k'}$ in view v, i.e., $\mathcal{C}_v^{3/4}(B_{k'})$ exists, there cannot exist a responsive certificate for a conflicting block at any height in view v. Thus, if there is a responsive certificate for B_k in view v, then $B_{k'}$ must extend B_k. Moreover, we also ensure that if $\mathcal{C}_v^{3/4}(B_k)$ exists, no replica will have synchronously committed on an equivocating block B_ℓ with certificate $\mathcal{C}_v(B_\ell)$. Thus, any equivocating chain with chain certificate $\mathcal{CC}$ will not contain committed blocks that are not extended by $\mathcal{CC}'$.

Finally, if both chain certificates are in the same view v and have a common responsive certificate in the view (or both do not have a responsive certificate), the chain certificates are ranked by the heights of synchronous certificates, i.e., $\mathcal{CC} < \mathcal{CC}'$ if $\ell < \ell'$. Our protocol ensures that if B_k is committed synchronously in view v, then there does not exist an equivocating certified block. Thus, if equivocating $\mathcal{C}_v^{1/2}(B_\ell)$ and $\mathcal{C}_v^{1/2}(B_{\ell'})$ exist, both B_ℓ and $B_{\ell'}$ could not have been committed. To ease the rule in the case where they do not equivocate and one chain certificate extends the other, we select higher of the two.

Thus, given two chain certificates $\mathcal{CC} = (\mathcal{C}_v^{3/4}(B_k), \mathcal{C}_v^{1/2}(B_\ell))$ and $\mathcal{CC}' = (\mathcal{C}_{v'}^{3/4}(B_{k'}), \mathcal{C}_{v'}^{1/2}(B_{\ell'}))$, we say $\mathcal{CC} < \mathcal{CC}'$ if:

1. $v < v'$ (the chain certificates are first ranked by view),
2. $v = v'$ and $k < k'$ (secondly by responsive certificates),
3. $v = v'$ and $k = k'$ and $\ell < \ell'$ (finally by sync certificates).

The above comparison uses numerical value -1 to represent a $\perp$.

Tip of a chain certificate. The tip of a chain certificate is the highest block in the chain. Given a $\mathcal{CC} = (\mathcal{C}^{3/4}(B_k), \mathcal{C}^{1/2}(B_\ell))$, if $\mathcal{C}^{1/2}(B_\ell) \neq \perp$ then define $\mathsf{tip}(\mathcal{CC}) = B_\ell$, otherwise define $\mathsf{tip}(\mathcal{CC}) = B_k$.

Updating chain certificates. Each replica stores $\mathcal{CC}$, the highest chain certificate it has ever received. Any time a new block certificate is received, the replica updates its highest ranked chain

certificate using the comparison rule described earlier.

3.4.1 Steady State Protocol

Our protocol executes the following steps in iterations within a view v. Refer Figures 3.1 and 3.2.

Let v be the view number and replica L be the leader of view v. While in view v, a replica r runs the following protocol:

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_v(B_{k-1})$, it broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ where B_k extends B_{k-1}.
2. **Vote.** Upon receiving the first proposal $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ with a valid view v certificate for a block at height $k-1$ (not necessarily from L) where B_k extends B_{k-1}, if no leader equivocation is detected, forward the proposal to all replicas, broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v\rangle_r$, set $\mathsf{commit\text{-}timer}_{v,k}$ to 2Δ, and start counting down.
3. **(Non-blocking) Commit rules.** Replica r commits block B_k using either of the following rules if r is still in view v:
   (a) *Responsive commit.* On receiving $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$, commit B_k and all its ancestors immediately. Abort $\mathsf{commit\text{-}timer}_{v,k}$.
   (b) *Synchronous commit.* If $\mathsf{commit\text{-}timer}_{v,k}$ reaches 0, commit B_k and all its ancestors.
4. **(Non-blocking) Blame and quit view.**
   - *Blame if no progress.* For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v\rangle_r$.
   - *Quit view on $t+1$ blame messages.* Upon gathering $t+1$ distinct $\langle \mathsf{blame}, v\rangle_r$ messages, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle$ along with $t+1$ blame messages where $\mathcal{CC}$ is the highest ranked chain certificate known to r. Abort all view v timers, and quit view v.
   - *Quit view on detecting equivocation.* If leader equivocation is detected, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle_r$ along with the equivocating proposals, abort all view v timers, and quit view v.

Figure 3.1: Steady state protocol for optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized quorum.

Propose. The leader L of view v proposes a block $B_k := (b_k, H(B_{k-1}))$ by broadcasting $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$. The proposal contains a block at height-k extending a block B_{k-1} at height $k-1$, the view number v, and a view-v certificate for B_{k-1}. The leader makes such a proposal as soon as it receives a view-v certificate for B_{k-1}. The first view-v certificate is obtained during the view-change process as will be described in the next subsection.

Vote. When a replica r receives the first proposal for B_k either from L or through some other replica, if r hasn't received a proposal for an equivocating block, i.e., it has not detected a leader

Let L and L' be the leaders of view v and $v+1$, respectively. Each replica r runs the following steps.

i) **Status.** Wait for 2Δ time. Until this time, if a replica receives any chain certificates, the replica updates its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to be the highest ranked chain certificate at the end of the 2Δ wait. Send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ to L'. Enter view $v+1$.

ii) **New-view.** The new leader L' waits for 2Δ time after entering view $v+1$. L' broadcasts $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}_{v+1} \rangle_{L'}$, where lock_{v+1} is the highest ranked chain certificate known to L' after this wait.

iii) **First vote.** Upon receiving the first $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$, if $\mathsf{lock}_{v+1} \leq \mathsf{lock}'$, then broadcast $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and $\langle \mathsf{vote}, \mathsf{tip}(\mathsf{lock}'), v+1 \rangle_r$.

Figure 3.2: View-change protocol for optimal optimistic responsiveness with 2Δ-synchronous latency and $> 3n/4$-sized quorum.

equivocation in view v, it broadcasts a vote for B_k in the form of $\langle \mathsf{vote}, B_k, v \rangle_r$, and forwards the proposal to all replicas. It also starts a synchronous $\mathsf{commit\text{-}timer}_{k,v}$ and sets it to 2Δ.

Observe that the certificate in the proposal need not be the same as the certificate that replica r has obtained. Specifically, replica r can vote for a proposal containing a synchronous certificate for the previous block even if it holds a responsive certificate for the same block, and vice versa.

Commit. The protocol includes two commit rules and the replica commits using the rule that is triggered first. In a responsive commit, a replica commits block B_k and its ancestors immediately if the replica receives $> 3n/4$ votes for B_k in view v. Note that a responsive commit doesn't depend on the commit-timer and Δ, and a replica can commit at the actual speed of the network (δ). When a replica's $\mathsf{commit\text{-}timer}_{v,k}$ for B_k expires in view v, the replica synchronously commits B_k and all its ancestors. When a replica commits B_k, it aborts commit-timers for all its ancestors.

The commit step is non-blocking and it does not affect the critical path of progress. The leader can make a proposal for the next block as soon as it receives a certificate for the previous block independent of whether replicas have committed blocks for previous heights.

Note that if an honest replica commits a block B_k in view v using one of the rules, it is not necessary that all honest replicas commit B_k in view v using the same rule, or commit B_k at all. Some Byzantine replicas may decide to send votes to only a few honest replicas causing some honest replicas to commit using a responsive rule whereas some others using a synchronous rule. A Byzantine leader could send an equivocating block to some honest replicas and prevent them from committing. The protocol ensures safety despite all inconsistencies introduced by Byzantine replicas.

Blame and quit view. A view-change is triggered when replicas observe lack of progress or an equivocating proposal from the current leader. If an honest replica learns an equivocation, it broadcasts $\langle \mathsf{blame}, v \rangle_r$ message along with the equivocating proposals and quits view v. The equivocating proposals serve as a proof of misbehavior and all honest replicas blame the leader to trigger a view-change. To ensure progress, the leader is expected to propose at least one block every 2Δ time that trigger the replica's vote. Otherwise, replicas blame the current leader. Replicas quit view v when they receive $t+1$ blame messages, detect equivocation. On quitting view v, replica r broadcasts $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC} \rangle_r$ where $\mathcal{CC}$ is the highest ranked chain certificate known to r.

We now provide some intuition on why either of these commit rules are safe within a view. We discuss safety across views in the subsequent section.

Why does a responsive commit ensure safety within a view? Consider an honest replica r that responsively commits a block B_k at time τ. This is because it received $\lfloor 3n/4 \rfloor + 1$ votes for B_k by time τ and it did not observe any equivocation until then. It is easy to see that if there exists $\lfloor 3n/4 \rfloor + 1$ votes for B_k, no other equivocating block $B'_{k'}$ at any height k' can be committed responsively due to a simple quorum intersection argument. Under a minority corruption, any two quorums of size $\lfloor 3n/4 \rfloor + 1$ intersect in $t+1$ replicas out of which at least one replica is honest. This honest replica will not vote for two equivocating blocks.

A synchronous commit of an equivocating block cannot happen due to the following reason. Since replica r hasn't received an equivocation until time τ, no replica has voted for an equivocating proposal until time $\tau - \Delta$. Hence, their synchronous 2Δ window for committing an equivocating block ends at time $> \tau + \Delta$. A commit for B_k at time τ implies that some honest replica must have voted and forwarded the corresponding proposal before time τ and this will arrive by time $\tau + \Delta$ at all honest replicas. This will prevent any other replica from committing an equivocating block. Observe that a responsive commit does not imply that an equivocating block $B'_{k'}$ will not be certified; hence, during a view-change, we need to be able to carefully extend the chain that contains a block that has been committed by some other replica.

Why does a synchronous commit ensure safety within a view? Consider an honest replica r that votes for a block B_k at time τ and commits at time $\tau + 2\Delta$ because it did not observe an equivocation until then. This implies (i) all honest replicas have received B_k by time $\tau + \Delta$, and (ii) no honest replica has voted for an equivocating block by time $\tau + \Delta$. Due to the rules of voting, no honest replica will vote for an equivocating block in this view after time $\tau + \Delta$ ruling out an equivocating commit through either of the two rules.

3.4.2 View-change Protocol

The view-change protocol is responsible for replacing a possibly faulty leader with a new leader to maintain liveness. In the process, it needs to maintain safety of any commit that may have happened in the previous views.

Status. After quitting view v, a replica waits for 2Δ time before entering view $v+1$. The 2Δ wait ensures that all honest replicas receive a certificate for a block B_k before entering view $v+1$ if some honest replica committed B_k in view v. This is critical to maintain the safety of the commit in view v. The replica updates its chain certificate $\mathcal{CC}$ to the highest possible rank and sets lock_{v+1} to $\mathcal{CC}$. It then sends lock_{v+1} to the next leader L' via a $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$.

New-view. Leader L' waits 2Δ time after entering view $v+1$ to receive a status message from all honest replicas. Based on these status messages, L' picks the highest ranked chain certificate lock'. It creates a $\mathsf{new\text{-}view}$ message $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ and sends it to all honest replicas. The highest ranked chain certificate across all honest replicas at the end of view v helps an honest leader to appropriately send a new-view message that will be voted upon by all honest replicas and maintain the liveness of the protocol.

First vote. Upon receiving a $\langle \mathsf{new\text{-}view}, v+1, \mathsf{lock}' \rangle_{L'}$ message, if the certified chain certificate lock' has a rank no lower than r's locked chain certificate lock_{v+1}, then it forwards the $\mathsf{new\text{-}view}$ message to all replicas and broadcasts a vote for it.

Next, we provide some intuition on how the view-change protocol ensures safety across views and liveness.

Why do replicas lock on chains extending committed blocks before entering the next view? In this protocol, we use locks to ensure safety. The protocol guarantees that if an honest replica commits a block (through either rule), then at the end of the view all honest replicas will lock on a chain certificate that extends the committed block. At the start of the next view, when the leader sends a lock through the new-view message, by testing whether this lock is higher than the lock stored locally, an honest replica ensures that only committed blocks are extended.

What ensures that replicas lock on chains extending committed blocks before entering the next view? Suppose an honest replica r responsively commits a block B_k in view v at time τ. Notice that no honest replica has entered view $v+1$ by time $\tau + \Delta$; otherwise, replica r must

have received blame certificate by time τ due to the 2Δ wait in the status step. In addition, replica r sends a quit-view message containing the highest certified chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k when quitting view v. $\mathcal{CC}$ reaches all honest replicas by the time an honest replica enters view $v+1$. In the proof, we show there does not exist an equivocating chain certificate $\mathcal{CC}'$ that ranks higher than $\mathcal{CC}$. Thus, all honest replicas lock on $\mathcal{CC}$ or higher before entering view $v+1$.

If replica r synchronously commits B_k in view v at time τ, then replica r voted for B_k at time $\tau - 2\Delta$. It did not detect an equivocation or blame certificate by time τ. This implies all honest replicas will vote for B_k at time $\tau - \Delta$ and receive $\mathcal{C}_v^{1/2}(B_k)$ by time τ. As noted earlier, there does not exist an equivocating certificate in view v during synchronous commit. Hence, all honest replicas will lock on $\mathcal{CC}$ containing $\mathcal{C}_v^{1/2}(B_k)$ before entering view $v+1$.

How does the protocol ensure liveness? The protocol ensures liveness by allowing a new honest leader to *always* propose a block that will be voted for by all honest replicas. All honest replicas send their locked chain certificate to the next leader L' at the start of the new view in a status message. L' could be lagging and enter $v+1$ Δ time after other replicas. Thus, it waits 2Δ time to collect chain certificates from all honest replicas. If L' is honest, it extends the highest ranked chain certificate lock'. This suffices to ensure that all honest replicas vote on its proposal, in turn, ensuring liveness when the leader is honest. In the new view, as long as the leader keeps proposing valid blocks, honest replicas will vote and keep committing new blocks.

3.4.3 Safety and Liveness

We say a *block B_k is committed directly in view v* if an honest replica successfully runs the responsive commit rule 3(a) or the synchronous commit rule 3(b) on block B_k. Similarly, we say a *block B_k is committed indirectly* if it is a result of directly committing a block B_ℓ ($\ell > k$) that extends B_k but is not equal to B_k.

We say that a replica is *in view v at time τ* if the replica executes the *Enter view v* of Step i) in Figure 3.2 by time τ and did not execute any *Quit view* of Step Step 2 in Figure 3.1 for view v at time τ or earlier.

Claim 5. *If a block B_k is committed directly in view v using the responsive commit rule, then a responsive certificate for an equivocating block $B'_{k'}$ in view v does not exist.*

Proof. If replica r commits B_k due to the responsive commit rule in view v, then r must have

received $\lfloor 3n/4 \rfloor + 1$ votes, i.e., $\mathcal{C}_v^{3/4}(B_k)$, forming a quorum for B_k in view v. A simple quorum intersection argument shows that a responsive certificate for equivocating block $B'_{k'}$ cannot exist. □

Claim 6. *If a block B_k is committed directly in view v using the responsive commit rule, then there does not exist a chain certificate $\mathcal{CC}$ in view v, such that $\mathcal{CC} > (\mathcal{C}_v^{3/4}(B_k), \perp)$ where a block in $\mathcal{CC}$ equivocates B_k.*

Proof. By Claim 5, no equivocating block can have a responsive block certificate. So all responsive block certificates must extend B_k. Since we assume that $\mathcal{CC} > (\mathcal{C}_v^{3/4}(B_k), \perp)$ then it must be that either $\mathcal{CC}$ is of the form $(\mathcal{C}_v^{3/4}(B_k), \mathcal{C}_v^{1/2}(B_\ell))$ and by definition B_ℓ extends B_k, or $\mathcal{CC}$ is of the form $(\mathcal{C}_v^{3/4}(B_{k'}), \mathcal{C}_v^{1/2}(B_{\ell'}))$ where $B_{k'}$ extends B_k and again by transitivity $B_{\ell'}$ must extend B_k. □

Claim 7. *If a block B_k is committed directly in view v using the synchronous commit rule, then a block certificate for an equivocating block $B'_{k'}$ does not exist in view v.*

Proof. Suppose replica r directly commits block B_k at time τ using the synchronous commit rule. So replica r voted and forwarded the proposal for B_k at time $\tau - 2\Delta$ and its $\mathsf{commit\text{-}timer}_{v,k}$ expired without detecting equivocation. By synchrony assumption, all replicas receive the forwarded proposal for B_k by time $\tau - \Delta$. Since they do not vote for equivocating blocks, they will not vote for $B'_{k'}$ received at time $> \tau - \Delta$. Moreover, no honest replica must have voted for an equivocating block at time $\leq \tau - \Delta$. Otherwise, replica r would have received the equivocating proposal by time τ and it wouldn't have committed. Since no honest replica votes for an equivocating block, $B'_{k'}$ will not be certified. □

Claim 8. *If a block B_k is committed directly in view v using the responsive commit rule, then all honest replicas receive a chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k before entering view $v+1$.*

Proof. Suppose replica r directly commits block B_k at time τ using the responsive commit rule. No honest replica r' has entered view $v+1$ at time $\leq \tau + \Delta$; otherwise replica r' must have sent a blame certificate at time $\leq \tau - \Delta$ (due to 2Δ wait in the status step) and r must receive the blame certificate at time $\leq \tau$ and wouldn't commit.

By Claim 6, there doesn't exists a conflicting chain certificate $\mathcal{CC}' > (\mathcal{C}_v^{3/4}(B_k), \perp)$ such that $\mathsf{tip}(\mathcal{CC}')$ does not extend B_k. Thus, the highest ranked chain certificate $\mathcal{CC}$ in view v must have $\mathsf{tip}(\mathcal{CC})$ extend B_k. Replica r sends $\mathcal{CC}$ when it quits view v after time τ.

Let τ' be the time in which replica r' enters view $v+1$ (with $\tau' > \tau + \Delta$). Replica r must have received a blame certificate between time τ and $\tau' - \Delta$ and sent a quit-view message containing $\mathcal{CC}$ which arrives at replica r' at time $\leq \tau'$. Hence, all honest replicas receive $\mathcal{CC}$ before entering view $v+1$. □

Claim 9. *If a block B_k is directly committed in view v at time τ using the synchronous commit rule, then all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$.*

Proof. We will prove that if a block B_k is directly committed in view v at time τ using the synchronous commit rule, then (i) all honest replicas are in view v at time $\tau - \Delta$, (ii) all honest replicas vote for B_k at time $\leq \tau - \Delta$, and (iii) all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. Part (iii) is the desired claim.

Suppose honest replica r synchronously commits B_k at time τ in view v. It votes for block B_k at time $\tau - 2\Delta$. Thus, replica r entered view v at time $\leq \tau - 2\Delta$. Due to the 2Δ wait before sending a status message, replica r must have sent a blame certificate or equivocating blocks at time $\leq \tau - 4\Delta$ which arrives all honest replicas at time $\leq \tau - 3\Delta$. Hence, all honest replicas enter view v at time $\leq \tau - \Delta$ (again due to 2Δ wait in the status step). Also, observe that no honest replica has quit view v at time $\leq \tau - \Delta$. Otherwise, replica r hears of blame certificate or equivocation at time $\leq \tau$. This proves part (i).

Replica r received a proposal for B_k which contains $\mathcal{C}_v(B_{k-1})$ at time $\tau - 2\Delta$. Thus, replica r's vote and forwarded proposal for B_k arrives all honest replicas by time $\tau - \Delta$. No honest replica has voted for an equivocating block at time $\leq \tau - \Delta$; otherwise replica r would have received an equivocation at time $\leq \tau$. Thus, all honest replicas will vote for B_k at time $\leq \tau - \Delta$. This proves part (ii).

The votes from all honest replicas will arrive at all honest replicas by time $\leq \tau$. By part(i) of the claim and 2Δ wait in the status step, honest replicas do not enter view $v+1$ at time $\leq \tau + \Delta$. Thus, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. □

Lemma 10. *If an honest replica directly commits a block B_k in view v, then: (i) all honest replicas have lock_{v+1} such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k, (ii) for any chain certificate $\mathcal{CC}'$ that the adversary can create and any honest lock lock_{v+1}, either $\mathcal{CC}' < \mathsf{lock}_{v+1}$ or $\mathsf{tip}(\mathcal{CC}')$ extends B_k.*

Proof. If B_k is committed using the responsive commit rule, then by Claim 8, all honest replicas receive $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k before entering view $v+1$ and by Claim 6 there doesn't

exist chain certificate $\mathcal{CC}'$ such that $\mathcal{CC}' > (\mathcal{C}_v(B_k), \perp)$ and $\mathcal{CC}'$ equivocates B_k. Similarly, If B_k is committed using the synchronous commit rule, then by Claim 9, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$ and by Claim 7, there doesn't exists a view v certificate that equivocates B_k. Since, honest replicas lock on highest ranked chain certificate, $\mathsf{tip}(\mathsf{lock}_{v+1})$ must extend B_k. By similar argument, any $\mathcal{CC}'$ that an adversary creates either has $\mathcal{CC}' < \mathsf{lock}_{v+1}$ or $\mathsf{tip}(\mathcal{CC}')$ extends B_k. □

The following lemma considers safety of directly committed blocks across views.

Lemma 11 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, and $\mathcal{C}_{v'}(B_{k'})$ is a view $v' > v$ block certificate, then $B_{k'}$ extends B_k. Moreover, all honest replicas have $\mathsf{lock}_{v'}$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. The proof is by induction on views $v' > v$. For a view v', we prove that if $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ exists then it must extend B_k. A simple induction shows that all later block certificates must also extend $\mathsf{tip}(\mathsf{lock}')$, this follows directly from the vote rule in Figure 3.1 step 7.

For the base case, where $v' = v+1$, the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from Lemma 10 because the only way such a block can be certified is some honest votes for it. However, all honest are locked on a block that extends B_k and a chain certificate with a higher rank for an equivocating block does not exist. Thus, no honest replica will first vote (Figure 3.2 step iii)) for a block that does not extend B_k. The second part follows directly from Lemma 10.

Given that the statement is true for all views below v', the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from the induction hypothesis because the only way such a block can be certified is if some honest votes for it. An honest party with a lock lock will vote only if $\mathsf{tip}(\mathsf{lock}_{v'})$ has a valid block certificate and $\mathsf{lock} \geq \mathsf{lock}_{v'}$. Due to Lemma 10 and the induction hypothesis on all block certificates of view $v < v'' < v'$ is must be that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}))$ extends B_k. □

Theorem 12 (Safety)**.** *Honest replicas do not commit conflicting blocks for any height ℓ.*

Proof. Suppose for contradiction that two distinct blocks B_ℓ and B'_ℓ are committed at height ℓ. Suppose B_ℓ is committed as a result of B_k being directly committed in view v and B'_ℓ is committed as a result of $B'_{k'}$ being directly committed in view v'. This implies B_k extends B_ℓ and $B'_{k'}$ extends B'_ℓ. Without loss of generality, assume $v \leq v'$; if $v = v'$, further assume $k \leq k'$. If $v = v'$ and $k \leq k'$, by Claim 6 and Claim 7, $B'_{k'}$ extends B_k. Similarly, if $v < v'$, by Lemma 11, $B'_{k'}$ extends B_k. Thus, $B'_\ell = B_\ell$. □

Theorem 13 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. In a view, a leader has to propose at least p blocks that trigger honest replicas votes in $(2p + 4)\Delta$ time. As long as the leader proposes at least p valid blocks, honest replicas will keep voting and committing proposed blocks. If the Byzantine leader equivocates or proposes less than p blocks, a view-change will occur. Eventually, there will be an honest leader due to round-robin leader election.

Next, by Lemma 10, all honest replicas lock on a highest certified chain before entering a new view. The leader may enter the new view Δ time later than others; hence need to wait for 2Δ before proposing. Due to 2Δ wait, the new leader receives the highest locked certified chains from all honest replicas. If the leader is honest, the leader will extend upon the tip of the highest ranked certified chain. Honest replicas will vote for the new block since the lock sent by the leader is at least as large as their lock. Moreover, the honest leader doesn't equivocate and keeps proposing at least p blocks. This prevents forming a blame certificate to cause view-change and all honest replicas will keep committing new blocks. □

3.5 Optimal Optimistic Responsiveness with Δ-synchronous Latency

Recall that our lower bound in Section 3.3 showed that we cannot have the following two commit latencies simultaneously: (i) a responsive commit with $O(\delta)$ latency where $\max(1, n-2t)$ faults are tolerated in the responsive mode, and (ii) a synchronous commit with $< 2\Delta$ latency simultaneously. The previous section showed a protocol when at least one fault is tolerated in the responsive commit. In this section, we will present a protocol with a synchronous latency of $\Delta + O(\delta)$ when no faults are tolerated in the responsive commit. For a synchronous commit, an honest replica commits a block in $\Delta+O(\delta)$ time after receiving a valid proposal for the block if no equivocating proposals are received and $t+1$ replicas have voted. A responsive commit completes immediately when a replica receives *acknowledgments* for a block from *all* replicas and no equivocation has been detected. The protocol has a commit latency of 2δ as long as all replicas are behaving honestly and responding promptly.

Unlike the protocol in the previous section where a replica immediately votes for a valid proposal, in this protocol, a replica sends an ack for the proposed block immediately and votes only if it does not detect any equivocation Δ time after its ack. Using an ack message to obtain Δ latency under

an honest leader was proposed by Abraham et al. [9]. We augment this idea to use a set of $2t+1$ signed ack messages to obtain responsiveness simultaneously. The $2t+1$ signed acks from the same view for a block B_k is called a *full certificate* and represented as $\mathcal{C}_v^f(B_k)$. As before, we call a set of $t+1$ signed vote messages for B_k as *synchronous certificate* and represent it as $\mathcal{C}_v^{1/2}(B_k)$. Whenever the distinction is not important, we represent certificates as $\mathcal{C}_v(B_k)$. Later in the section, we show that if there exists a certificate (either full or synchronous) for a block B_k in a view v, there cannot exist a certificate for an equivocating block in view v. Hence, we define a simple certificate ranking rule. Certified blocks are first ranked by views and then by height, i.e., (i) blocks certified in a higher view have a higher rank, and (ii) for blocks certified in the same view, a higher height implies a higher rank.

3.5.1 Protocol

The steady state protocol runs following steps within a view v.

Propose. The Leader L of view v proposes a block B_k by extending a highest certified block $\mathcal{C}_{v'}(B_{k-1})$ known to L. If the leader has just entered the view, it waits for 2Δ time to receive the highest certified blocks from all honest replicas in which case $v' < v$. Otherwise, the leader proposes as soon as it learns a certificate for the previous block proposed in the same view.

Ack. The protocol includes an additional ack step before voting. A replica r broadcasts an ack $\langle \mathsf{ack}, B_k, v \rangle$ for a proposed block B_k if (i) it hasn't detected any equivocation in view v, and (ii) $\mathcal{C}_{v'}(B_{k-1})$ has rank equal to or higher than its own locked block. Once replica r sends an ack, it starts a $\mathsf{vote\text{-}timer}_{v,k}$ initialized to Δ time and starts counting down. Replica r also broadcasts the received proposal.

Vote. When $\mathsf{vote\text{-}timer}_{v,k}$ for block B_k expires, if replica r hasn't heard of any equivocation in view v, it broadcasts a vote $\langle \mathsf{vote}, B_k, v \rangle$.

Commit. Replica r can commit either responsively or synchronously based on which rule is triggered first. A responsive commit is triggered when r receives $2t+1$ ack messages for B_k, i.e., $\mathcal{C}_v^f(B_k)$ and r commits B_k and all its ancestors immediately. Replica r stops $\mathsf{vote\text{-}timer}_{v,k}$ and broadcasts $\mathcal{C}_v^f(B_k)$ to all honest replicas. Similarly, replica r synchronously commits B_k along with its all ancestors when it receives $t+1$ vote messages for B_k, i.e., $\mathcal{C}_v^{1/2}(B_k)$. r also broadcasts $\mathcal{C}_v^{1/2}(B_k)$ to all replicas. Like before, both the commit paths are non-blocking and the leader can

Let v be the view number and replica L be the leader of the current view. A replica r runs the following protocol in iterations:

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_{v'}(B_{k-1})$, it broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$ where B_k extends B_{k-1}. If it is the first block in this view, i.e., $v' < v$, then it waits for an additional 2Δ time after entering the view before proposing the highest certified block received from the status step.
2. **Ack.** Upon receiving the first proposal $\langle \mathsf{propose}, B_k, v, \mathcal{C}_{v'}(B_{k-1})\rangle_L$ (not necessarily from L) at height k in view v, if $\mathcal{C}_{v'}(B_{k-1})$ is ranked greater than or equal to its locked block, forward the proposal to all replicas and broadcast an acknowledgment in the form of $\langle \mathsf{ack}, B_k, v\rangle$. Set $\mathsf{vote\text{-}timer}_{v,k}$ to Δ and start counting down.
3. **Vote**. If $\mathsf{vote\text{-}timer}_{v,k}$ reaches 0, send a vote for B_k in the form of $\langle \mathsf{vote}, B_k, v\rangle$.
4. **(Non-blocking) Commit.** Replicas can commit block B_k using either of the following rules:
   (a) *Responsive commit.* On receiving $2t+1$ acks for B_k, i.e., $\mathcal{C}^f_v(B_k)$ in view v, commit B_k and all its ancestors immediately. Stop $\mathsf{vote\text{-}timer}_{v,k}$ and notify the certificate $\mathcal{C}^f_v(B_k)$.
   (b) *Synchronous commit.* On receiving $t+1$ votes for B_k, i.e., $\mathcal{C}^{1/2}_v(B_k)$ in view v, commit B_k and all its ancestors immediately. Notify the certificate $\mathcal{C}^{1/2}_v(B_k)$ to all replicas.
5. **(Non-blocking) Blame and quit view.**
   - *Blame if no progress.* For $p > 0$, if fewer than p proposals trigger r's votes in $(3p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v\rangle_r$.
   - *Quit view on $t+1$ blame messages.* Upon gathering $t+1$ distinct $\langle \mathsf{blame}, v\rangle_r$ messages, broadcast them, abort all view v timers, and quit view v.
   - *Quit view on detecting equivocation.* If leader equivocation is detected, broadcast the equivocating proposals signed by L, abort all view v timers, and quit view v.

Figure 3.3: Steady state protocol for optimal optimistic responsiveness with Δ-synchronous latency and n-sized quorum.

keep proposing as soon as it learns a certificate for previous block.

3.5.2 View Change Protocol

Let L and L' be the leader of view v and $v+1$, respectively. Each replica r runs the following steps.

(i) **Status.** Wait for 2Δ time. Pick the highest certified block $B_{k'}$ with certificate $\mathcal{C}_{v'}(B_{k'})$. Lock on $\mathcal{C}_{v'}(B_{k'})$, and send $\mathcal{C}_{v'}(B_{k'})$ to the new leader L'. Enter view $v+1$.

Figure 3.4: View-change protocol for optimal optimistic responsiveness with Δ-synchronous latency and n-sized quorum.

Blame and quit view step remains identical to the one in Figure 3.2.

Status. During this step, a replica r waits for 2Δ time and locks on the highest certified block $\mathcal{C}_{v'}(B_{k'})$ known to r. It forwards $\mathcal{C}_{v'}(B_{k'})$ to the next leader and enters next view. As shown in Lemma 17, the 2Δ wait ensures that all honest replicas lock on the highest-certified block corresponding to a commit at the end of the view, which, in turn, is essential to maintain the safety of the protocol. The status message along with the accompanying 2Δ wait in the propose step ensures liveness, i.e., it ensures that an honest leader proposes a block that extends locks held by all honest replicas and hence will be voted upon by all honest replicas.

Next, we provide some intuition on why either of these commit rules are safe within a view.

Why does a responsive commit ensure safety within a view? A replica commits a block B_k responsively only when it receives acks from all replicas which includes all honest replicas. This implies no honest replicas will either ack or vote for an equivocating block $B'_{k'}$ at any height k'. Hence, an equivocating block $B'_{k'}$ will neither receive $2t+1$ acks nor $t+1$ votes required for a commit.

Why does a synchronous commit ensure safety within a view? An honest replica r synchronously commits a block B_k at time t when it receives $t+1$ votes for B_k and hears no equivocation by time τ. This implies no honest replica has voted for an equivocating block $B'_{k'}$ by time $\tau - \Delta$. At least one honest replica r' sent an ack for B_k by time $\tau - \Delta$. r's ack arrives all honest replicas by time τ. Hence, honest replicas will neither ack nor vote for an equivocating block $B'_{k'}$ after time τ. This also prevents honest replicas from committing an equivocating block after time τ.

3.5.3 Safety and Liveness

We say a block B_k is committed *directly* in view v if any of the two commit rules are triggered for B_k. Similarly, a block B_k is committed *indirectly* if it is a result of directly committing a block B_ℓ ($\ell > k$) that extends B_k but is not equal to B_k.

Claim 14. *If an honest replica directly commits a block B_k in view v using the responsive commit rule, then there does not exist a certificate for an equivocating block in view v.*

Proof. If replica r commits B_k in view v using responsive commit rule, r must have received $2f+1$ acks, i.e., $\mathcal{C}^f_v(B_k)$. This implies all honest replicas have sent ack for B_k and no honest replica would send ack or vote for an equivocating block $B'_{k'}$ in view v. Since, a certificate for $B'_{k'}$ requires either

$2t+1$ acks for full certificate or at least one vote from an honest replica for synchronous certificate, a certificate for an equivocating block cannot exist. □

Claim 15. *If an honest replica directly commits a block B_k in view v using the synchronous commit rule, then there does not exist a certificate for an equivocating block in view v.*

Proof. Suppose replica r synchronously commits B_k in view v at time τ without detecting an equivocation. Observe that an equivocating responsive certificate does not exist since replica r would not ack two equivocating blocks. Hence, we need to only show that a synchronous equivocating certificate does not exist. We show it with the following two arguments. First, r votes for B_k at time $\leq \tau$ and sends an ack for B_k at time $\leq \tau - \Delta$. r's ack for B_k arrives all honest replicas by time τ. Hence, no honest replica will vote for an equivocating block $B'_{k'}$ at time $\geq \tau$. Second, no honest replica must have sent an equivocating ack at time $\leq \tau - \Delta$. Otherwise, replica r would not have committed. This also implies that no honest replica will vote for an equivocating block at time $\leq \tau$ (due to the Δ wait between ack and vote).

□

Lemma 16. *If an honest replica directly commits a block B_k in view v then, (i) there doesn't exist an equivocating certificate in view v, and (ii) all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$.*

Proof. Part(i) follows immediately from Claim 14 and Claim 15.

Suppose replica r commits B_k at time τ either responsively or synchronously. r notifies the certificate ($\mathcal{C}^f_v(B_k)$ or $\mathcal{C}^{1/2}_v(B_k)$) which arrives at all honest replicas at time $\leq \tau + \Delta$. Observe that no honest replica r' has entered view $v+1$ at time $\leq \tau + \Delta$. Otherwise, due to 2Δ wait before entering the new view, r' must have sent either equivocating or a blame certificate at time $\leq \tau - \Delta$; r must have received the blame certificate at time $\leq \tau$. It would have quit view and not committed. Hence, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. □

Lemma 17. *If an honest replica directly commits a block B_k in view v, then all honest replicas lock on a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$.*

Proof. By Lemma 16 part (ii), all honest replicas will receive $\mathcal{C}_v(B_k)$ before entering view $v+1$. By Lemma 16 part (i), no equivocating certificate exists in view v. Since replicas lock on the highest

certified block as soon as they enter the next view, all honest replicas lock on a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$. □

Lemma 18 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, then any certified block that ranks equal to or higher than $\mathcal{C}_v(B_k)$ must extend B_k.*

Proof. Any certified block $B'_{k'}$ in view v of rank equal to or higher than $\mathcal{C}_v(B_k)$ must extend B_k. Otherwise, $B'_{k'}$ equivocates B_k and by Lemma 16, $B'_{k'}$ cannot be certified in view v. For views higher than v, we prove the lemma by contradiction. Let S be the set of certified blocks that rank higher than $\mathcal{C}_v(B_k)$, but do not extend B_k. Suppose for contradiction $S \neq \emptyset$. Let $\mathcal{C}_{v^*}(B_{\ell^*})$ be a lowest ranked block in S. Also, note that if B_{ℓ^*} does not extend B_k, then B_{ℓ^*-1} does not extend B_k either.

For $\mathcal{C}_{v^*}(B_{\ell^*})$ to exist, some honest replica must vote for B_{ℓ^*} in view v either upon receiving a proposal $\langle \mathsf{propose}, B_{\ell^*}, v^*, \mathcal{C}_{v'}(B_{\ell^*-1})\rangle$ for $v' < v$ or $\langle \mathsf{propose}, B_{\ell^*}, v^*, \mathcal{C}_{v^*}(B_{\ell^*-1})\rangle$. If it is the former, then $\mathcal{C}_{v'}(B_{\ell^*-1})$ must rank higher than or equal to $\mathcal{C}_v(B_k)$. This is because due to Lemma 17 all honest replicas will have received a certified block that ranks higher than or equal to $\mathcal{C}_v(B_k)$ before entering view $v+1$. Moreover, replicas only lock on blocks of monotonically increasing ranks. However, since $v' < v^*$, the rank of $\mathcal{C}_{v'}(B_{\ell^*-1})$ is less than $\mathcal{C}_{v^*}(B_{\ell^*})$ by our certificate ranking rule. This contradicts the fact that $\mathcal{C}_{v^*}(B_{\ell^*})$ is a lowest ranked block in S. If it is the latter, then observe that $\mathcal{C}_{v^*}(B_{\ell^*-1})$ exists in view v^*. Again, this certificate is ranked higher than $\mathcal{C}_v(B_k)$ since $v^* > v$. Also, this certificate is ranked lower than $\mathcal{C}_{v^*}(B_{\ell^*})$ due to its height. Hence, this contradicts the fact that $\mathcal{C}_{v^*}(B_{\ell^*})$ is a lowest ranked block in S. □

Safety. The safety proof is identical to that of Theorem 12 except Lemma 18 needs to be invoked.

Liveness. The liveness proof is similar to that of Theorem 13.

3.6 Optimistic Responsiveness with Optimistically Responsive View-Change

The protocols in Section 3.4 and Section 3.5 are optimistically responsive in the steady-state. However, whenever a leader needs to be replaced, the view-change protocol must always incur a synchronous wait. This suffices if leaders are replaced occasionally, e.g., when a leader replica

crashes. However, in a democracy-favoring approach it may be beneficial to replace leaders after every block, or every few blocks. In such a scenario, the synchronous wait during view-change will increase the latency of the protocol. For example, the protocol in Section 3.4 waits at least 4Δ time during view-change to ensure that the new leader collects status from all honest replicas. Thus, in an execution where leaders are changed after every block, even when the leader is honest, this protocol requires at least $4\Delta + O(\delta)$ for one block to be committed even during optimistic executions, and requires at least 6Δ when $< 3n/4$ replicas are honest.

In this section, we present a protocol that is optimistically responsive in both the steady state as well as view-change. In a world with rotating honest leaders, when $> 3n/4$ replicas are honest, this protocol can commit blocks in $O(\delta)$ time and replace leaders in $O(\delta)$ time. When more than $n/4$ replicas are malicious with rotating honest leaders, the protocol still commits in $5\Delta + O(\delta)$ time.

3.6.1 Steady State Protocol

We make following modifications to the steady state protocol in Section 3.4 to support a responsive view-change. In a synchronous commit, a replica commits within 3Δ time after voting if no equivocation or blame certificate has been received. The additional Δ wait in the synchronous commit accounts for the responsive view-change that may occur before all honest replicas receive a certificate for committed blocks. The propose and vote steps remain identical. However, after voting for B_k, the commit-timer$_{v,k}$ is set to 3Δ time.

Pre-commit. The protocol includes an additional pre-commit step with two pre-commit rules active simultaneously. The pre-commit is identical to the commit step in the previous protocol. A replica pre-commits using the rule that is triggered first. In a responsive pre-commit, a replica r pre-commits a block B_k immediately when it receives $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$ in view v and broadcasts commit message via $\langle \mathsf{commit}, B_k, v \rangle_r$.

In a synchronous pre-commit, a replica pre-commits B_k when its commit-timer$_{v,k}$ reaches Δ and broadcasts $\langle \mathsf{commit}, B_k, v \rangle_r$.

Commit. In a responsive commit, a replica commits a block B_k immediately along with its ancestors when it receives $\lfloor 3n/4 \rfloor + 1$ commit messages for B_k. In a synchronous commit, a replica commits B_k and all its ancestors when its commit-timer$_{v,k}$ expires and it doesn't detect an equivocation or blame certificate. As before, the commit rules are non-blocking to rest of the execution.

Let v be the view number and replica L be the leader of the current view. While in view v, a replica r runs the following steps in iterations:

1. **Propose.** If replica r is the leader L, upon receiving $\mathcal{C}_v(B_{k-1})$, it broadcasts $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ where B_k extends B_{k-1}.
2. **Vote.** Upon receiving the first proposal $\langle \mathsf{propose}, B_k, v, \mathcal{C}_v(B_{k-1})\rangle_L$ with a valid view v certificate for B_{k-1} (not necessarily from L) where B_k extends B_{k-1}, forward the proposal to all replicas, broadcast a vote in the form of $\langle \mathsf{vote}, B_k, v\rangle_r$. Set $\mathsf{commit\text{-}timer}_{v,k}$ to 3Δ and start counting down.
3. **Pre-commit.** Replica r pre-commits B_k using one of the following rules if r is still in view v:
   (a) *Responsive Pre-commit.* On receiving $\lfloor 3n/4 \rfloor + 1$ votes for B_k, i.e., $\mathcal{C}_v^{3/4}(B_k)$ in view v, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v\rangle_r$.
   (b) *Synchronous Pre-commit.* If $\mathsf{commit\text{-}timer}_{v,k}$ reaches Δ, pre-commit B_k and broadcast $\langle \mathsf{commit}, B_k, v\rangle_r$ to all replicas.
4. **(Non-blocking) Commit.** If replica r is still in view v, r commits B_k using the following rules:
   (a) *Responsive Commit.* On receiving $\lfloor 3n/4 \rfloor + 1$ commit messages for B_k in view v, commit B_k and all its ancestors. Stop $\mathsf{commit\text{-}timer}_{v,k}$.
   (b) *Synchronous Commit.* If $\mathsf{commit\text{-}timer}_{v,k}$ reaches 0, commit B_k and all its ancestors.
5. **Yield.** Upon committing at least a block in view v, Leader L broadcasts $\langle \mathsf{yield}, v\rangle_L$ when it wants to renounce leadership.
6. **(Non-blocking) Blame and quit view.**
   - *Blame if no progress.* For $p > 0$, if fewer than p proposals trigger r's votes in $(2p+4)\Delta$ time in view v, broadcast $\langle \mathsf{blame}, v\rangle_r$.
   - *Quit view on $t+1$ blame messages.* Upon gathering $t+1$ distinct blame messages, broadcast $\langle \mathsf{quit\text{-}view}, v, CC\rangle$ along with $t+1$ blame messages where $\mathcal{CC}$ is the highest ranked chain certificate known to r. Abort all view v timers, and quit view v. Set $\mathsf{view\text{-}timer}_{v+1}$ to 2Δ and start counting down.
   - *Quit view on detecting equivocation.* If leader equivocation is detected, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle_r$ along with the equivocating proposals, abort all view v timers, and quit view v. Set $\mathsf{view\text{-}timer}_{v+1}$ to 2Δ and start counting down.
   - *Quit view on yield.* Upon receiving yield, broadcast $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC}\rangle_r$ message along with yield message, abort all view v timers, and quit view v. Set $\mathsf{view\text{-}timer}_{v+1}$ to 2Δ and start counting down.

Figure 3.5: Steady state protocol for optimistically responsive view-change.

Yield. When leader L wants to relinquish his leadership in view v, L broadcasts $\langle \mathsf{yield}, v \rangle_L$. The yield message forces an explicit view-change and useful for *democracy-favoring* leader policy and change leader after every block. Ideally, an honest leader issues yield after committing at least one block itself in view v.

Blame and quit view. The conditions for blaming the leader remains identical to earlier protocols. We make modifications in how a replica quits a view. Replicas quit view v when they receive $t+1$ blame messages, detect equivocation or receive a yield message from the current leader. On quitting view v, replica r broadcasts $\langle \mathsf{quit\text{-}view}, v, \mathcal{CC} \rangle_r$ where $\mathcal{CC}$ is the highest ranked chain certificate known to r. Replica r also broadcasts messages that triggered quitting view v, for example, a blame certificate or yield message. After quitting view v, replica r sets $\mathsf{view\text{-}timer}_{v+1}$ to 2Δ and starts counting down.

The requirements for a pre-commit in this protocol is identical to the requirements for a commit in the protocol in Section 3.4. Hence, a similar intuition for those steps apply here as well.

3.6.2 View-change Protocol

Let L and L' be the leader of view v and $v+1$, respectively.

i) **Status.** Replica r can enter view $v+1$ using one of the following rules:

   a) *Responsive.* Upon gathering $\lfloor 3n/4 \rfloor + 1$ distinct $\mathsf{quit\text{-}view}$ messages, broadcast them. Update its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to $\mathcal{CC}$ and send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ to L'. Enter view $v+1$ immediately. Stop $\mathsf{view\text{-}timer}_{v+1}$.

   b) *Synchronous.* When $\mathsf{view\text{-}timer}_{v+1}$ expires, update its chain certificate $\mathcal{CC}$ to the highest possible rank. Set lock_{v+1} to $\mathcal{CC}$ and send $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$ to L'. Enter view $v+1$.

ii) **New View.** Upon receiving a set $\mathcal{S}$ of $t+1$ distinct status messages after entering view $v+1$, broadcast $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}_{v+1} \rangle_{L'}$ along with $\mathcal{S}$ where lock_{v+1} is highest ranked chain certificate in $\mathcal{S}$.

iii) **First Vote.** Upon receiving the first $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$ along with $\mathcal{S}$, if lock' has a highest rank in $\mathcal{S}$, update lock_{v+1} to lock', broadcast $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$, and $\langle \mathsf{vote}, \mathsf{tip}(\mathsf{lock}'), v+1 \rangle_r$.

Figure 3.6: The optimistically responsive view-change protocol

Unlike a synchronous view-change as shown in Figure 3.2 that waits 2Δ before entering a new view, a responsive view-change allows replicas to quit current view and immediately transition to the next view without any delay. In the new view, a leader can also propose blocks without waiting for an additional 2Δ time. We make the following modifications to the view-change protocol to accommodate the responsive view-change.

Status. The status step includes two rules for entering into the new view. A replica r enters into view $v+1$ based on which rule is triggered first. A responsive rule is triggered when replica r receives responsive quit-view certificate $Q_B^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ quit-view messages in view v and enters view $v+1$ immediately. Replica r broadcasts $Q_B^{3/4}$ to all replicas, updates its lock, lock_{v+1} to a highest ranked chain certificate and sends lock_{v+1} to the new leader L' via a status message. The responsive status rule ensures that a replica receives a responsively committed blocks when making immediate transition to a higher view. This is critical to maintain the safety of protocol (explained later). Due to the synchrony assumption, all other honest replicas receive $Q_B^{3/4}$ within Δ time and transition immediately to view $v+1$.

The synchronous status rule is triggered when $\mathsf{view\text{-}timer}_{v+1}$ expires. Note that the $\mathsf{view\text{-}timer}_v$ was set to 2Δ. The 2Δ wait ensures that all honest replicas receive a highest ranked chain certificate $\mathcal{CC}$ in the quit-view message before entering view $v+1$. Replica r enters view $v+1$, and updates its lock, lock_{v+1} to a highest ranked chain certificate and sends lock_{v+1} to the new leader L' via $\langle \mathsf{status}, \mathsf{lock}_{v+1} \rangle_r$.

New-View. Upon entering view $v+1$, the leader waits for a set $\mathcal{S}$ of $t+1$ status messages. We call the set $\mathcal{S}$ of $t+1$ status messages as *status certificate*. Based on the status certificate $\mathcal{S}$, L' picks the highest ranked chain certificate lock_{v+1} and broadcasts new-view message $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}_{v+1} \rangle_{L'}$ along with $\mathcal{S}$. Sending $\mathcal{S}$ along with new-view message justifies that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends committed blocks in previous view.

First-Vote. Upon receiving a $\langle \mathsf{new\text{-}view\text{-}resp}, v+1, \mathsf{lock}' \rangle_{L'}$ message along with status certificate $\mathcal{S}$, if chain certificate lock' has the highest rank in $\mathcal{S}$, then it forwards the new-view message to all replicas and broadcasts a vote for it. Note that replica r may have lock_{v+1} with rank higher than lock'. A replica votes for lock' as long as lock' is vouched by $\mathcal{S}$. This is critical to ensure safety across views.

Next, we provide some intuition on how the view-change protocol provides liveness and safety across views.

How is the safety of a responsive commit maintained across views? Suppose an honest replica r responsively commits a block B_k at time t. A responsive commit for a block B_k requires a set $Q_C^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ commit messages. A responsive view-change requires a set $Q_B^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ quit-view messages. Due to a quorum intersection argument, $Q_C^{3/4}$ and $Q_B^{3/4}$ intersect in at least one honest replica h which sends chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$. Observe that

this also explains why highest ranked chain certificate is sent with a quit-view message. The highest chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$ from the honest replica h at the intersection allows another replica r' performing a responsive view change to learn about the commit of B_k.

A synchronous view-change waits 2Δ time before moving to a higher view. If an honest replica $h \in Q_C^{3/4}$ pre-commits responsively, the chain certificate $\mathcal{CC}$ sent by replica h in quit-view message reaches replica r' by the time the replica r' enters view $v+1$. Similarly, if replica $h \in Q_C^{3/4}$ pre-commits synchronously, honest replicas making a synchronous view-change receive $\mathcal{C}_v(B_k)$ by the time replica h pre-commits. Thus, all honest replicas lock on chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$.

How is the safety of a synchronous commit maintained across views? Consider replica r votes for B_k at time $\tau - 3\Delta$ and synchronously commits at time τ. Note that no honest replica has entered a higher view by time $\tau - \Delta$. This implies all honest replicas receive $\mathcal{C}_v(B_k)$ by time $\tau - \Delta$. Any view-change after $\tau - \Delta$ will receive $\mathcal{C}_v(B_k)$ or higher and honest replicas will lock on chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends $\mathcal{C}_v(B_k)$ before entering a higher view.

Why is it safe to vote for a valid new-view message with a lower ranked lock? The commit rules in the protocol ensure that there does not exist an equivocating chain certificate CC' such that $\mathsf{tip}(CC')$ does not extend committed blocks. This implies honest replicas lock on chain certificates that extend the committed blocks. After entering a higher view, honest replicas send their locked chain certificates via a status message. The new leader collects a status certificate $\mathcal{S}$ of $t+1$ status messages, extends on the highest ranked certified block in $\mathcal{S}$. Note that an honest replica sends a status message only after entering a higher view and has locked on a chain certificate that extends committed blocks in the previous view. As $\mathcal{S}$ contains status from at least one honest replica, the highest ranked chain certificate lock' in $\mathcal{S}$ will extend committed blocks in the previous view. Thus, it is safe for replicas to unlock a lock with a rank higher than lock'.

In the new view, due to the status certificate, all honest replicas will vote for the new-view message sent by an honest leader. Subsequently, in the steady state, honest replicas will keep committing new blocks.

3.6.3 Safety and Liveness

Claim 19. *If a block B_k is committed directly in view v using the responsive commit rule, then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k*

and a block in $\mathcal{CC}'$ equivocates B_k.

Proof. If a replica r responsively commits a block B_k in view v, then r must have received $\lfloor 3n/4 \rfloor + 1$ distinct commit messages out of which at least a set $\mathcal{R}$ of $\lfloor (n-t)/2 + 1 \rfloor$ are from honest replicas. An honest replica (say, $r' \in \mathcal{R}$) sends commit message only if it pre-commits and has not sent a blame message.

Replica r' can pre-commit in two ways. First, r' received $\lfloor 3n/4 \rfloor + 1$ votes for B_k in view v and pre-committed responsively. This case is identical to responsive commit rule for the protocol in Section 3.4. By Claim 6, an equivocating chain certificate $\mathcal{CC}'$ of rank higher than $(\mathcal{C}_v^{3/4}(B_k), \perp)$ cannot exist in view v. Second, replica r' voted for B_k at time $\tau - 2\Delta$ and received no equivocation or blame certificate by time τ and synchronously pre-commits at time τ. This case is identical to synchronous commit rule for the protocol in Section 3.4. By Claim 7, there does not exist a block certificate for an equivocating block in view v. Thus, chain certificate $\mathcal{CC}'$ with an equivocating block such that $\mathcal{CC}' > \mathcal{CC}$ cannot exist in view v. □

Claim 20. *If a block B_k is directly committed in view v, using the synchronous commit rule then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k.*

Proof. Replica r synchronously commits a block B_k when its $\mathsf{commit\text{-}timer}_{v,k}$ expires. Replica r could pre-commit in two ways. First, replica r pre-commits responsively. The responsive pre-commit rule is identical to the responsive commit rule for the protocol in Section 3.4. By Claim 6, an equivocating chain certificate $\mathcal{CC}'$ of rank higher than $(\mathcal{C}_v^{3/4}(B_k), \perp)$ cannot exist in view v.

Second, replica r synchronously pre-commits at time τ, i.e., it voted for B_k at time $\tau - 2\Delta$ and received no equivocation or blame certificate by time τ. This case is identical to synchronous commit rule for the protocol in Section 3.4. By Claim 7, there does not exist a block certificate for an equivocating block in view v. Thus, chain certificate $\mathcal{CC}'$ with an equivocating block cannot exist in view v. □

Lemma 21. *If a block B_k is directly committed in view v, then there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k.*

Proof. Straightforward from Claim 19 and Claim 20. □

Claim 22. *Let B_k be a block proposed in view v using Step 1 in Figure 3.5. If an honest replica votes for B_k at time τ in view v and detects no equivocation or blame certificate at time $\leq \tau + 2\Delta$, then (i) all honest replicas are in view v at time $\tau + \Delta$, (ii) all honest replicas vote for B_k at time $\leq \tau + \Delta$.*

Proof. Suppose an honest replica r votes for B_k at time τ in view v and detects no equivocation or blame certificate by time $\tau + 2\Delta$. This implies two facts. First, replica r entered view v at time $\leq \tau$. If r entered view v responsively, i.e., by receiving a responsive quit-view certificate, $Q_B^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ quit-view messages, it must have sent $Q_B^{3/4}$ at time $\leq \tau$. All honest replicas receive $Q_B^{3/4}$ and enter view v at time $\leq \tau + \Delta$. If r quit the previous view due to $t+1$ blame messages, it must have sent the blame certificate at time $\leq \tau - 2\Delta$ which arrives all honest replicas at time $\leq \tau - \Delta$. Due to the 2Δ wait after receiving $t+1$-sized blame certificate, all honest replicas enter view v at time $\leq \tau + \Delta$. We note that no honest replica has quit view v at time $\leq \tau + \Delta$; otherwise, replica r receives a blame certificate at time $\leq \tau + 2\Delta$. This proves part (i) of the claim.

Replica r received a proposal for B_k which contains $\mathcal{C}_v(B_{k-1})$ at time τ. Replica r's vote and forwarded proposal for B_k arrives at all honest replicas at time $\leq \tau + \Delta$. No honest replica has voted for an equivocating block or received a blame certificate at time $\leq \tau + \Delta$; otherwise replica r would have received an equivocation or blame certificate at time $\leq \tau + 2\Delta$. Thus, all honest replicas will vote for B_k at time $\leq \tau + \Delta$. This proves part (ii) of the claim. □

Claim 23. *Let B_k be a block proposed in view v using Step 1 in Figure 3.5. If an honest replica votes for B_k at time τ in view v and detects no equivocation or blame certificate at time $\leq \tau + 3\Delta$, then (i) all honest replicas are still in view v at time $\tau + 2\Delta$ (ii) all honest replicas receive $\mathcal{C}_v(B_k)$ at time $\leq \tau + 2\Delta$.*

Proof. Suppose an honest replica r votes for a block B_k at time τ in view v and detects no equivocation or blame certificate by time $\tau + 3\Delta$. Trivially, replica r has not received an equivocation or blame certificate by time $\tau + 2\Delta$. By Claim 22 (i), all honest replicas are in view v at time $\tau + \Delta$. No honest replica has quit view v by time $\tau + 2\Delta$; otherwise replica r must receive blame certificate by time $\tau + 3\Delta$ contradicting our hypothesis. Thus, all honest replicas are still in view v at time $\tau + 2\Delta$. This proves part (i) of the claim.

If replica r receives no equivocation or blame certificate at time $\leq \tau + 3\Delta$, it is easy to see that replica r receives no equivocation or blame certificate by time $\tau + 2\Delta$. By Claim 22, all honest replicas vote at time $\leq \tau + \Delta$. By synchrony assumption, all honest replicas receive at least $t+1$

votes for B_k i.e., $\mathcal{C}_v(B_k)$ at time $\leq \tau + 2\Delta$. This proves part (ii) of the claim. □

Claim 24. *If an honest replica directly commits a block B_k in view v using the responsive commit rule, then all honest replicas receive a chain certificate $\mathcal{CC}$ before entering view $v+1$ such that* $\mathsf{tip}(\mathcal{CC})$ *extends B_k.*

Proof. We first discuss the case where some replica performs a view-change due to a responsive quit-view certificate, and then discuss a view-change due to a synchronous blame certificate. Suppose an honest replica r receives a set $Q_C^{3/4}$ of $\lfloor 3n/4 \rfloor + 1$ commit messages for block B_k in view v and responsively commits B_k at time τ. Thus, all honest replicas in $Q_C^{3/4}$ must have received $\mathcal{C}_v(B_k)$ before sending the commit message. By Claim 19, there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k. Consider the quorum $Q_B^{3/4}$ that made some honest replica r' enter view $v+1$. r' receives a responsive quit-view certificate of $\lfloor 3n/4 \rfloor + 1$ quit-view messages each of which contains a chain certificate when the quit-view message was sent. By quorum intersection argument, $Q_C^{3/4}$ and $Q_B^{3/4}$ must intersect in at least one honest replica. Thus, the intersecting honest replica must include a higher ranked chain certificate $\mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k in $\mathsf{quit\text{-}view}$ message. This implies any replica that makes a responsive view-change must receive $\mathcal{CC}$ before entering view $v+1$.

Consider a view-change due to a synchronous blame certificate. Observe that any honest replica (say, replica u) that quits view v due to a synchronous blame certificate has not entered view $v+1$ at time $\tau + \Delta$; otherwise replica u must have sent a blame certificate at time $\leq \tau - \Delta$ (due to the 2Δ wait in the status step) and r must receive the blame certificate at time $\leq \tau$ and r wouldn't commit.

Let τ' be the time in which replica u enters view $v+1$ (with $\tau' > \tau + \Delta$). If some honest replica r' in $Q_C^{3/4}$ pre-committed responsively, r' must have received a blame certificate between time τ and $\tau' - \Delta$ and sent a $\mathsf{quit\text{-}view}$ message containing $\mathcal{CC}$ and replica u receives $\mathcal{CC}$ at time $\leq \tau'$. Similarly, if replica r' synchronously pre-commits B_k by time τ, it votes for B_k by time $\tau - 2\Delta$ and detects no equivocation or blame certificate by time τ. By Claim 22 (ii), all honest replicas vote for B_k by time $\tau - \Delta$. Hence, replica u receives $C_v(B_k)$ by time τ before entering view $v+1$. This implies any replica that makes a synchronous view-change has $\mathcal{CC}$ before entering view $v+1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. □

Claim 25. *If an honest replica directly commits a block B_k in view v using the synchronous commit rule, then all honest replicas receive a chain certificate $\mathcal{CC}$ before entering view $v+1$ such that* $\mathsf{tip}(\mathcal{CC})$ *extends B_k.*

Proof. Suppose an honest replica r synchronously commits a block B_k at time τ in view v. Its $\mathsf{commit\text{-}timer}_{v,k}$ for B_k expires at time τ without detecting an equivocation or blame certificate.

Replica r waits for 3Δ before its $\mathsf{commit\text{-}timer}_{v,k}$ expires. Replica r votes for B_k in view v at time $\tau - 3\Delta$ and detects no equivocation or blame certificate by time τ. By Claim 23, all honest replicas are in view v at time $\tau - \Delta$ and receive $C_v(B_k)$ by time $\tau - \Delta$. Thus, all honest replicas receive $\mathcal{C}_v(B_k)$ before entering view $v + 1$. This implies all honest replicas have a chain certificate $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. □

Lemma 26. *If an honest replica directly commits a block B_k in view v, then all honest replicas have lock_{v+1} before entering view $v + 1$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. By Claim 24 and Claim 25, all honest replicas receive a certificate chain $\mathcal{CC}$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. By Lemma 21, there does not exists an equivocating chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$. Since, honest replicas lock on highest ranked chain certificate, all honest replicas update lock_{v+1} to $\mathcal{CC}$ with $\mathsf{tip}(\mathsf{lock}_{v+1})$ extending B_k. □

Claim 27. *If an honest replica directly commits a block B_k in view v, the tip of a highest ranked chain certificate $\mathcal{CC}$ in a view v status certificate, i.e., $\mathsf{tip}(\mathcal{CC})$ must extend B_k.*

Proof. Suppose an honest replica r commits a block B_k in view v. By Lemma 26, all honest replicas lock on $\mathcal{CC}$ before entering view $v+1$ such that $\mathsf{tip}(\mathcal{CC})$ extends B_k. An honest replica sends status message containing their $\mathcal{CC}$ only after entering view $v+1$. A view v status certificate contains a set $\mathcal{S}$ of $t + 1$ status messages which includes the status message from at least one honest replica. By Lemma 21, there does not exist a chain certificate $\mathcal{CC}'$ in view v such that $\mathcal{CC}' > \mathcal{CC}$ where $\mathsf{tip}(\mathcal{CC})$ extends B_k and a block in $\mathcal{CC}'$ equivocates B_k. Thus, the tip of highest ranked chain certificate $\mathcal{CC}$ in $\mathcal{S}$, i.e., $\mathsf{tip}(\mathcal{CC})$ must extend B_k. □

Corollary 28. *If the tip of highest ranked chain certificate $\mathcal{CC}$ in a view v status certificate, i.e., $\mathsf{tip}(\mathcal{CC})$ does not extend a block B_k, then B_k has not been committed in view v.*

Lemma 29 (Unique Extensibility)**.** *If an honest replica directly commits a block B_k in view v, and $\mathcal{C}_{v'}(B_{k'})$ is a view $v' > v$ block certificate, then $B_{k'}$ extends B_k. Moreover, all honest replicas have $\mathsf{lock}_{v'}$ such that $\mathsf{tip}(\mathsf{lock}_{v+1})$ extends B_k.*

Proof. The proof is by induction on the view $v' > v$. For a view v', we prove that if $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ exists then it must extend B_k. A simple induction then shows that all later block certificates must also extend $\mathsf{tip}(\mathsf{lock}')$, this follows directly from the Vote rule in line 2.

For the base case, where $v' = v+1$, the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from Lemma 26 because the only way such a block can be certified is if some honest replica votes for it. However, all honest replicas are locked on a block that extends B_k and a chain certificate with a higher rank for an equivocating block does not exist. Although, honest replicas unlock on their locked chain certificates lock_{v+1} and lock on a highest ranked chain certificate lock' in a status certificate $\mathcal{S}$, by Claim 27, $\mathsf{tip}(\mathsf{lock}')$ must extend B_k. Thus, no honest replica will first vote (Figure 3.2 step iii)) for a block that does not extend B_k. The second part follows directly from Lemma 26.

Given that the statement is true for all views below v', the proof that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}'))$ extends B_k follows from the induction hypothesis because the only way such a block can be certified is if some honest votes for it. An honest party with a lock lock will vote only if $\mathsf{tip}(\mathsf{lock}_{v'})$ has a valid block certificate and $\mathsf{lock} \geq \mathsf{lock}_{v'}$. Due to Lemma 26 and the induction hypothesis on all block certificates of view $v < v'' < v'$ is must be that $\mathcal{C}_{v'}(\mathsf{tip}(\mathsf{lock}))$ extends B_k. □

Safety. The safety proof remains identical to that of Theorem 12 except Lemma 21 and Lemma 29 needs to be invoked.

Theorem 30 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. In a view, a leader has to propose at least p blocks that trigger honest replica's votes in $(2p + 4)\Delta$ time. As long as the leader proposes at least p valid blocks, honest replicas will keep voting for the blocks and keep committing the proposed blocks. If the Byzantine leader equivocates or proposes less than p blocks, a view-change will occur. Eventually, there will be an honest leader due to round-robin leader election.

Next, we show that once the leader is honest, a view-change will not occur and all honest replicas keep committing new blocks. If a block B_k has been committed in a previous view, by Lemma 26, all honest replicas lock on a chain certificate lock_{v+1} such that $\mathsf{tip}(\mathcal{CC})$ extends B_k before entering a new view. After entering a new view, honest replicas send their locked $\mathcal{CC}$ to the new leader in status message. The new leader extends on the tip of a highest ranked chain certificate (say, lock') in a status certificate $\mathcal{S}$. Even if some honest replicas are locked on chain certificates (say, $\mathcal{CC}$") that rank higher than lock', by Corollary 28 it is safe to unlock on $\mathcal{CC}$". Hence, honest replicas will vote for blocks that extend $\mathsf{tip}(\mathsf{lock}')$. After that, the honest leader can propose at least one block in 2Δ time and keep making progress. Moreover, the honest leader doesn't equivocate. This ensures all honest replicas keep committing new blocks. □

3.7 Evaluation

In this section, we evaluate the performance of the protocol with optimal optimistic responsiveness with 2Δ synchronous latency and $> 3n/4$ sized quorum (Section 3.4). Here after, we call the protocol *OptSync* for brevity. We first evaluate the throughput and latency of OptSync under varying batch sizes and payload. We then compare OptSync with Sync HotStuff [7] and HotStuff [104] at optimal batch size under different payloads and system size.

3.7.1 Implementation Details and Methodology

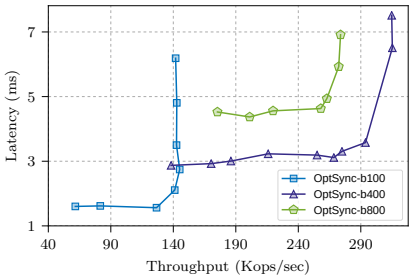
Our implementation is an adaption of the open-source implementation of Sync HotStuff. We modify the core consensus logic to replace the core Sync HotStuff code with OptSync.

In our implementation, each block consists of a batch of client commands. Each command contains a unique command identifier and an associated payload. The number of commands in a block determines its batch size. The throughput and latency results were measured from the perspective of external clients that run on separate machines from that of the replicas. The clients broadcast a configurable outstanding number of commands to every replica. Clients issue more commands when the issued commands have been committed. In all of our experiments, we ensure that the performance of replicas are not limited by lack of client commands.

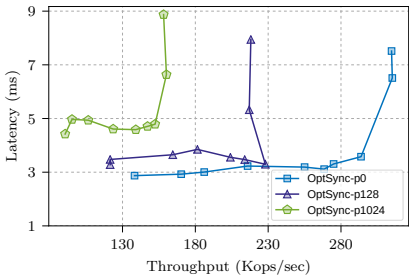
Experimental Setup. All our replicas and clients were installed on Amazon EC2 `c5.4xlarge` instances. Each instance has 16 vCPUs supported by Intel Xeon Platinum 8000 processors with maximum network bandwidth of upto 10Gbps. The network latency between two machines is measured to be less than 1ms. We used `secp256k1` for digital signatures in votes and quorum certificate consists of an array of `secp256k1` signatures.

Baselines. We make comparisons with two state-of-the-art protocols: (i) HotStuff, a partially synchronous protocol, and (ii) Sync HotStuff, a synchronous protocol. OptSync shares the same codebase with HotStuff and Sync HotStuff, and thus enables a fair comparison between the protocols. Although, HotStuff has a revolving leader policy, for fair comparison we chose to compare with HotStuff under stable leader policy as both OptSync and Sync HotStuff have a stable leader in the steady state. In all of the experiments, the curves represented by OptSync show the protocol's performance when the optimistic conditions are met. When the optimistic conditions are not met, our protocol behaves identically to Sync HotStuff (without responsiveness) and the curves marked

as Sync HotStuff describe the protocol's performance.



(a) Varying batch sizes.



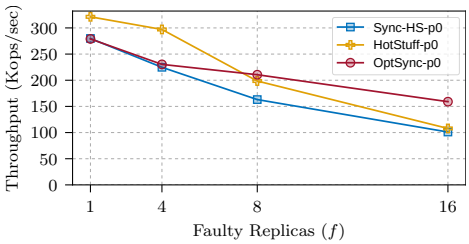
(b) Varying payload.

Figure 3.7: Throughput vs. latency at varying batch sizes and payload at $\Delta = 50$ms and $t = 1$.

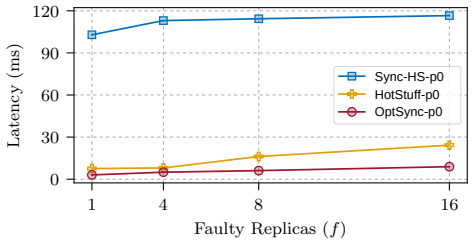
3.7.2 Basic Performance

We first evaluate the basic performance of OptSync when the tolerating $t = 1$ fault with a synchronous delay $\Delta = 50$ms. We measure the observed throughput (i.e., number of committed commands per second) and the end-to-end latency for clients. In our first experiment (Figure 3.7a), each command has a zero-byte payload and we vary batch size at different values, 100, 400, and 800 as represented by the three lines in the graph. Each point in the graph represents the measured throughput and latency for a run with a given load sent by clients. Basically, clients maintain an outstanding number of commands at any moment and issue more commands immediately when previous commands have been committed. We vary the size of outstanding commands to simulate different loads. As seen in the graph, the throughput increases with increasing load without increasing latency upto a certain point before reaching saturation. After saturation, the latency increases while the throughput either remains consistent or slightly degrades. We observe that the throughput is maximum at around 280 Kops/sec when the batch size is 400 with a good latency of around 3ms. We set the batch size to be 400 for our following experiments.

In our second experiment (Figure 3.7b), we vary the command request/response payload at different values in bytes 0/0, 128/128 and 1024/1024 with a fixed batch size of 400. Not surprisingly, as the payload size increases, each command requires a higher bandwidth and the throughput, measured in number of commands, decreases. We also observe a marginal drop in latency with increasing payload.

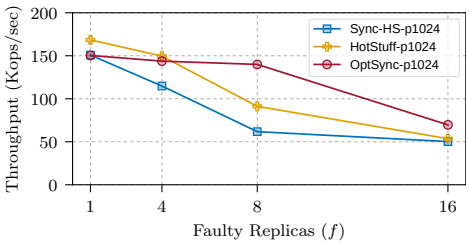


(a) f vs. throughput.

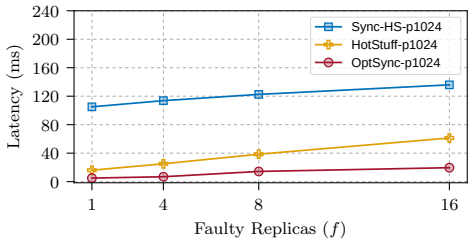


(b) f vs. latency.

Figure 3.8: Performance as function of faults at $\Delta = 50$ms, optimal batch size, and 0/0 payload.



(a) t vs throughput.



(b) t vs latency.

Figure 3.9: Performance as function of faults at $\Delta = 50$ms, optimal batch size, and 1024/1024 payload.

3.7.3 Scalability and Comparison with Prior Work

Next, we study how OptSync scales as the number of replicas increase. We also compare with HotStuff and Sync HotStuff. First, we study how the protocols perform with zero-payload commands to understand the raw overhead incurred by the underlying consensus mechanism at different values of t (Figure 3.8). Then, we study how the protocols perform at a higher payload of 1024/1024 (Figure 3.9). We use a batch size of 400 and a synchronous delay Δ of 50ms for both these experiments. Each data point in the graphs represent the throughput and latency at the saturation point without overloading the replicas. We note that we are using $2t + 1$ replicas for OptSync and Sync HotStuff, and $3t + 1$ replicas for HotStuff.

Comparison with HotStuff. The throughput of OptSync is slightly less than HotStuff for smaller system sizes (Figures 3.8a, 3.9a). But at higher faults, OptSync performs better than HotStuff for all payloads. This is because in both cases the system is bottlenecked by a leader communicating with all other replicas and since OptSync requires fewer replicas to tolerate t faults, its performance scales better than HotStuff.

In terms of latency (Figures 3.8b, 3.9b), OptSync performs much better than HotStuff. OptSync

commits in a single round of votes wherease HotStuff requires 3 rounds.

Comparison with Sync HotStuff. OptSync is identical to Sync HotStuff except for the responsive commit-path. The throughput of OptSync is consistently better than Sync HotStuff (Figures 3.8a, 3.9a). This is because Sync HotStuff, due to the synchronous wait time, needs to maintain a higher load of blocks at any time. In terms of latency, since the optimistic commit in OptSync does not incur $O(\Delta)$ delays, it's latency is far superior. We note that Sync HotStuff [7] work does describe an optimistically responsive protocol (that was not implemented). However, since they explicitly need to know whether optimistic conditions are met, they will always incur at least a 2Δ delay to switch paths, and hence will have a worse latency.

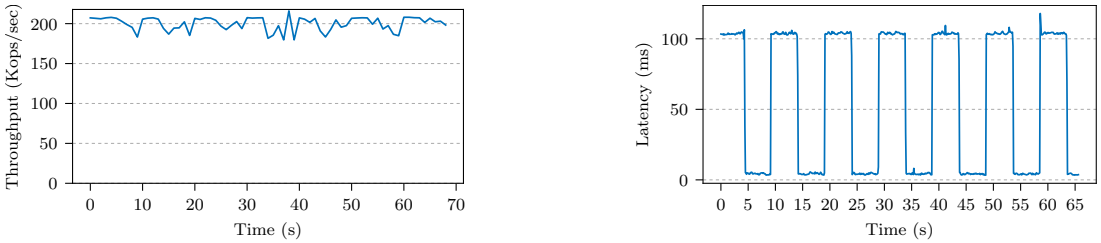


Figure 3.10: Throughput and latency vs time with two commit rules triggered intermittently at $\Delta = 50$ms and $t = 1$.

Performance under changing conditions. We further evaluate the performance of OptSync when the optimistic conditions are triggered intermittently and replicas commit using different commit rules. To simulate adversarial behavior where some replicas intermittently do not vote, we have t replicas who only intermittently vote and switch their behavior every 5s. The other $t + 1$ replicas always vote for all the proposed blocks. Figure 3.10 shows the throughput and latency for the commands across execution time. For latency, each point refers to the average latency for commands that were committed in the past 50ms. The commit latency switches between 3ms when optimistic conditions are met and 104ms when optimistic conditions are not met. The throughput remains consistent at around 200Kops/sec irrespective of the commit rules triggered. In comparison, protocols such as Sync HotStuff that follow the fast-path–slow-path paradigm will require an explicit view-change if sufficient replicas do not vote in the fast path, and hence require a view-change to commit.

3.8 Related Work

There has been a long line of work on Byzantine agreement starting at the Byzantine Generals Problem [77]. Dolev and Strong [43] presented a deterministic solution to the Byzantine Broadcast problem in the synchronous model tolerating $t < n - 1$ faults with a $t + 1$ round complexity. Several other works [4, 17, 50, 55, 71, 94, 65] have been proposed to improve the round complexity. We review the most recent and closely related works below. In particular, we make comparisons with synchronous BFT protocols with the notion of optimistic and synchronous commit paths. Compared to all of these protocols, our responsive commit incurs an optimal latency of 2δ and synchronous commit incurs a latency of 2Δ time while tolerating the same number of faults.

Thunderella. The idea of optimistic responsiveness in a back-and-forth slow-path–fast-path paradigm was first introduced in Thunderella [90]. They commit a decision in a single round under optimistic executions. Their path switching time and the synchronous latency is $O(\kappa\Delta)$ or $O(n\Delta)$, where κ is a security parameter.

Sync HotStuff. Like Thunderella, Sync HotStuff [7] is presented in a back-and-forth slow-path–fast-path paradigm. If started in the wrong path, their responsive commit will incur a latency of $2\Delta + O(\delta)$ time and synchronous commit incurs $4\Delta + O(\delta)$ time. Compared to them, our protocol in Section 3.6 can also perform an optimistically responsive view change, while their view change always incurs a 2Δ delay.

Comparison with works having simultaneity in commits. Our upper bound results are not the first results to use simultaneous paths. There are works such as Zyzzyva [74], SBFT [62] and FaB [80] which have considered the notion of simultaneous paths under partial synchrony. Similarly, a recent work called PiLi [35] achieves simultaneity under a synchronous assumption. Ours is the first work that achieves simultaneity under a synchrony assumption while obtaining optimal latency.

PiLi. PiLi [35] presents a BFT SMR protocol that progresses through a series of epochs. The protocol assumes lock-step execution in epochs. Each epoch lasts for $O(\delta)$ (resp. 5Δ) under optimistic (resp. synchronous) conditions or $O(\delta)$. The protocol commits 5 blocks after 13 consecutive epochs. PiLi has a responsive (resp. synchronous) latency of at least 16δ-26δ (resp. 40Δ-65Δ).

Hybrid-BFT. Hybrid-BFT [83] is an independent and concurrent work. They propose an optimistically responsive protocol with both responsive and synchronous commit paths existing simultane-

ously. However, after a responsive commit, their protocol waits for 7Δ time before starting the next block. From the perspective of a client, if a command is sent to replicas just after processing some command, the replicas will not process them for 7Δ time; though after that, it will immediately commit within $O(\delta)$ time. In comparison, our protocols will commit within $O(\delta)$ time without waiting for a synchronous delay. Their synchronous commits also incur a similar 7Δ delay after starting a block. They also introduce a responsive view-change; however, a synchronous wait of 7Δ before the view-change makes it not responsive in essence.

After this work, Kim et al. [72] investigated optimistic responsiveness in the weakly-synchronous model called *mobile sluggish* model [63]. Abraham et al. [10] studied optimistically responsive view-change with $t < n/2$ fault tolerance.

Chapter 4

Efficient State Machine Replication without Threshold Signatures

4.1 Introduction

There has been a long sequence of work on improving the communication complexity of consensus protocols [71, 50, 4, 104, 27, 8, 84]. In the synchronous SMR setting, the optimal communication complexity per consensus decision of an SMR protocol is $O(\kappa n^2)$ bits [4, 7, 100, 84]. However, all of these solutions use threshold signatures. Our result improves upon the communication complexity in the absence of threshold signatures. Specifically, we show the following:

Theorem 31 (Informal)**.** *Assuming public-key infrastructure and a universal structured reference string setup under q-SDH assumption, there exists a state machine replication protocol with amortized $O(\kappa n^2)$ communication complexity per consensus decision tolerating $t < n/2$ Byzantine faults.*

To be precise, the protocol incurs $O(\kappa n^2)$ communication complexity under q-strong Diffie-Hellman (SDH) assumption [25] (whose parameters can be generated using distributed protocols) or $O(\kappa n^2 \log n)$ without it. Getting rid of threshold signatures allows for efficient reconfiguration of the participating replicas and does not require generating threshold keys each time a new replica joins the system. It is in this sense that our system is reconfiguration-friendly. Thus, an efficient BFT protocol in this setting is of independent interest. We reduce communication by making use of efficient erasure coding schemes [95] and cryptographic accumulators [15] to efficiently broadcast large messages at the expense of increase in latency of SMR protocol. The resulting protocol has been used in

RandPiper [21] to obtain communication efficient random beacon protocol of the same complexity metrics.

4.2 Model and Preliminaries

We consider a system consisting of n replicas out of which at most $t = \lfloor (n-1)/2 \rfloor$ replicas can be Byzantine. The Byzantine replicas may behave arbitrarily. When we assume an adaptive adversary; the replicas can be corrupted into being Byzantine at any time during the execution of the protocol. A replica that is not faulty throughout the execution is considered to be honest and executes the protocol as specified.

We assume the network between replicas consists of point-to-point secure (authenticated and confidential) synchronous communication channels. Messages between replicas may take at most Δ time before they arrive, where Δ is a known maximum network delay. To provide safety under adversarial conditions, we assume that the adversary is capable of delaying the message for an arbitrary time upper bounded by Δ. In addition, we assume all honest replicas have clocks moving at the same speed. They also start executing the protocol within Δ time from each other. This can be easily achieved by using the clock synchronization protocol [4] once at the beginning of the protocol.

We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a replica p is digitally signed by p's private key and is denoted by $\langle x \rangle_p$. In addition, we use $H(x)$ to denote the invocation of the random oracle H on input x.

4.2.1 Primitives

In this section, we present several primitives used in our protocol.

Linear erasure and error correcting codes. We use standard (b, n) Reed-Solomon (RS) codes [95] with $b = t + 1$. This code encodes $t + 1$ data symbols into code words of n symbols and can decode the $t + 1$ elements of code words to recover the original data. The interfaces to (b, n) RS codes are presented in Section 2.5

Cryptographic accumulators. A cryptographic accumulator scheme constructs an accumulation

value for a set of values using Eval function and produces a witness for each value in the set using CreateWit function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. More details on these functions are provided in Section 2.5.

In this protocol, we use *collision free bilinear accumulators* from Nguyen [88] as cryptographic accumulators which generates constant sized witness, but requires q-SDH assumption. Alternatively, we can use Merkle trees [81] (and avoid q-SDH assumption) at the expense of $O(\log n)$ multiplicative communication.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator. Further, let $\kappa = \mathsf{max}(\kappa_h, \kappa_a)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the chapter, we will use the same parameter κ to denote the hash size, signature size and accumulator size for convenience.

4.3 BFT SMR Protocol

In this section, we present our BFT SMR protocol. Our SMR protocol achieves $O(\kappa n^2)$ bits communication complexity with a universal structured reference string (SRS) setup under the q-SDH assumption, or $O(\kappa n^2 \log n)$ bits communication complexity without the q-SDH setup assumption. In particular, we do not use threshold signatures, and thus avoid any distributed key generation during the setup or proactive secret sharing during reconfiguration. We note that prior synchronous BFT SMR protocols [7, 35, 100] with honest majority incur $O(\kappa n^3)$ communication per consensus decision without threshold signatures.

Epochs. Our protocol progresses through a series of numbered *epochs* with each epoch coordinated by a distinct leader. Epochs are numbered by integers starting with 1. The leaders for each epoch are rotated irrespective of the progress made in each epoch. For simplicity, we use round-robin leader election in this section and the leader of epoch e, represented as L_e, is determined by $e \bmod n$. Each epoch lasts for 11Δ time.

Blocks and block format. An epoch leader's proposal is represented as a *block*. Each block references its predecessor with the exception of the genesis block which has no predecessor. We call a block's position in the chain as its height. A block B_h at height h has the format, $B_h := (b_h, H(B_{h-1}))$ where b_h denotes the proposed payload at height h, B_{h-1} is the block at height $h-1$

and $H(B_{h-1})$ is the hash digest of B_{h-1}. The predecessor for the genesis block is $\perp$. A block B_h is said to be *valid* if (1) its predecessor block is valid, or if $h = 1$, predecessor is $\perp$, and (2) the payload in the block meets the application-level validity conditions.

A block B_h *extends* a block B_l ($h \geq l$) if B_l is an ancestor of B_h. Note that a block's height h and its epoch e need not necessarily be the same.

Certified blocks, and locked blocks. A block certificate on a block B_h consists of $t+1$ distinct signatures in an epoch e and is represented by $\mathcal{C}_e(B_h)$. Block certificates are ranked by epochs, i.e., blocks certified in a higher epoch has a higher rank. During the protocol execution, each replica keeps track of all certified blocks and keeps updating the highest ranked certified block to its knowledge. Replicas will lock on highest ranked certified blocks and do not vote for blocks that do not extend highest ranked certified blocks to ensure safety of a commit.

Equivocation. Two or more messages of the same *type* but with different payload sent by an epoch leader are considered an equivocation. In this protocol, the leader of an epoch e sends propose and vote-cert messages (explained later) to all other replicas. In order to facilitate efficient equivocation checks, the leader sends the payload along with the signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by L_e.

4.3.1 Protocol Details

We first describe a simple function that is used by an honest replica to forward a long message received from the epoch leader.

Deliver function. The Deliver() function (refer Figure 4.2) implements efficient broadcast of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are message type mtype, long message b, accumulation value z_e corresponding to object b and epoch e in which the deliver function is invoked. The input message type mtype corresponds to message *type* containing large message b sent by leader L_e of epoch e. In order to facilitate efficient leader equivocation checks, the input message type mtype, hash of object b, accumulation value z_e and epoch e are signed by leader L_e.

When the function is invoked using the above input parameters, the message b is partitioned into $t+1$ data symbols. The $t+1$ data symbols are then encoded into n codewords $(s_1, \ldots, s_n)$ using ENC function (defined in Section 4.2). Then, the cryptographic witness w_i is computed for each

Let e be the current epoch and L_e be the leader of epoch e. For each epoch e, replica r performs the following operations:

1. **Epoch advancement.** When $\mathsf{epoch\text{-}timer}_{e-1}$ reaches 0, enter epoch e. Upon entering epoch e, send highest ranked certificate $\mathcal{C}_{e'}(B_l)$ to L_e. Set $\mathsf{epoch\text{-}timer}_e$ to 11Δ and start counting down.

2. **Propose.** L_e waits for 2Δ time after entering epoch e and broadcasts $\langle \mathsf{propose}, B_h, \mathcal{C}_{e'}(B_l), z_{pe}, e \rangle_{L_e}$ where B_h extends B_l. $\mathcal{C}_{e'}(B_l)$ is the highest ranked certificate known to L_e.

3. **Vote.** If $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$ and replica r receives the first proposal $p_e = \langle \mathsf{propose}, B_h, \mathcal{C}_{e'}(B_l), z_{pe}, e \rangle_{L_e}$ where B_h extends a highest ranked certificate, invoke $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$. Set $\mathsf{vote\text{-}timer}_e$ to 2Δ and start counting down. When $\mathsf{vote\text{-}timer}_e$ reaches 0, send $\langle \mathsf{vote}, H(B_h), e \rangle_i$ to L_e.

4. **Vote cert.** Upon receiving $t+1$ votes for B_h, L_e broadcasts $\langle \mathsf{vote\text{-}cert}, \mathcal{C}_e(B_h), z_{ve}, e \rangle_{L_e}$.

5. **Commit.** If $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ and replica r receives the first $v_e = \langle \mathsf{vote\text{-}cert}, \mathcal{C}_e(B_h), z_{ve}, e \rangle_{L_e}$, invoke $\mathsf{Deliver}(\mathsf{vote\text{-}cert}, v_e, z_{ve}, e)$. Set $\mathsf{commit\text{-}timer}_e$ to 2Δ and start counting down. When $\mathsf{commit\text{-}timer}_e$ reaches 0, if no equivocation for epoch-e has been detected, commit B_h and all its ancestors.

6. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_e and stop performing epoch e operations.

Figure 4.1: **BFT SMR Protocol** with $O(\kappa n^2)$ bits communication per epoch and optimal resilience

$\mathsf{Deliver}(\mathsf{mtype}, b, z_e, e)$:

1. Partition input b into $t+1$ data symbols. Encode the $t+1$ data symbols into n codewords $(s_1, \ldots, s_n)$ using ENC function. Compute witness w_j $\forall s_j \in (s_1, \ldots, s_n)$ using $\mathsf{CreateWit}$ function. Send $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e \rangle_i$ to j^{th} replica $\forall j \in [n]$.

2. If j^{th} replica receives the first valid codeword $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e \rangle$ for the accumulator z_e, forward the codeword to all the replicas.

3. Upon receiving $t+1$ valid codewords for the accumulator z_e, decode b using DEC function.

Figure 4.2: **Deliver function**

codewords $(s_1, \ldots, s_n)$ using CreateWit (defined in Section 4.2). Then, the codeword and witness pair (s_j, w_j) is sent to the j^{th} replica along with the accumulation value z_e, message type mtype, and L_e's signature on the message.

When the j^{th} replica receives the first valid codeword s_j for an accumulation value z_e such that the witness w_j verifies the codeword s_j (using Verify function defined in Section 4.2), it forwards the codeword and witness pair (s_j, w_j) to all replicas. Note that j^{th} replica forwards only the first codeword and witness pair (s_j, w_j). Thus, it is required that all honest replicas forward the codeword and witness pair (s_j, w_j) for long message b; otherwise all honest replicas may not receive $t + 1$ codewords for b. When a replica r receives $t + 1$ valid codewords corresponding to the first accumulation value z_e it receives, it reconstructs the object b. Note that replica r reconstructs object b for the first valid share even though it detects equivocation in an epoch.

The Deliver function contains two communication steps and hence requires 2Δ time to ensure all honest replicas can receive at least $t + 1$ codewords sufficient to reconstruct the original input b. Invoking Deliver on a long message of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$ bits where κ is the size of accumulator and w is the size of the accumulator *witness*. The witness size is $O(\kappa)$ and $O(\kappa \log n)$ when bilinear accumulators and Merkle trees are respectively used as witnesses. Thus, the total communication complexity to broadcast a single message of size ℓ is $O(n\ell + \kappa n^2)$ bits, or $O(n\ell + \kappa n^2 \log n)$ bits without the q-SDH assumption.

BFT SMR Protocol. Our BFT SMR protocol is described in Figure 4.1. Consider an epoch e and its epoch leader L_e. To ensure an honest leader can always make progress, leader L_e first collects highest ranked certificate $\mathcal{C}_{e'}(B_h)$ from all honest replicas. In each epoch, at a high level, there are two "rounds" of communication from the epoch leader. The first round involves leader making a proposal and the second round involves sending certificates to aid in committing the proposal.

Efficient propagation of proposal. In the first round, the leader proposes a block B_h to every replica (step 2) by extending the highest ranked certificate $\mathcal{C}_{e'}(B_h)$. The proposal for B_h, conceptually, has the form $\langle \mathsf{propose}, B_h, \mathcal{C}_{e'}(B_l), z_{pe}, e \rangle_{L_e}$ where z_{pe} is the accumulation value for the pair $(B_h, \mathcal{C}_{e'}(B_l))$. In order to facilitate efficient equivocation checks, the leader signs the tuple $\langle \mathsf{propose}, H(B_h, \mathcal{C}_{e'}(B_l)), z_{pe}, e \rangle$ and sends B_h and $\mathcal{C}_{e'}(B_l)$ separately. The size of this signed message is $O(\kappa)$ bits. In case of equivocation, all-to-all broadcast of this signed message incur only $O(\kappa n^2)$ in communication.

If the received proposal is valid and it extends the highest ranked certificate known to a replica

r, replica r forwards the proposal. Forwarding the received proposal is required to ensure all honest replicas receive a common proposal; otherwise only a subset of the replicas may receive the proposal if the leader is Byzantine. Observe that the size of the proposal is linear as it contains certificate $\mathcal{C}_{e'}(B_l)$ (which is linear in the absence of threshold signatures). A naïve approach of forwarding the entire proposal incurs $O(\kappa n^3)$ when all replicas broadcast their proposal. In order to save communication, replicas forward the proposal by invoking Deliver function. For linear sized proposal, invoking Deliver incurs $O(\kappa n^2)$ bits (or $O(\kappa n^2 \log n)$ bits without q-SDH assumption) in communication.

Observe that the Deliver primitive requires 2Δ time. In particular, we need to ensure all honest replicas forward their codeword and witness pair for the proposal. Thus, our protocol waits for 2Δ time (i.e., vote-timer$_e$) before voting to check for equivocation. Hence, if no equivocation is detected at the end of 2Δ wait, all honest replicas forwarded their codeword and witness pair for the proposal and all honest replicas can reconstruct the proposal. At the end of 2Δ wait, if there no equivocation is detected, replicas vote for the proposed block B_h (step 3).

Ensuring the receipt of a certificate efficiently. Observe that a vote message is $O(\kappa)$ sized and hence, it can be broadcast using all-to-all communication with communication complexity of $O(\kappa n^2)$. However, if every replica that commits needs to ensure that all honest parties receive a certificate for the block being committed, this can result in $O(\kappa n^3)$ complexity again. This is because, all-to-all broadcast of linear sized certificate incurs $O(\kappa n^3)$. One might try to invoke Deliver to propagate the certificate. However, this does not save communication. This is because, in general, there can be exponentially many combinations of $t+1$ signatures forming a certificate depending on the set of signers, and each replica may invoke Deliver on a different combination.

This issue can be addressed if we ensure that there is a single certificate for a block. Hence, we use the leader to collect signatures and form a single certificate (step 3). The leader forwards this certificate via $\langle \text{vote-cert}, \mathcal{C}_e(B_h), z_{ve}, e \rangle_{L_e}$ to all replicas (step 4) where z_{ve} is the accumulation value of $\mathcal{C}_e(B_h)$. Similar to the proposal, the hash of the certificate is signed to allow for efficient equivocation checks. It is important to note that two different certificates for the same value is still considered an equivocation in this step.

To ensure that every honest replica receives this certificate, we again resort to the Deliver primitive which yields a communication complexity of $O(\kappa n^2)$ when all honest parties are invoking it using the same certificate. Again, to tolerate malicious behaviors such as sending multiple different certificates for the same block (due to which none of them may be delivered), we treat the vote-cert message similar to the proposal and perform equivocation checks. Thus, replicas commit only if

they observe no equivocation 2Δ time after they invoke Deliver (step 5).

Epoch timers. Observe that we set the epoch timer $\mathsf{epoch\text{-}timer}_e$ for each epoch e to be 11Δ. This is the maximum time required for an epoch when the leader is honest and all messages take Δ time. Similarly, in different steps, we make appropriate checks w.r.t. $\mathsf{epoch\text{-}timer}_e$ to ensure that the protocol is making sufficient progress within the epoch.

Latency. We note that all honest replicas commit in the same epoch when the epoch leader is honest. However, when the epoch leader is Byzantine, only some honest replicas may commit in that epoch. Due to the round-robin leader selection, there will be at least one honest leader every $t+1$ epochs and all honest replicas commit common blocks up to the honest epoch. Thus, our protocol has a worst-case commit latency of $t+1$ epochs.

4.3.2 Safety and Liveness

We say a block B_h is committed directly in epoch e if it is committed as a result of its own $\mathsf{commit\text{-}timer}_e$ expiring. We say a block B_h is committed indirectly if it is a result of directly committing a proposal B_ℓ ($\ell > h$) that extends B_h.

Claim 32. *If an honest replica delivers an object b at time τ in epoch e and no honest replica has detected an epoch e equivocation by time $\tau+\Delta$, then all honest replicas will receive object b by time $\tau+2\Delta$ in epoch e.*

Proof. Suppose an honest replica r delivers an object b at time τ in epoch e. Replica r must have sent valid codewords and witness $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e\rangle_i$ computed from object b to all replicas at time τ. The codewords arrive at all honest replicas by time $\tau+\Delta$.

Since no honest replica has detected an epoch e equivocation by time $\tau+\Delta$, it must be that either honest replicas will forward their codeword $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e\rangle$ when they receive the codewords sent by replica r or they already sent the corresponding codeword when they either delivered object b or received the codeword from some other replica r'. In any case, all honest replicas will forward their epoch e codeword corresponding to object b by time $\tau+\Delta$. Thus, all honest replicas will have received $t+1$ valid codewords for a common accumulation value z_e by time $\tau+2\Delta$ sufficient to decode object b by time $\tau+2\Delta$. □

Claim 33. *If an honest replica votes for a block B_h at time τ in epoch e, then all honest replicas*

receive B_h by time τ.

Proof. Suppose an honest replica r votes for a block B_h at time τ in epoch e. Replica r must have received proposal p_e for B_h by time $\tau - 2\Delta$ and detected no epoch e equivocation by time τ. This implies no honest replica detected an epoch e equivocation by time $\tau - \Delta$. Replica r must have invoked $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$ at time $\tau - 2\Delta$. By Claim 32, all honest replicas receive p_e by time τ. Thus, all honest replicas must have received B_h by time τ. □

Lemma 34. *If an honest replica directly commits a block B_h in epoch e, then (i) no equivocating block certificate exists in epoch e, and (ii) all honest replicas receive $\mathcal{C}_e(B_h)$ before quitting epoch e.*

Proof. Suppose an honest replica r commits a block B_h in epoch e at time τ. Replica r must have received a $\mathsf{vote\text{-}cert}$ for B_h at time $\tau - 2\Delta$ such that its $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ and did not detect an equivocation by time τ. This implies no honest replica detected an epoch e equivocation by time $\tau - \Delta$. In addition, some honest replica r' must have voted for B_h by time $\tau - 2\Delta$. By Claim 33, all honest replicas would receive B_h by time $\tau - 2\Delta$.

For part (i), observe that no honest replica received an equivocating proposal by time $\tau - 2\Delta$; otherwise, all honest replicas would have received a codeword for equivocating proposal by time $\tau - \Delta$ and replica r would not commit. And, no honest replica would vote for an equivocating block after time $\tau - 2\Delta$ (since they have received B_h by time $\tau - 2\Delta$). Thus, an equivocating block certificate does not exist in epoch e.

For part (ii), observe that replica r must have invoked $\mathsf{Deliver}(\mathsf{vote\text{-}cert}, v_e, z_{ve}, e)$ for $v_e = \mathcal{C}_e(B_h)$ at time $\tau - 2\Delta$ and did not detect epoch e equivocation by time τ. By Claim 32, all honest replicas receive v_e by time τ. Note that replica r must have its $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ at time $\tau - 2\Delta$. Since, all honest replicas are synchronized within Δ time, all other honest replicas must have $\mathsf{epoch\text{-}timer}_e \geq 2\Delta$ at time $\tau - 2\Delta$. Thus, all replicas are still in epoch e at time τ and receive $\mathcal{C}_e(B_h)$ before quitting epoch e. □

Lemma 35 (Unique Extensibility)**.** *If an honest replica directly commits a block B_h in epoch e, then any certified blocks that ranks higher than $\mathcal{C}_e(B_h)$ must extend B_h.*

Proof. The proof is by induction on epochs $e' > e$. For an epoch e', we prove that if a certificate $\mathcal{C}_{e'}(B_{h'})$ exists then it must extend B_h.

For the base case, where $e' = e + 1$, the proof that $\mathcal{C}_{e'}(B_{h'})$ extends B_h follows from Lemma 34. The only way $\mathcal{C}_{e'}(B_{h'})$ for $B_{h'}$ forms is if some honest replica votes for $B_{h'}$. However, by Lemma 34, there does not exist any equivocating block certificate in epoch e and all honest replicas receive and lock on $\mathcal{C}_e(B_h)$ before quitting epoch e. Thus, a block certificate cannot form for a block that does not extend B_h.

Given that the statement is true for all epochs below e', the proof that $\mathcal{C}_{e'}(B_{h'})$ extends B_h follows from the induction hypothesis because the only way such a block certificate forms is if some honest replica votes for it. An honest replica votes in epoch e' only if $B_{h'}$ extends a valid certificate $\mathcal{C}_{e''}(B_{h''})$. Due to Lemma 34 and the induction hypothesis on all block certificates of epoch $e < e'' < e'$, $\mathcal{C}_{e'}(B_{h'})$ must extend B_h. □

Theorem 36 (Safety)**.** *Honest replicas do not commit conflicting blocks for any epoch e.*

Proof. Suppose for the sake of contradiction two distinct blocks B_h and B'_h are committed in epoch e. Suppose B_h is committed as a result of $B_{h'}$ being directly committed in epoch e' and B'_h is committed as a result of $B'_{h''}$ being directly committed in epoch e''. Without loss of generality, assume $h' < h''$. Note that all directly committed blocks are certified. By Lemma 35, $B'_{h''}$ extends $B_{h'}$. Therefore, $B_h = B'_h$. □

Claim 37. *Let B_h be a block proposed in epoch e. If the leader of an epoch e is honest, then all honest replicas commit B_h and all its ancestors in epoch e.*

Proof. Suppose leader L_e of an epoch e is honest. Let τ be the earliest time when an honest replica r enters epoch e. Due to Δ delay between honest replicas, all honest replicas enter epoch e by time $\tau + \Delta$. Some honest replicas might have received a higher ranked certificate than leader L_e before entering epoch e; thus, they send their highest ranked certificate to leader L_e.

Leader L_e might have entered epoch e at time τ while some honest replicas enter epoch e only at time $\tau + \Delta$. The 2Δ wait in the Propose step ensures that the leader can receive highest ranked certificates from all honest replicas. However, leader L_e may enter epoch e Δ time after the earliest honest replicas. Due to 2Δ wait after entering epoch e, leader L_e collects the highest ranked certificate $\mathcal{C}_{e'}(B_l)$ by time $\tau + 3\Delta$ and sends a valid proposal $p_e = \langle \mathsf{propose}, B_h, e, \mathcal{C}_{e'}(B_l), z_{pe} \rangle_{L_e}$ for a block B_h that extends $\mathcal{C}_{e'}(B_l)$ which arrives all honest replicas by time $\tau + 4\Delta$.

Thus, all honest replicas satisfy the constraint $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$. In addition, B_h extends the highest ranked certificate. So, all honest replicas will invoke $\mathsf{Deliver}(\mathsf{propose}, p_e, z_{pe}, e)$ and set

vote-timer$_e$ to 2Δ which expires by time $\tau + 6\Delta$. All honest replicas send vote for B_h to L_e which arrives L_e by time $\tau + 7\Delta$. Leader L_e forwards $\mathcal{C}_e(B_h)$ which arrives all honest replicas by time $\tau + 8\Delta$. Note that all honest replicas satisfy the constraint epoch-timer$_e \geq 3\Delta$ and honest replicas set their commit-timer$_e$ to 2Δ which expires by time $\tau + 10\Delta$. Moreover, no equivocation exists in epoch e. Thus, all honest replicas will commit B_h and its ancestors in epoch e before their epoch-timer$_e$ expires. □

Theorem 38 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. For any epoch e, if the leader L_e is Byzantine, it may not propose any blocks or propose equivocating blocks. Whenever an honest leader is elected in epoch e, by Claim 37, all honest replicas commit in epoch e. Since we assume a round-robin leader rotation policy, there will be an honest leader every $t + 1$ epochs, and thus the protocol has a commit latency of $t + 1$ epochs. □

Lemma 39 (Communication complexity)**.** *Let ℓ be the size of block B_h, κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O(n\ell + (\kappa + w)n^2)$ bits per epoch.*

Proof. At the start of an epoch e, each replica sends a highest ranked certificate to leader L_e. Since, size of each certificate is $O(\kappa n)$, this step incurs $O(\kappa n^2)$ bits communication. A proposal consists of a block of size ℓ and block certificate of size $O(\kappa n)$. Proposing $O(n+\ell)$-sized object to n replicas incurs $O(\kappa n^2 + n\ell)$. Delivering $O(\kappa n + \ell)$-sized object has a cost $O(n\ell + (\kappa + w)n^2)$, since each replica broadcasts a codeword of size $O((n + \ell)/n)$, a witness of size w and an accumulator of size κ.

In Vote cert step, the leader broadcasts a certificate for block B_h which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_e(B_h)$ incurs $O((\kappa + w)n^2)$ bits. Hence, the total cost is $O(n\ell + (\kappa + w)n^2)$ bits. □

4.4 Related Work

There has been a long line of work in improving the latency and communication complexity of consensus protocols [71, 50, 4, 104, 27, 8, 84, 100, 9]. The state-of-the-art BFT SMR protocols [4, 7, 100, 9] incur quadratic communication per consensus decision while using threshold signatures. Without threshold signatures, they incur cubic communication per consensus decision. Our BFT

SMR protocol makes progress in the setting where threshold signatures are not desirable. Our protocol incurs $O(\kappa n^2)$ communication complexity under the q-SDH assumption or $O(\kappa n^2 \log n)$ without it at the expense of increased latency.

Chapter 5

Efficient Optimistically Responsive State Machine Replication without Threshold Signatures

5.1 Introduction

Improving the communication complexity of consensus protocols has been the research agenda of many works [71, 50, 4, 104, 27, 8, 84, 21] as it directly relates to the scalability of the system. The best communication complexity of SMR protocols is $O(\kappa n^2)$ bits per consensus decision [4, 7, 100, 84], where κ is the security parameter. However, all of these protocols use threshold signatures. In the previous chapter, we presented a BFT SMR protocol with $O(\kappa n^2)$ communication per consensus decision without the use of threshold signatures. Getting rid of threshold signatures allows for efficient reconfiguration of the participating replicas and does not require generating threshold keys each time a new replica joins the system. However, the BFT SMR protocol incurred a large latency to check for any misbehavior from the leader while propagating large messages. This prevents the protocol from progressing at network speed even during optimistic conditions. Another challenge in obtaining optimistic responsiveness is to synchronize all replicas when some replicas move to the next epoch. Typically, this is performed by replicas sharing synchronization proofs to all other replicas [40, 4]; in the absence of threshold signatures, these proofs tend to be $O(n)$ sized, making the communication cubic again.

Our protocol relies on aggregated secret opened in a verifiable manner in an epoch to synchronize all the replicas. The size of a aggregated secret is $O(\kappa)$ bits and thus, the communication complexity for synchronization stays quadratic. In addition, our protocol makes use of erasure coding and cryptographic accumulators in a way that does not require replicas to wait for $\Omega(\Delta)$ to check for equivocating behavior from the leader. Combining these two ideas, our protocol achieves optimistic responsiveness. Our protocol closely follows the optimistic responsive paradigm introduced in Chapter 3.

5.2 Model and Definitions

We consider a system $\mathcal{P} := \{P_1, \ldots, P_n\}$ consisting of n replicas in a reliable, authenticated all-to-all network, where up to $t < n/2$ replicas can be Byzantine faulty. We assume static corruption and the Byzantine replicas can behave arbitrarily. A replica that is not corrupted is considered to be honest and executes the protocol as specified.

Communications between replicas are synchronous. If an honest replica P_i sends a message x to another replica P_j at time τ, P_j receives the message by time $\tau + \delta$. The delay parameter δ is upper bounded by Δ. The upper bound Δ is known, but δ is unknown to the system. δ can be regarded as an actual delay in the real-world network. We assume all honest replicas have clocks moving at the same speed. They also start executing the protocol within Δ time from each other. This can be easily achieved by using the clock synchronization protocol [4] once at the beginning of the protocol.

We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a replica p is digitally signed by p's private key and is denoted by $\langle x \rangle_p$. In addition, we use $H(x)$ to denote the invocation of the random oracle H on input x; $H(x)$ is also called hash-digest of input x.

5.2.1 Primitives

In this section, we present primitives used in our protocol.

Linear erasure and error correcting codes. We use standard (n, b) Reed-Solomon (RS) codes [95]. This code encodes b data symbols into codewords of n symbols and can decode b

elements of the codewords to recover the original data. The interfaces to (b, n) RS codes are presented in Section 2.5

In this protocol, we instantiate the RS codes with n equal the number of all replicas, and b equal to $\lfloor n/4 \rfloor + 1$.

Cryptographic accumulators. A cryptographic accumulator scheme constructs an accumulation value for a set of values using Eval function and produces a witness for each value in the set using CreateWit function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. More details on these functions are provided in Section 2.5.

In this protocol, we use *collision free bilinear accumulators* from Nguyen [88] as cryptographic accumulators which generates constant sized witness, but requires q-SDH assumption. Alternatively, we can use Merkle trees [81] (and avoid q-SDH assumption) at the expense of $O(\log n)$ multiplicative communication.

Publicly Verifiable Secret Sharing. We assume the existence of an aggregatable Publicly Verifiable secret sharing scheme PVSS [64]. We use the interfaces to a secure PVSS scheme PVSS as described as follows:

- PVSS.Deal(s) : Given input a secret s, outputs a vector of commitments $\mathbf{v} :=$ (PVSS.v_1,..., PVSS.v_n) and an encrypted secret shares $\mathbf{c} :=$ (PVSS.c_1,..., PVSS.c_n).
- PVSS.Verify($\mathbf{v}, \mathbf{c}$) : Given input a vector of commitments and encrypted secret shares, output 1 if the secret sharing is valid; otherwise 0.
- PVSS.Aggregate($\mathbf{v}_1, \mathbf{c}_1, \mathbf{v}_2, \mathbf{c}_2$) : Given two valid PVSS tuples, output an aggregated PVSS tuple $(\mathbf{v}, \mathbf{c})$.
- PVSS.ShVrfy($\mathbf{v}_1, s_i$): Verify if the secret share s_i is correct. 0 indicates a failure and 1 indicates a success.
- PVSS.Recon(PVSS.$\vec{S}$): Given a set of $t+1$ secret shares, PVSS.$\vec{S}$, reconstruct the shared secret s.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator and $\kappa_v = \kappa_v(\lambda)$ denote the size of secret share and witness of a secret.

Further let $\kappa = \mathsf{max}(\kappa_h, \kappa_a, \kappa_v)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_v) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the chapter, we can use the same parameter κ to denote the hash size, signature size, accumulator size and secret share size for convenience.

5.3 Optimistically Responsive State Machine Replication

In this section, we present OptRand, an optimistically responsive random beacon protocol. Our protocol is a novel combination of state machine replication (SMR) protocol and random beacon protocol to achieve an optimistically responsive random beacons. Prior work [21, 36] used SMR as a black box to achieve consensus on the shared secrets to construct a random beacon protocol. In contrast, our protocol uses the generated random beacons to achieve responsiveness. In particular, we use aggregated secrets to synchronize between honest replicas and achieve responsiveness.

The underlying SMR protocol includes an optimistic path that can make progress at the network speed i.e., in $O(\delta)$ time during optimistic condition when the leader and $> 3n/4$ replicas behave honestly. Under standard conditions, i.e., when only $> n/2$ replicas behave honestly, the SMR protocol makes progress in $O(\Delta)$ time. We follow the optimistic responsive paradigm introduced in Chapter 3, i.e., our protocol does not require explicit back-and-forth switching between slow synchronous mode and fast optimistic mode. Similar to the optimistically responsive view-change protocol in OptSync, our protocol changes leaders in an optimistically responsive manner.

Epochs. Our protocol progresses through a series of numbered *epochs* with epoch r coordinated by a distinct leader L_r rotated in a round-robin manner. During optimistic conditions, the system progresses through epochs responsively, i.e., in $O(\delta)$ time; otherwise each epoch lasts for 11Δ time.

Blocks and block format. A block B_h at height h has the format, $B_h := (b_h, H(B_{h-1}))$ where b_h denotes the proposed payload at height h and $H(B_{h-1})$ is the hash digest of B_{h-1}. The predecessor for the genesis block is $\perp$. In our protocol, the payload b_h is set to the aggregated PVSS commitment and encryption. A block B_h is said to be *valid* if (1) its predecessor block is valid, or if $h = 1$, predecessor is $\perp$, and (2) the payload in the block is a valid PVSS vector, and (3) a valid DLPoK decomposition proof is provided by the leader. A block B_h *extends* a block B_l ($h \geq l$) if B_l is an ancestor of B_h.

Certified blocks, and locked blocks. A *block certificate* represents a set of signatures on a block in an epoch by a quorum of replicas. We use two types of signed vote messages: a responsive vote

resp-vote and a synchronous vote sync-vote. Accordingly, we consider two *types* of block certificates. A *responsive certificate* $\mathcal{C}_r^{3/4}(B_h)$ for a block B_h consists of $\lfloor 3n/4 \rfloor + 1$ distinct resp-vote on B_h in epoch r. Similarly, a *synchronous certificate* $\mathcal{C}_r^{1/2}(B_h)$ consists of $t+1$ distinct sync-vote on B_h in epoch r. Whenever the distinction is not important, we will represent the certificates by $\mathcal{C}_r(B_h)$.

Certified blocks are ranked by epochs, i.e., blocks certified in a higher epoch have a higher rank. We do not rank between responsive and synchronous certificate from the same epoch. During the protocol execution, each replica keeps track of all certified blocks and keeps updating the highest certified block to its knowledge. Replicas will lock on highest ranked certified blocks and do not vote for blocks that do not extend highest ranked block certificates to ensure safety of a commit.

Equivocation. Two or more messages of the same *type* but with different payload sent by an epoch leader is considered an equivocation. In this protocol, the leader of an epoch e sends propose, resp-cert, and sync-cert messages (explained later) to all other replicas. In order to facilitate efficient equivocation checks, the leader sends the payload along with signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by L_r.

Background: Dissecting BFT SMR in Chapter 4. In Chapter 4, we presented a communication efficient BFT SMR protocol that incurs $O(\kappa n^2)$ communication per decision to decide on $O(n)$-sized input without using threshold signatures. The efficient communication was achieved by making use of erasure coding schemes, cryptographic accumulators and broadcast of equivocating hashes (if any). In that protocol, we used $(n, t+1)$ RS codes to encode large messages. When a replica receives a valid proposal from the leader, it uses RS codes to encode the proposal into n codewords $(s_1, \ldots, s_n)$ and compute corresponding cryptographic witnesses $(w_1, \ldots, w_n)$, and send each codeword and witness pair (s_i, w_i) to replica j $\forall j \in [n]$. A replica votes for the proposed block only if it does not detect any equivocation for 2Δ time. The 2Δ wait before voting ensures (i) no honest replica received an equivocating proposal and conflicting (s_i', w_i') before receiving (s_i, w_i) (ii) all honest replicas receive at least $t+1$ codewords for the proposed block sufficient to reconstruct the proposal.

To ensure safety of a committed block, in general, SMR protocols ensure that all honest replicas receive and lock a certificate for the proposed block. A certificate consisting of $t+1$ signatures for the proposed block is linear in size in the absence of threshold signatures. Thus, an all-to-all broadcast of the certificate trivially incurs cubic communication. The BFT SMR protocol of RandPiper solves the issue using following technique. First, replicas send their vote only to the leader. The leader is expected to collect $t+1$ votes, form a single certificate and send it to all

replicas. Second, in order to ensure the certificate is propagated among all honest replicas, instead of broadcasting it to all replicas, they use RS codes to encode the certificate, send the codeword and witnesses and wait for 2Δ to check for an equivocation before making a commit.

Achieving optimistic responsiveness. The techniques employed by the BFT SMR protocol enables communication efficient consensus on $O(n)$-sized input. However, that technique requires waiting for $\Omega(\Delta)$ time to detect equivocation before making a decision.

In this chapter, we propose a new technique that allows us to responsively make decision and change leaders without relying on equivocation detection.

We modify the BFT SMR in the following manner: First, we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes to encode large messages (in the Deliver primitive in Figure 5.1). This allows decoding with $\lfloor n/4 \rfloor + 1$ codewords at the expense of doubled codeword size. Second, a replica sends a responsive vote to the leader as soon as it receives a valid block proposal. The replica also sends the RS coded codewords and witnesses to all other replicas. The leader collects $\lfloor 3n/4 \rfloor + 1$ votes, forms a responsive certificate and sends the responsive certificate to all replicas. The replicas broadcast an ack message in response to the responsive certificate and commit on receiving $> 3n/4$ distinct ack messages. In addition, they also send RS coded codewords and witnesses for the responsive certificate. The existence of $> 3n/4$ ack messages ensures that all honest parties can reconstruct the proposed blocks and the responsive certificate. In particular, at least $\lfloor n/4 \rfloor + 1$ honest replicas must have received the block proposal and the responsive certificate for the committed block and they have forwarded their codewords to all replicas. Thus, all honest replica must receive $\lfloor n/4 \rfloor + 1$ codewords sufficient to reconstruct the proposed blocks and the responsive certificate.

Responsively changing epochs. The above technique allows an honest replica to responsively commit a decision. In order to responsively change epochs, a synchronization primitive is required to signal all honest replicas to move to a higher epoch. Prior works [7, 100, 9] perform an all-to-all broadcast of certificates to synchronize between epochs which incurs cubic communication without threshold signatures. In this protocol, we broadcast aggregated secret opened in an epoch to synchronize all the replicas. The size of aggregated secret is $O(\kappa)$ bits and all-to-all broadcast of $O(\kappa)$-sized aggregated secret does not blow up communication.

In cases when optimistic conditions are not met, the underlying consensus mechanism works similar to the BFT SMR in RandPiper except we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes.

5.3.1 Protocol Details

Deliver(mtype, m, z_r, r):

1. Partition input m into $\lfloor n/4 \rfloor + 1$ data symbols. Encode the $\lfloor n/4 \rfloor + 1$ data symbols into n codewords $(s_1, \ldots, s_n)$ using ENC function. Compute witness w_j $\forall s_j \in (s_1, \ldots, s_n)$ using CreateWit function. Send $\langle \text{codeword}, \text{mtype}, s_j, w_j, z_r, r \rangle_r$ to replica j $\forall j \in [n]$.
2. If j^{th} replica receives the first valid codeword $\langle \text{codeword}, \text{mtype}, s_j, w_j, z_r, r \rangle$ for the accumulator z_r, forward the codeword to all the replicas.
3. Upon receiving $\lfloor n/4 \rfloor + 1$ valid codewords for a common accumulator z_r, decode m using DEC function.

Figure 5.1: **Deliver function**

Deliver function. We first present a Deliver function (refer Figure 5.1) that is used by an honest replica to propagate long messages received from the epoch leader.

The Deliver function enables efficient broadcast of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are a keyword mtype, long message m, accumulation value z_r corresponding to message m and epoch r in which Deliver function is invoked. The input keyword mtype corresponds to message *type* containing long message b sent by leader L_r. In order to facilitate efficient leader equivocation, the input keyword mtype, hash of long message m, accumulation value z_r, and epoch r are signed by leader L_r.

When the Deliver() function is invoked using above input parameters, the message m is first divided into $\lfloor n/4 \rfloor + 1$ data symbols. The $\lfloor n/4 \rfloor + 1$ data symbols are then encoded into n codewords $(s_1, \ldots, s_n)$ using ENC function (defined in Section 2.5). Then, the cryptographic witness w_i is computed for each codewords $(s_1, \ldots, s_n)$ using CreateWit (defined in Section 2.5). Then, the codeword and witness pair (s_j, w_j) is sent to the replica j $\forall j \in [n]$ along with the accumulation value z_r, keyword mtype, and L_r's signature on the message.

When a j^{th} replica receives the first valid codeword and witness pair (s_j, w_j) for an accumulation value z_r such that the witness w_j verifies the codeword s_j, it forwards the share (s_j, w_j) to all replicas. The validity of the codeword can be checked using Verify function (defined in Section 2.5). We note that j^{th} replica forwards a single codeword and witness pair (s_j, w_j) for each message type mtype in an epoch.

An honest replica r considers only the first codeword for each message type mtype from each replica $j \in [n]$. When a replica r receives $\lfloor n/4 \rfloor + 1$ valid codewords along with their witnesses for a common accumulation value z_r, it decodes long message m corresponding to accumulation value

Let r be the current epoch, L_r be the leader of epoch r and $\mathcal{P}_r$ be the set of removed replicas. For each epoch r, replica r performs following operations:

1. **Epoch advancement.** Replica P_i advances to epoch r using following rules:
   (a) When $\mathsf{epoch\text{-}timer}_{r-1}$ reaches 0, enter epoch r.
   (b) On receiving aggregated secret R_{r-1}, broadcast R_{r-1}. Wait until $\mathcal{C}_{r-1}(B_l)$ is received and enter epoch r.

   Upon entering epoch r, send PVSS tuple $(\mathbf{v}_i, \mathbf{c}_i)$ and highest ranked certificate $\mathcal{C}_{r'}(B_l)$ to L_r. Set $\mathsf{epoch\text{-}timer}_r$ to 11Δ and start counting down.

2. **Propose.** Wait for $t+1$ PVSS tuples and either $\mathcal{C}_{r-1}(B_l)$ or 2Δ time after entering epoch r. Upon receiving $t+1$ valid PVSS tuples, L_r aggregates them to obtain $(\mathbf{v}, \mathbf{c})$. Set $b_h := (\mathbf{v}, \mathbf{c})$ and send $\langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$ to replica $P_j \; \forall P_j \in \mathcal{P}$ where B_h extends B_l and $\mathcal{C}_{r'}(B_l)$ is the highest ranked certificate known to L_r.

3. **Vote.** If $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$ and replica P_i receives the first proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r\rangle_{L_r}$, check the validity of the $(\mathbf{v}, \mathbf{c})$. If valid and B_h extends a highest ranked certificate, invoke $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and send $\langle \mathsf{resp\text{-}vote}, H(B_h), r\rangle_{P_i}$ to L_r. Set $\mathsf{vote\text{-}timer}_r$ to 2Δ and start counting down. When $\mathsf{vote\text{-}timer}_r$ reaches 0, send $\langle \mathsf{sync\text{-}vote}, H(B_h), r\rangle_{P_i}$ to L_r.

4. **Resp cert.** On receiving $\lfloor 3n/4 \rfloor + 1$ $\mathsf{resp\text{-}vote}$ for B_h, L_r broadcasts $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$.

5. **Sync cert.** On receiving $t+1$ $\mathsf{sync\text{-}vote}$ for B_h, L_r broadcasts $\langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{sa}, r\rangle_{L_r}$.

6. **Ack.** Upon receiving the first responsive certificate $rc := \langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r\rangle_{L_r}$, invoke $\mathsf{Deliver}(\mathsf{resp\text{-}cert}, rc, z_{ra}, r)$ and broadcast $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle_{P_i}$.

7. **Commit.** Replica P_i commits using one of the following rules:
   (a) *Responsive.* If $\mathsf{epoch\text{-}timer}_r \geq 2\Delta$ and replica P_i receives $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle$ from $\lfloor 3n/4 \rfloor + 1$ distinct replicas and detects no equivocation, commit B_h and all its ancestors.
   (b) *Synchronous.* If $\mathsf{epoch\text{-}timer}_r \geq 3\Delta$ and replica P_i receives the first certificate (either responsive or synchronous), set $\mathsf{commit\text{-}timer}_r$ to 2Δ and start counting down. If the received certificate is synchronous i.e., $sc := \langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{sa}, r\rangle_{L_e}$, invoke $\mathsf{Deliver}(\mathsf{sync\text{-}cert}, sc, z_{sa}, r)$. When $\mathsf{commit\text{-}timer}_r$ reaches 0, if no epoch-r equivocation has been detected, commit B_h and all its ancestors.

8. **Update, reconstruct and output.** When replica P_i commits or when epoch r ends, perform following operations:
   (a) Commit block B_ℓ proposed in epoch $r-t$ if the highest ranked chain extends B_ℓ (if B_ℓ has not been committed).
   (b) If block B_ℓ proposed by L_{r-t} has been committed by epoch r, update $\mathcal{Q}(L_{r-t})$ with $(\mathbf{v}, \mathbf{c})$ shared in b_ℓ. Otherwise, remove L_{r-t} from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{r-t}\}$.
   (c) Obtain $(\mathbf{v}, \mathbf{c})$ corresponding to block committed in $\mathsf{Dequeue}(\mathcal{Q}(L_r))$. Broadcast decrypted share d_i. On receiving share d_j from another replica P_j, ensure that $\mathsf{PVSS.ShVrfy}(d_j) = 1$. On receiving $t+1$ valid shares in $\mathbf{S}$, reconstruct B and $R_r \leftarrow \mathsf{PVSS.Recon}(\mathbf{S})$. Broadcast (B, R_r). On receiving (B, R_r) from others, accept R_r if $R_r = e(B, g_2')$ and $e(B, g_2) = e(g_1, g_2^s)$.
   (d) Compute and output $\mathcal{O}_r \leftarrow H(R_r)$.

9. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_r and stop performing epoch r operations, except Step 8.

Figure 5.2: Optimistically responsive BFT SMR with $O(\kappa n^2)$ bits communication per epoch.

z_r using DEC function (defined in Section 2.5). Observe that since we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes, $\lfloor n/4 \rfloor + 1$ valid codewords are sufficient to decode the original message. Any invalid codewords are discarded. We note that replica r decodes message m as long as there are $\lfloor n/4 \rfloor + 1$ valid codewords (considering only the first codeword from each replica) corresponding to message m even though it detects equivocation in an epoch. Since, $n > 3(\lfloor n/4 \rfloor + 1)$, an honest replica r may receive $\lfloor n/4 \rfloor + 1$ valid codewords for upto three different long messages and replica r decodes all of them. This does not affect the correctness of our protocol in any way. Looking ahead, our protocol requires that a committed block is received by all honest replicas. We ensure that at least one of three decoded message belongs to the committed block.

This function is similar to the Deliver function in Chapter 4 except that we use $(n, \lfloor n/4 \rfloor + 1)$ RS codes instead of $(n, t + 1)$ RS codes used in [21]. As a result, the size of codeword is doubled and the communication is increased by a factor of 2. However, this does not linearly blow up the communication complexity and the communication complexity still remains $O(\kappa n^2)$ (more details in Lemma 39). Our beacon protocol is described in Figure 5.2. Replicas maintain a chain of blocks to add blocks proposed by leaders, a queue $\mathcal{Q}()$ to store a recently committed PVSS vector proposed by an epoch leader and set $\mathcal{P}_r$ to keep track of removed replicas. Before the start of the beacon protocol execution, a setup phase is executed where we establish PVSS parameters and public keys pk_i for every replica $P_i \in \mathcal{P}$. We also buffer one secret share for aggregated PVSS tuples for every replica P_i, i.e., fill $\mathcal{Q}(P_i)$ for $P_i \in \mathcal{P}$. The replicas in $\mathcal{P} \setminus \mathcal{P}_r$ are selected as leaders in a round-robin manner.

After the setup phase, the replicas execute following steps in each epoch r.

Epoch advancement. Each replica keeps track of epoch duration $\mathsf{epoch\text{-}timer}_r$ for epoch r. A replica P_i enters epoch r (i) when its $\mathsf{epoch\text{-}timer}_{r-1}$ expires, or (ii) when it receives a round $r-1$ aggregated secret R_{r-1} and a round $r-1$ block certificate $\mathcal{C}_{r-1}(B_l)$. Upon entering epoch r, replica P_i generates PVSS vector $(\mathbf{v}_i, \mathbf{c}_i)$ and sends the PVSS tuple and its highest ranked certificate to the leader L_r. In addition, it aborts all timers below epoch r and sets $\mathsf{epoch\text{-}timer}_r$ to 11Δ and starts counting down.

Propose. Upon entering epoch r, if Leader L_r has $\mathcal{C}_{r-1}(B_l)$, it proposes as soon as it receives $t + 1$ PVSS tuples; otherwise, it waits for 2Δ time to ensure it can receive the highest ranked certificate from all honest replicas. Upon receiving $t + 1$ PVSS tuples from $I \subset [n]$, it aggregates the PVSS tuples to obtain aggregated PVSS committments $\mathbf{v}$, aggregated encrypted secret shares $\mathbf{c}$. The leader L_r constructs a block B_h by extending on the highest ranked certificate $\mathcal{C}_{r'}(B_l)$

known to L_r with payload b_h set to $(\mathbf{v}, \mathbf{c})$ and sends proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r \rangle_{L_r}$ Here, z_{pa} is the accumulation value for the pair $(B_h, \mathcal{C}_{r'}(B_l))$. The proposal for B_h is common to all replicas while the NIZK proof $\boldsymbol{\pi}_j$ corresponds to replica P_j. While conceptually, the leader is sending $\langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r \rangle_{L_r}$, to facilitate equivocation checks it instead sends $\langle \mathsf{propose}, H(B_h, \mathcal{C}_{r'}(B_l)), z_{pa}, r \rangle_{L_r}$ with B_h and $\mathcal{C}_{r'}(B_l)$ sent separately. Here, z_{pa} is the accumulation value for the pair $(B_h, \mathcal{C}_{r'}(B_l))$. The size of the signed message is $O(\kappa)$ and hence can be broadcast during equivocation or while delivering p_r without incurring cubic communication overhead.

Vote. If replica P_i receives a proposal $p_r := \langle \mathsf{propose}, B_h, \mathcal{C}_{r'}(B_l), z_{pa}, r \rangle_{L_r}$ it first checks PVSS verification for $(\mathbf{v}, \mathbf{c})$. We call such a proposal valid. If replica P_i receives the valid proposal and the proposed block B_h extends the highest ranked certificate known to the replica such that its $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$, then it invokes $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and sends a responsive vote $\langle \mathsf{resp\text{-}vote}, H(B_h), r \rangle_{P_i}$ immediately to L_r. In addition, the replica sets its $\mathsf{vote\text{-}timer}_r$ to 2Δ and starts counting down. When $\mathsf{vote\text{-}timer}_r$ reaches 0 and detects no epoch r equivocation, the replica sends a synchronous vote $\langle \mathsf{sync\text{-}vote}, H(B_h), r \rangle_{P_i}$ to L_r. If block B_h does not extend the highest ranked certificate known to the replica or receives proposal p_r when its $\mathsf{epoch\text{-}timer}_r < 7\Delta$, the replica simply ignores the proposal and does not vote for B_h.

Resp cert. When the leader L_r receives $\lfloor 3n/4 \rfloor + 1$ distinct $\mathsf{resp\text{-}vote}$ messages for the proposed block B_h in epoch r, denoted by $\mathcal{C}_r^{3/4}(B_h)$, L_r broadcasts $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r \rangle_{L_r}$ to all replicas where z_{ra} is the accumulation value of $\mathcal{C}_r^{3/4}(B_h)$. Similar to the proposal, the hash of the certificate $\mathcal{C}_r^{3/4}(B_h)$ is signed to allow for efficient equivocation checks. Since our protocol requires the certificate to be *delivered* to all parties in case of a commit, we require two different certificates for the same block shared by a leader to be considered an equivocation.

Sync cert. When leader L_r receives $t + 1$ distinct $\mathsf{sync\text{-}vote}$ messages for the proposed block B_h in epoch r, denoted by $\mathcal{C}_r^{1/2}(B_h)$, L_r broadcasts $\langle \mathsf{sync\text{-}cert}, \mathcal{C}_r^{1/2}(B_h), z_{ra}, r \rangle_{L_r}$ to all replicas where z_{ra} is the accumulation value of $\mathcal{C}_r^{1/2}(B_h)$. Again, the hash of the certificate $\mathcal{C}_r^{1/2}(B_h)$ is signed to allow for efficient equivocation checks.

Ack. When a replica P_i receives a responsive certificate $rc := \langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r \rangle_{L_r}$ while in epoch r, it invokes $\mathsf{Deliver}(\mathsf{resp\text{-}cert}, rc, z_{ra}, r)$ to deliver rc and broadcasts $\langle \mathsf{ack}, H(B_h), z_{ra}, r \rangle_{P_i}$ to all replicas. If $\mathsf{epoch\text{-}timer}_r \leq 3\Delta$, replica P_i sets $\mathsf{commit\text{-}timer}_r$ to 2Δ and starts counting down.

Commit. The protocol includes two commit rules and the replica commits using the rule that is

triggered first. In responsive commit, a replica commits block B_h and all its ancestors immediately when it receives $> 3n/4$ ack messages for a responsive certificate $\mathcal{C}_r^{3/4}(B_h)$ with a common accumulation value z_{ra} such that its epoch-timer$_r$ is large enough (2Δ). Note that a responsive commit happens at the actual speed of the network (δ).

In synchronous commit, when replica P_i receives the valid epoch r certificate when its epoch-timer$_r$ is large enough (3Δ), it sets commit-timer$_r$ to 2Δ and starts counting down. If the received certificate is synchronous i.e., $sc := \langle \text{sync-cert}, \mathcal{C}_r^{1/2}(B_h), z_{ra}, r \rangle_{L_r}$, it invokes Deliver(sync-cert, sc, z_{sa}, r) and sets commit-timer$_r$ to 2Δ. When commit-timer$_r$ reaches 0, if no equivocation for epoch-r has been detected, replica P_i commits B_h and all its ancestors. The Deliver() message ensures that all honest replicas have received $\mathcal{C}_r(B_h)$ before quitting epoch r.

In addition to above commit rules, we include an additional commit rule. We consider a block B_ℓ proposed in epoch $r - t$ proposed by L_{r-t} committed if the highest ranked chain at the end of epoch r extends B_ℓ even though none of the blocks that extends B_ℓ proposed after epoch $r - t$ have been committed using either of the above commit rules. This commit rule helps in committing safe blocks possibly uncommitted due to responsively moving to higher epoch.

We note that if an honest replica commits a block B_h in epoch r using one of the commit rules, it is not necessary that all honest replicas commit B_h in epoch r using the same rule, or commit B_h at all. Depending on how Byzantine replicas behave, only some honest replicas may receive $> 3n/4$ ack messages and commit using responsive commit rule while some other honest replicas may commit using synchronous commit rule. It is also possible that only some honest replica commits B_h while no commit rules are triggered for rest of the honest replicas. For example, an honest replica commits a block B_h responsively but all other replicas detect equivocation in the epoch. In such a case, we ensure that all honest replicas receive and lock on a certificate for B_h, i.e., $\mathcal{C}_r(B_h)$, to ensure safety of a commit. Eventually after $t + 1$ epochs, all honest replicas will commit B_h using our third commit rule.

Equivocation. At any time in epoch r, if a replica P_i detects an equivocation, it broadcasts equivocating hashes signed by leader L_r. Replica P_i also stops performing epoch r operations except update, reconstruct and output steps described below. In addition, if epoch-timer$_r > 2\Delta$, replica P_i resets epoch-timer$_r$ to 2Δ to assist in terminating a faulty epoch faster.

Update. The update step ensures that the leaders failing to commit a block in $t + 1$ epochs are removed the active set of replicas, i.e., if the leader L_{r-t} of epoch $r - t$ fails to add a new block by the end of epoch r, L_{r-t} is removed from future proposals, i.e., $\mathcal{P}_r \leftarrow \mathcal{P}_r \cup \{L_{r-t}\}$. On the other

hand, if block B_ℓ proposed by L_{r-t-1} has been committed by epoch r, update $\mathcal{Q}(L_{r-t})$ with $(\mathbf{v}, \mathbf{c})$ shared in b_ℓ.

Reconstruct and output. replica P_i starts to reconstruct aggregated secret R_r when replica P_i commits or when its $\mathsf{epoch\text{-}timer}_r$ expires. It obtains $(\mathbf{v}, \mathbf{c})$ corresponding to block committed in $\mathtt{Dequeue}(\mathcal{Q}(L_r))$ and decrypts the share by computing $d_i = c_i^{sk_i^{-1}}$. It then broadcasts d_i to all other replicas. On receiving share d_j from another replica P_j, it verifies it using $\mathsf{PVSS.ShVrfy}(d_j)$. On receiving $t+1$ valid shares in $\mathbf{S}$, it reconstructs $R_r \leftarrow \mathsf{PVSS.Recon}(\mathbf{S})$. In addition, it also broadcasts the aggregated secret R_r. An epoch r beacon output $\mathcal{O}_r$ is the hash of the aggregated secret R_r, i.e., $\mathcal{O}_r \leftarrow H(R_r)$.

Observe that the size of aggregated secret R_r is $O(\kappa)$ and all-to-all broadcast of the aggregated secret does not blow up communication. Moreover, the aggregated secret R_r cannot be reconstructed without an honest replica sending its secret share. Thus, we use the aggregated secret R_r to synchronize all other replicas and responsively change epochs.

Latency and communication complexity. When the epoch leader is Byzantine, not all honest replicas may be locked on a certificate for a common block at the end of the epoch. When the epoch leader is honest, at least one honest replica commits block B_h proposed by an honest epoch leader and all honest replicas lock on a certificate for common block B_h and do not act on block proposals that do not extend B_h afterwards. Thus, block B_h and all its ancestors are finalized in an honest epoch. Due to round-robin leader selection, there will be at least one honest leader every $t+1$ epochs and all honest replicas finalize on common blocks up to the honest epoch. Thus, our protocol has a commit latency of $t+1$ epochs. Our protocol has communication cost of $O((\kappa+w)n^2)$ bits per epoch.

Why is it safe to commit a block B_ℓ proposed $t+1$ epochs earlier if the highest ranked chain extends B_ℓ? The round robin leader selection policy ensures that there will be at least one honest leader in last $t+1$ epochs. An honest epoch leader L_r ensures it extends the highest ranked block certificate from all honest replicas. Our protocol ensures that the block B_h proposed by the leader L_r is committed by at least one honest replica in epoch r and all honest replicas receive and lock on a certificate for block B_h. Thus, no honest replica acts on the future block proposals that do not extend B_h and the highest ranked chain after epoch r always extends B_h, and all its ancestors. This concludes that if block B_ℓ proposed $t+1$ epochs earlier is extended by the highest ranked chain, there will never be an equivocating chain that does not extend B_ℓ and it is safe commit a block B_ℓ.

5.3.2 Safety and Liveness

We say a block B_h is committed directly in epoch r if an honest replica successfully runs one of the following commit rules (i) responsive commit rule (Step 7a), or (ii) synchronous commit rule (Step 7b) in Figure 5.2. We say a block B_h is committed indirectly if it is a result of directly committing a block B_ℓ ($\ell > h$) that extends B_h.

Fact 40. *If an honest replica sends* sync-vote *for block B_h in epoch r, then no equivocating block certificate exists in epoch r.*

Proof. Suppose an honest replica P_i sends a sync-vote for block B_h in epoch r at time τ. Replica P_i must have invoked Deliver(propose, p_r, z_{pa}, r) to deliver proposal p_r for B_h at time $\tau - 2\Delta$ and did not detect an epoch r equivocation by time τ. Observe that no honest replica invoked Deliver and sent resp-vote nor sync-vote for equivocating block proposals before time $\tau - \Delta$; otherwise replica P_i must have received a codeword for equivocating proposal i.e., an epoch r equivocation by time τ. In addition, all honest replicas receive their codeword for proposal p_r by time $\tau - \Delta$ and will neither send resp-vote nor sync-vote for equivocating block proposals after time $\tau - \Delta$. Thus, no equivocating block certificate exists in epoch r. □

Fact 41. *If a responsive certificate for block B_h exists in epoch r, then no equivocating block certificate exists in epoch r.*

Proof. A responsive certificate for block B_h in epoch r, i.e., $\mathcal{C}_r^{3/4}(B_h)$ requires resp-vote from $\lfloor 3n/4 \rfloor + 1$ replicas in epoch r. A simple quorum intersection argument shows that a responsive certificate for an equivocating block $B'_{h'}$ cannot exist in epoch r.

Suppose for the sake of contradiction, an equivocating synchronous block certificate $\mathcal{C}_r^{1/2}(B'_{h'})$ for block $B'_{h'}$ exists in epoch r. At least one honest replica, say replica P_i, must have sent sync-vote for $B'_{h'}$ in epoch r. By Fact 40, an equivocating block certificate i.e., $\mathcal{C}_r^{3/4}(B_h)$ cannot exists. However, since $\mathcal{C}_r^{3/4}(B_h)$ exists, replica P_i must not have sent sync-vote for $B'_{h'}$ in epoch r. A contradiction. □

Lemma 42. *If an honest replica directly commits a block B_h in epoch r using the responsive commit rule (Step 7b), then (i) no equivocating block certificate exists in epoch r, and (ii) all honest replicas receive a block certificate for B_h before entering epoch $r + 1$.*

Proof. Suppose an honest replica P_i directly commits a block B_h in epoch r using responsive commit rule at time τ. Replica P_i must have received $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle$ for B_h from a set R of $\lfloor 3n/4 \rfloor + 1$ replicas when its $\mathsf{epoch\text{-}timer}_r \geq 2\Delta$ and detected no epoch r equivocation by time τ. At least $\lfloor n/4 \rfloor + 1$ of them are honest and have received $\mathcal{C}_r^{3/4}(B_h)$ (corresponding to accumulation value z_{ra}). By Fact 41, there does not exist an equivocating block certificate in epoch r. This proves part(i) of the Lemma.

For part (ii), observe that replica P_i has its $\mathsf{epoch\text{-}timer}_r \geq 2\Delta$ at time τ. Since, honest replicas are synchronized within Δ time, honest replicas that are still in epoch r at time τ must have $\mathsf{epoch\text{-}timer}_r \geq \Delta$ at time τ. In addition, since replica P_i did not detect an epoch r equivocation at time τ, no honest replica detected epoch r equivocation before time $\tau - \Delta$ and did not reset their $\mathsf{epoch\text{-}timer}_r$ to 2Δ before time $\tau - \Delta$. Thus, honest replicas that are still in epoch r at time τ must have $\mathsf{epoch\text{-}timer}_r \geq \Delta$ at time τ. Since, replica P_i received $\langle \mathsf{ack}, H(B_h), z_{ra}, r\rangle$ from a set R' of at least $\lfloor n/4 \rfloor + 1$ honest replicas at time τ, replicas in R' must have invoked $\mathsf{Deliver}(\mathsf{resp\text{-}cert}, rc, z_{ra}, r)$ for $rc := \mathcal{C}_r^{3/4}(B_h)$ by time τ and their codewords for $\mathcal{C}_r^{3/4}(B_h)$ arrives all honest replicas by time $\tau + \Delta$.

Suppose for the sake of contradiction, some honest replica P_j did not receive an epoch r block certificate before entering epoch $r + 1$. If replica P_j entered epoch $r + 1$ when its $\mathsf{epoch\text{-}timer}_r$ expired, replica P_j must be in epoch r at time $\tau + \Delta$ (since, its $\mathsf{epoch\text{-}timer}_r \geq \Delta$ at time τ) and must have received $\lfloor n/4 \rfloor + 1$ valid codewords for $\mathcal{C}_r^{3/4}(B_h)$ sufficient to reconstruct $\mathcal{C}_r^{3/4}(B_h)$ by time $\tau + \Delta$. A contradiction. On the other hand, if replica P_j enters epoch $r + 1$ by receiving a aggregated secret R_r before its $\mathsf{epoch\text{-}timer}_r$ expired, it waits for an epoch r block certificate. By Fact 41, there does not exist an equivocating block certificate in epoch r. Thus, the only block certificate replica P_j can receive is $\mathcal{C}_r^{1/2}(B_h)$ or $\mathcal{C}_r^{3/4}(B_h)$. If replica P_j has not received any block certificate, replica P_j receives $\lfloor n/4 \rfloor + 1$ valid codewords for $\mathcal{C}_r^{3/4}(B_h)$ by time $\tau + \Delta$ sufficient to reconstruct $\mathcal{C}_r^{3/4}(B_h)$. Again a contradiction. This proves part(ii) of the Lemma. □

Lemma 43. *If an honest replica directly commits a block B_h in epoch r using synchronous commit rule (Step 7b), then (i) no equivocating block certificate exists in epoch r, and (ii) all honest replicas receive a block certificate for B_h before entering epoch $r + 1$.*

Proof. Suppose an honest replica P_i directly commits a block B_h in epoch r at time τ. Replica P_i must have received either a synchronous certificate i.e., $\mathcal{C}_r^{1/2}(B_h)$ (or a responsive certificate i.e., $\mathcal{C}_r^{3/4}(B_h)$) at time $\tau - 2\Delta$ when its $\mathsf{epoch\text{-}timer}_r \geq 3\Delta$ and did not detect an epoch r equivocation by time τ. If replica P_i received $\mathcal{C}_r^{1/2}(B_h)$, at least one honest replica must have sent $\mathsf{sync\text{-}vote}$ for B_h and by Fact 40, no equivocating block certificate exists in epoch r. Similarly, if replica P_i

received $\mathcal{C}_r^{3/4}(B_h)$, by Fact 41, there does not exist an equivocating block certificate in epoch r. This proves part(i) of the Lemma.

For part (ii), observe that replica P_i must have invoked Deliver(sync-cert, sc, z_{ve}, e) for $sc = \mathcal{C}_r^{1/2}(B_h)$ (or Deliver(resp-cert, rc, z_{ve}, e) for $rc = \mathcal{C}_r^{3/4}(B_h)$) at time $\tau - 2\Delta$ and did not detect an epoch r equivocation by time τ. Moreover, no honest replica received aggregated secret R_r along with $\mathcal{C}_r(B_h)$ before time $\tau - \Delta$; otherwise replica P_i must have received aggregated secret R_r before time τ and having already received $\mathcal{C}_r(B_h)$, would not commit using synchronous commit rule. Observe that no honest replica detected an epoch r equivocation by time $\tau - \Delta$; otherwise, replica P_i must have detected the equivocation by time τ and would not commit. Thus, all honest replicas will receive and forward their codewords for $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) by time $\tau - \Delta$ and all honest replicas will receive at least $t+1$ codewords sufficient to decode $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) by time τ.

Since, all honest replicas are synchronized within Δ time, all other honest replicas must have $\mathsf{epoch\text{-}timer}_r \geq 2\Delta$ at time $\tau - 2\Delta$. Thus, honest replicas that quit epoch r when their $\mathsf{epoch\text{-}timer}_r$ expired must still be in epoch r at time τ and receive $\mathcal{C}_r^{1/2}(B_h)$ (or $\mathcal{C}_r^{3/4}(B_h)$) before entering epoch $r+1$. If some honest replica, say replica P_j, enters epoch $r+1$ by receiving aggregated secret R_r, it waits for an epoch r block certificate. By part(i) of the Lemma, there does not exist an equivocating block certificate in epoch r. Thus, the block certificate must be either $\mathcal{C}_r^{3/4}(B_h)$ or $\mathcal{C}_r^{1/2}(B_h)$. This proves part(ii) of the Lemma. □

Lemma 44. *If an honest replica directly commits a block B_h in epoch r, then (i) no equivocating block certificate exists in epoch r, and (ii) all honest replicas receive and lock on $\mathcal{C}_r(B_h)$ before entering epoch $r+1$.*

Proof. Straight forward from Lemma 42 and Lemma 43. □

Lemma 45 (Unique Extensibility)**.** *If an honest replica directly commits a block B_h in epoch r, then any certified blocks that ranks higher than $\mathcal{C}_r(B_h)$ must extend B_h.*

Proof. The proof is by induction on epochs $r' > r$. For an epoch r', we prove that if a $\mathcal{C}_{r'}(B_{h'})$ exists then it must extend B_h.

For the base case, where $r' = r+1$, the proof that $\mathcal{C}_{r'}(B_{h'})$ extends B_h follows from Lemma 44. The only way $\mathcal{C}_{r'}(B_{h'})$ for $B_{h'}$ forms is if some honest replica votes for $B_{h'}$. However, by Lemma 44, there does not exist any equivocating block certificate in epoch r and all honest replicas receive

and lock on $\mathcal{C}_r(B_h)$ before quitting epoch r. Thus, a block certificate cannot form for a block that does not extend B_h.

Given that the statement is true for all epochs below r', the proof that $\mathcal{C}_{r'}(B_{h'})$ extends B_h follows from the induction hypothesis because the only way such a block certificate forms is if some honest replica votes for it. An honest replica votes in epoch r' only if $B_{h'}$ extends a valid certificate $\mathcal{C}_{r''}(B_{h''})$. Due to Lemma 44 and the induction hypothesis on all block certificates of epoch $r < r'' < r'$, $\mathcal{C}_{r'}(B_{h'})$ must extend B_h. □

Lemma 46. *Let B_h be a block proposed in epoch r. If the leader of an epoch r is honest, then at least one honest replica commits block B_h and all honest replicas lock on $\mathcal{C}_r(B_h)$ before entering epoch $r+1$.*

Proof. Suppose leader L_r of an epoch r is honest. Let τ be the earliest time when an honest replica P_i enters epoch r. Due to Δ delay between honest replicas, all honest replicas enter epoch r by time $\tau + \Delta$. Some honest replicas might have received a higher ranked certificate than leader L_r before entering epoch r; thus, they send their highest ranked certificate to leader L_r.

Leader L_r needs to ensure that it has the highest ranked certificate before proposing in epoch r. If L_r has $\mathcal{C}_{r-1}(B_l)$, $\mathcal{C}_{r-1}(B_l)$ is already the highest ranked certificate and L_r proposes immediately. Otherwise, it waits for 2Δ time to ensure it can receive highest ranked certificates $\mathcal{C}_{r'}(B_l)$ from all honest replicas. If L_r entered epoch r Δ time after replica P_i, L_r sends a valid proposal $p_r = \langle \mathsf{propose}, B_h, r, \mathcal{C}_{r'}(B_l), z_{pa} \rangle_{L_r}$ by time $\tau + 3\Delta$ which arrives all honest replicas by time $\tau + 4\Delta$.

In any case, all honest replicas receive a valid proposal B_h that extends the highest ranked certificate while satisfying the constraint $\mathsf{epoch\text{-}timer}_r \geq 7\Delta$. Thus, all honest replicas will send $\mathsf{resp\text{-}vote}$ to leader L_r. In addition, all honest replicas will invoke $\mathsf{Deliver}(\mathsf{propose}, p_r, z_{pa}, r)$ and set $\mathsf{vote\text{-}timer}_r$ to 2Δ which expires by time $\tau + 6\Delta$. If $\lfloor 3n/4 \rfloor + 1$ replicas send $\mathsf{resp\text{-}vote}$ to L_r, leader L_r can immediately broadcast $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r \rangle_{L_r}$ to all other replicas. All honest replicas will then broadcast ack to all other replicas. If an honest replica, say replica P_i, receives $\lfloor 3n/4 \rfloor + 1$ ack messages, it commits responsively. Observe that all honest replicas receive $\mathcal{C}_r^{3/4}(B_h)$ from the leader before epoch r ends. On the other hand, if no honest replica received $\lfloor 3n/4 \rfloor + 1$ ack messages, all honest replicas will set their $\mathsf{commit\text{-}timer}_r$ to 2Δ on receiving $\langle \mathsf{resp\text{-}cert}, \mathcal{C}_r^{3/4}(B_h), z_{ra}, r \rangle_{L_r}$. Since, no equivocation exists in epoch r, at least one honest replica (the earliest honest replica), say replica P_i, will commit in epoch r. All other replicas will either commit when their $\mathsf{commit\text{-}timer}_r$ expires or move to epoch $r+1$ on receiving aggregated secret R_r. In either case, they lock on $\mathcal{C}_r^{3/4}(B_h)$ and satisfies the Lemma.

Next we discuss the case when optimistic conditions are not met, i.e., $\lfloor 3n/4 \rfloor + 1$ replicas do not send resp-vote for B_h or do not send ack messages. In this case, all honest replicas will at least send sync-vote for B_h to L_r which arrives L_r by time $\tau + 7\Delta$. Leader L_r forwards $\mathcal{C}_r(B_h)$ which arrives all honest replicas by time $\tau + 8\Delta$. Note that all honest replicas satisfy the constraint $\mathsf{epoch\text{-}timer}_r \geq 3\Delta$ and honest replicas set their $\mathsf{commit\text{-}timer}_r$ to 2Δ which expires by time $\tau + 10\Delta$. Moreover, no equivocation exists in epoch r. Thus, the earliest honest replica, say replica P_j that sets $\mathsf{commit\text{-}timer}_r$ will commit. All other replicas will either commit when their $\mathsf{commit\text{-}timer}_r$ expires or move to epoch $r + 1$ on receiving aggregated secret R_r. In either case, they lock on $\mathcal{C}_r^{1/2}(B_h)$ and satisfies the Lemma. □

Lemma 47. *If an honest replica commits a block B_ℓ proposed in epoch $r - t$ at the end of epoch r such that the highest ranked chain in epoch r extends B_ℓ, then any certified blocks in epoch r or higher must extend B_ℓ.*

Proof. Due to round robin leader election, there will at least one honest leader between epoch $r - t$ and r, say epoch r'. By Lemma 46, at least one honest replica directly commits a block B_h proposed in epoch r'. By Lemma 44 all honest replicas lock on $\mathcal{C}_{r'}(B_h)$ and do not vote for blocks that do not extend B_h. By Lemma 45, any certified blocks that ranks higher than $\mathcal{C}_{r'}(B_h)$ must extend B_h. Thus, the highest ranked chain at the end of epoch r must extend $\mathcal{C}_{r'}(B_h)$.

Since, the highest ranked chain at the end of epoch r extends B_ℓ and B_ℓ was proposed at epoch $r - t < r'$, B_h extend B_ℓ. By Lemma 45, any certified blocks that ranks higher than $\mathcal{C}_{r'}(B_h)$ must extend B_h. Thus, any certified blocks in epoch r or higher must extend B_ℓ. □

Theorem 48 (Safety)**.** *Honest replicas do not commit conflicting blocks for any epoch r.*

Proof. Suppose for the sake of contradiction two distinct blocks B_h and B'_h are committed in epoch r. Suppose B_h is committed as a result of $B_{h'}$ being directly committed in epoch r' and B'_h is committed as a result of $B'_{h''}$ being directly committed in epoch r''. Without loss of generality, assume $h' < h''$. Note that all directly committed blocks are certified. By Lemma 45 and Lemma 47, $B'_{h''}$ extends $B_{h'}$. Therefore, $B_h = B'_h$. □

Theorem 49 (Liveness)**.** *All honest replicas keep committing new blocks.*

Proof. For any epoch r, if the leader L_r is Byzantine, it may not propose any blocks or propose equivocating blocks. Whenever an honest leader is elected in epoch r, by Lemma 46, at least one honest replica commits block B_h proposed in epoch r and all other honest replicas lock on $\mathcal{C}_r(B_h)$

proposed in epoch r i.e., all honest replicas add block B_h proposed in epoch r. Since we assume a round-robin leader rotation policy, there will be an honest leader every $t+1$ epochs, and every time an honest leader is selected, all honest replicas keep committing new blocks. □

Lemma 50 (Communication complexity)**.** *Let κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O((\kappa+w)n^2)$ bits per epoch.*

Proof. At the start of an epoch r, each replica sends a highest ranked certificate and $O(\kappa n)$-sized PVSS tuple to leader L_r. Since, size of each certificate and PVSS tuple is $O(\kappa n)$, this step incurs $O(\kappa n^2)$ bits communication. A proposal for a block B_h consists of $O(\kappa n)$-sized aggregated PVSS tuple, and a block certificate of size $O(\kappa n)$. Proposing $O(\kappa n)$-sized block to n replicas incurs $O(\kappa n^2)$. Delivering $O(\kappa n)$-sized message has a cost $O((\kappa+w)n^2)$, since each replica broadcasts a codeword of size $O((\kappa n)/n)$, a witness of size w and an accumulator of size κ.

In Resp cert step, the leader broadcasts a responsive certificate for block B_h, i.e, $\mathcal{C}_r^{3/4}(B_h)$ which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_r^{3/4}(B_h)$ incurs $O((\kappa+w)n^2)$ bits. Again, in Sync cert step, the leader broadcasts a synchronous certificate for block B_h, i.e, $\mathcal{C}_r^{1/2}(B_h)$ which incurs $O(\kappa n^2)$ communication. Delivering $O(\kappa n)$-sized $\mathcal{C}_r^{1/2}(B_h)$ incurs $O((\kappa+w)n^2)$ bits. In Ack step, replicas perform an all-to-all broadcast of κ-sized accumulator which incurs $O(\kappa n^2)$ bits communication.

During reconstruction, replicas broadcast κ-sized secret shares and w-sized witness. All-to-all broadcast of secret shares and witness incur $O((\kappa+w)n^2)$ bits communication. In addition, replicas broadcast $O(\kappa)$-sized aggregated secret R_r at the end of epoch r which incurs $O(\kappa n^2)$. Hence, the total cost is $O((\kappa+w)n^2)$ bits. □

5.4 Related Work

There has been a long line of work in improving communication complexity of consensus protocols [71, 50, 4, 104, 8, 84] and round complexity of consensus protocols [43, 4, 17, 50, 55, 71, 94]. We review the most recent and closely related works below. Compared to all of these protocols, our protocol incurs $O(\kappa n^2)$ communication per consensus decision while avoiding the use of threshold signatures. Moreover, our protocol is optimistically responsive with a responsive commit latency of 4δ and synchronous commit latency of $4\Delta+3\delta$ in common case when messages arrive at network speed (or 7Δ in the worst case). Our protocol follows rotating leader paradigm and can change

leaders in optimistically responsive manner.

With respect to the communication complexity, the state-of-the-art synchronous BFT SMR protocols [4, 7, 100, 9] incur quadratic communication per consensus decision while using threshold signatures. Without threshold signatures, they incur cubic communication per consensus decision. To the best of our knowledge, the only optimally resilient protocol to achieve $O(\kappa n^2)$ communication without threshold signature is BFT SMR protocol presented in chapter 4. However, the protocol is not responsive even under optimistic conditions and commits a decision every 11Δ time.

With respect to optimistic responsiveness, protocols due to Thunderella [90] and Sync HotStuff [7] are presented in a back-and-forth slow-path–fast-path paradigm. If started in the wrong path, these protocol cannot commit responsively. Recent work such as PiLi [35], OptSync [100] and Hybrid-BFT [83] achieve simultaneity between responsive and synchronous modes. However, they incur cubic communication without the use of threshold signatures. Ours is the first work that achieves simultaneity under synchrony assumption with $O(\kappa n^2)$ communication while avoiding threshold signatures.

OptSync. OptSync [100] (Chapter 3) presents an optimistically responsive protocol with optimal 2δ latency during responsive commit and 2Δ synchronous latency. However, the protocol follow stable leader paradigm and incur synchronous delay of 2Δ while changing leaders. We also provide a separate protocol that support changing leaders in optimistically responsive manner in $O(\delta)$ time. Compared to the protocols in Chapter 3, OptRand can change leaders responsively only when the new leader has highest ranked certificate; otherwise our protocol incurs 2Δ wait.

Hybrid-BFT. Hybrid-BFT [83] presents an optimistically responsive protocol with both responsive and synchronous commit paths existing simultaneously. They also follow rotating leader paradigm and has responsive commit latency of 2δ and synchronous commit latency of $2\Delta + 2\delta$. Similar to our work, their protocol can also change leaders in responsive manner only when the new leader has highest ranked certificate; otherwise the protocol waits for 2Δ time.

Chapter 6

Synchronous Distributed Key Generation without Broadcasts

6.1 Introduction

The problem of distributed key generation (DKG) is setting up a common public key and its corresponding secret keys among a set of participating parties without a trusted entity. DKG protocols are used to reduce the number of trust assumptions placed in cryptographic protocols such as threshold signatures [24, 99] and threshold encryption schemes [39]. These threshold cryptosystems can themselves be used to implement random beacons [44, 29], reduce the complexity of consensus protocols [104, 7], in multiparty computation protocols [66, 67], or to outsource management of secrets to multiple, semi-trusted authorities [46, 76].

Given its widespread applications and their recent adoption in practice (e.g., [44]), we need efficient solutions for DKG. An ideal solution for DKG would have low communication complexity, low latency, optimal resilience, and provide uniform randomness of generated keys such that the generated keys can be useful in a wider class of cryptosystems while being secure. This work focuses on the synchronous network setting where messages sent by a sender will arrive at a receiver within a single round. Synchronous protocols have the advantage of tolerating up to a minority corruption. While a myriad of DKG protocols [93, 59, 30, 87, 64] have been proposed in this setting, existing solutions fall short in one way or the other. For example, Pedersen's DKG [93] produces non-uniform keys in the presence of the adversary, the DKG protocol due to Gennaro et al. [59] has high latency as it requires additional secret sharing using Feldman's VSS [48], and the protocol

due to Gurkhan et al. [64] does not generate keys for discrete log-based cryptosystems.

Moreover, all the DKG protocols considered in the synchronous model assume a *broadcast channel* (that provides a consensus abstraction) and invoke $\Omega(n)$ broadcasts across two or more rounds [14], where n is the number of parties. Since the best-known Byzantine consensus protocols with optimal resilience incur at least $O(\kappa n^3)$ communication (κ is a security parameter) in the absence of DKG-based threshold signatures, instantiating a broadcast channel with state-of-the-art Byzantine broadcast [43, 4] or Byzantine agreement [71] trivially blows up the communication complexity to $O(\kappa n^4)$. Moreover, due to the use of multiple broadcast channel rounds, the latency of such protocols in a point-to-point network setting has not been explored. This leaves us with the following open question: *Can we design a synchronous DKG protocol supporting a wide class of cryptosystems with $o(\kappa n^4)$ communication complexity, good latency, and tolerating a minority corruption?*

We answer this question positively by showing two DKG protocols for discrete log-based cryptosystems each with $O(\kappa n^3)$ communication complexity. The first protocol has worst-cast $O(\kappa n^3)$ communication and $O(t)$ rounds whereas the second protocol has expected $O(\kappa n^3)$ communication and constant rounds in expectation.

6.1.1 Key Technical Ideas and Results

Our DKG protocols avoid the broadcast channel assumption and use a Byzantine consensus process in a non-black-box fashion to achieve $O(\kappa n^3)$ communication. Compared to the existing broadcast-based DKG protocols which require $\Omega(n)$ broadcasts over two or more rounds, our protocols require a single invocation of consensus instance. While DKG protocols [73, 5] without broadcast channel assumption have been explored in the asynchronous model, they either incur high communication [73] or do not generate keys for discrete log-based cryptosystems [5] or use stronger cryptographic assumptions [38]. More importantly, protocols designed for asynchronous or partially-synchronous settings can only tolerate up to $t < n/3$ Byzantine failures, which is sub-optimal for many DKG applications such as random beacons [44]. In the synchronous model, we provide the first solutions to DKG without a broadcast channel with all the desirable properties with $O(\kappa n^3)$ communication.

A typical approach among existing works is to perform n parallel verifiable secret sharings [48, 92] such that all honest parties agree on a common set of qualified parties `QUAL` who correctly performed secret sharing and then compute final public key and secret keys from the secret shares of all parties in `QUAL`. In our protocols, we replace broadcast channels with weaker primitives such

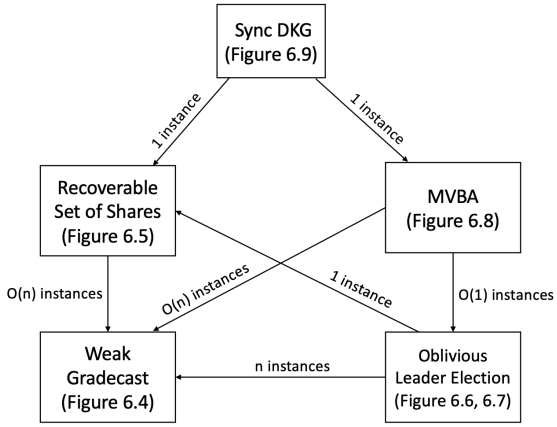


Figure 6.1: Overview of sub-protocols and their dependencies

as gradecast [49, 71]. Thus, parties first perform secret sharing by using this weaker primitive to identify a set of at least $n-t$ parties who correctly shared their secrets, where t is the fault tolerance. During the sharing phase, no consensus primitives are invoked to agree on the set of qualified parties. The downside of this approach is that different honest parties may have different views regarding the acceptance of shared secrets. As a result, different honest parties obtain different sets of at least $n-t$ parties (say $\mathsf{AcceptList}_i$ for party P_i) who they accept to have performed secret sharing correctly. For DKG, it is required that all honest parties compute the final public key and secret keys from a common set of parties. Thus, we need to agree on a common set of parties too. Parties then use a Byzantine consensus primitive to agree on one common set where the input is their individual $\mathsf{AcceptList}$. Once, the Byzantine consensus primitive terminates and outputs a common set $\mathsf{AcceptList}_k$, the final public key and secret keys are computed from $\mathsf{AcceptList}_k$. Note that this approach requires only a single instance of Byzantine consensus.

Key Building Blocks

1. Communication optimal weak gradecast. As a building block, we first provide a communication optimal weak gradecast protocol satisfying the gradecast definition of Katz and Koo [71][1]. Our weak gradecast protocol incurs $O(n\ell + \kappa n^2)$ communication for ℓ bit input and does not require use of threshold signatures. In the same setting, the gradecast protocol of Katz and Koo [71] incurs a communication complexity of $O(\kappa n^3)$ even for a single bit input. Specifically, we show the

[1]This definition is slightly weaker than the one presented by Feldman and Micali [49].

Table 6.1: **Comparison of related works on Distributed Key Generation**

		Net.	Res.	Comm.	Round	Sim.	Dlog	Setup	Crypto Assumption
Pedersen	[93]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	DL
Gennaro et al.	[59]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	DL
Canetti et al.	[30]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	DL
Neji et al.	[87]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+CDH
ETHDKG	[96]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	RO+CDH
Gurkhan et al.	[64]	sync.	$\log n$	$\tilde{O}(\kappa n^3)$	$O(t)$	✗	✗	PKI	RO+SXDH+CBDH
NIDKG	[61]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+DDH+...*
Hybrid-DKG	[69]	psync.	1/3	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+DL
Kokoris et al.	[73]	async.	1/3	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	RO+DDH
Abraham et al.	[5]	async.	1/3	$\tilde{O}(\kappa n^3)$	$E(O(1))$	✗	✗	PKI	RO+SXDH
Das et al.	[38]	async.	1/3	$O(\kappa n^3)$	$E(O(\log n))$	✓	✓	PKI	RO+DCR+DDH
Das et al.	[37]	async.	1/3	$O(\kappa n^3)$	$E(O(\log n))$	✓	✓	PKI	RO+DL
Our work (§ 6.9.1)		sync.	1/2	$O(\kappa n^3)$	$E(O(1))$	✓	✓	PKI+PoT	RO+CDH+q-SDH
Our work (§ 6.9.2)		sync.	1/2	$O(\kappa n^3)$	$O(t)$	✓	✓	PKI+PoT	RO+q-SDH

κ is the security parameter. **Net.** refers to the network model. **Res.** refers to the number of Byzantine faults tolerated in the system. **Comm.** refers to the communication complexity. **Sim.** means the protocol maintains secrecy which can be proven via a simulator. **Primitive** refers to the cryptographic primitives used. **PoT** refers to the power of tau setup required for bilinear accumulators. This setup can be removed by making use of Merkle trees at the cost of $\log n$ multiplicative communication overhead. **E(.)** implies "in expectation". *NIDKG assumes RO, rleaf-IND-CCA, DDH, Erasures, and one-more DH.

following result:

Theorem 51 (Informal)**.** *Assuming a public-key infrastructure and a universal structured reference string under q-SDH assumption, there exists a gradecast protocol for an input of size ℓ bits with $O(n\ell + \kappa n^2)$ communication tolerating $t < n/2$ Byzantine faults.*

2. Recoverable set of shares using weak gradecast. We use the gradecast primitive to perform communication efficient secret sharing. A consequence of using gradecast (instead of broadcast channels) is that parties may have different views regarding the acceptance of the shared secrets. For instance, each party P_i outputs a different set $\mathsf{AcceptList}_i$ and this set may also contain Byzantine parties. However, we still do guarantee that for any set output by any party (including Byzantine parties), there is verifiable proof vouching that all parties in the set have correctly shared their secrets and these secrets are thus recoverable. We call this sub-protocol "Recoverable set of shares". Using our communication optimal gradecast, our recoverable set of shares protocol can be achieved in $O(\kappa n^3)$ communication and constant rounds.

3. Oblivious leader election. We design a communication efficient oblivious leader election (OLE) protocol (aka, common coin) with $O(\kappa n^3)$ communication and constant rounds. The OLE protocol elects a common honest leader with probability at least $\frac{1}{2}$. While OLE protocols have been

Table 6.2: **Comparison of related works on MVBA with ℓ-bit input**

		Network	Resilience	Communication	Round	Assumption
Cachin et al.	[28]	async.	1/3	$O(n^2\ell + \kappa n^2 + n^3)$	$E(O(1))$	Threshold setup
VABA	[8]	async.	1/3	$O(n^2\ell + \kappa n^2)$	$E(O(1))$	Threshold setup
DUMBO-MVBA	[79]	async.	1/3	$O(n\ell + \kappa n^2)$	$E(O(1))$	Threshold setup
Our work		sync.	1/2	$O(n^2\ell + \kappa n^3)$	$E(O(1))$	PKI

E(.) implies "in expectation".

designed in the past, they either required n^2 weaker VSS instances with $\Omega(n^4)$ communication [71] or required stronger cryptographic assumptions to achieve $O(\kappa n^3)$ communication [5]. In this work, we build an OLE protocol using only n weaker VSS instances. Our OLE protocol makes use of n weaker VSS instance and a non-interactive threshold signature scheme [29] to generate randomness. The threshold signature scheme requires a prior threshold setup which is essentially a DKG setup. To circumvent this necessity, we make use of the AcceptList output by parties in the recoverable set of shares protocol as an intermediate threshold setup for each party. The intermediate threshold setup suffices to use the threshold signature scheme and generate the required randomness for each party. Our OLE protocol works in the random oracle and has CDH assumption. In particular, we show the following:

Theorem 52 (Informal)**.** *Assuming a public-key infrastructure, a universal structured reference string under q-SDH assumption, random oracle, and CDH, there exists an oblivious leader election protocol with $O(\kappa n^3)$ communication and $O(1)$ rounds tolerating $t < n/2$ Byzantine faults.*

4. Agreeing on a recoverable set of shares using efficient multi-valued validated Byzantine agreement. Our next goal is to agree on one such set output by one of the parties. We stress that due to the proof associated with the output of the recoverable set of shares protocol, we can agree on the set output by any party, including a Byzantine party. However, here, the size of the set and its proof is linear, which can potentially worsen the communication complexity again. Thus, we need a consensus primitive that takes long messages as inputs and outputs one of the "valid" input values. Such a primitive is called *multi-valued validated Byzantine agreement* (MVBA) [28] in the literature.

MVBA was first formulated by Cachin et al. [28] to allow honest parties to decide on any externally valid values. Recent works [8, 79] have given communication efficient protocols for MVBA in the asynchronous model tolerating $t < n/3$ Byzantine faults. For long messages of size ℓ, the protocol due to Abraham et al. [8] incurs $O((\ell+\kappa)n^2)$ communication and the protocol due to Lu et al. [79]

incurs $O(n\ell + \kappa n^2)$. Both of these works assume a threshold setup. Without threshold setup assumptions, the communication blows up by a factor of n in all of the above protocols.

To the best of our knowledge, no MVBA protocols have been formulated in the synchronous model tolerating $t < n/2$ faults. Recently, Nayak et al. [86] provides an efficient BA protocol for long messages. However, since it is a BA protocol, they output a value only when all honest parties start with the same large input. We construct the first MVBA protocol in the synchronous setting without threshold setup. Our MVBA protocol incurs expected $O(n^2\ell + \kappa n^3)$ communication for inputs of ℓ bit and expected 36 rounds. Specifically, we show the following result:

Theorem 53 (Informal)**.** *Assuming a public-key infrastructure, random oracle, CDH, and a universal structured reference string under q-SDH assumption, there exists a multi-valued validated Byzantine agreement protocol for an input of size ℓ with expected $O(n^2\ell + \kappa n^3)$ communication and expected 36 rounds tolerating $t < n/2$ Byzantine faults.*

Efficient distributed key generation. Using our recoverable set of shares protocol where parties output different sets of size at least $n - t$ parties and our MVBA protocol, honest parties can agree on a common set from which the final public key and secret keys are computed. In particular, we obtain a DKG protocol with expected $O(\kappa n^3)$ communication and expected 47 rounds.

Theorem 54 (Informal)**.** *Assuming public-key infrastructure, random oracle, a universal structured reference string under q-SDH assumption and CDH, there exists a protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with expected $O(\kappa n^3)$ communication and expected 47 rounds.*

Although the DKG protocol terminates in constant expected time, it can take linear time in the worst case. In this case, the protocol incurs $O(\kappa n^4)$ communication. As an alternative, we provide a protocol that incurs $O(\kappa n^3)$ communication in the worst-case. RandPiper [21] provides a BFT SMR protocol with $O(\kappa n^2)$ communication per epoch even for $O(n)$-sized input. Here, an epoch is a period that incurs 7 rounds. In this protocol, we execute the BFT SMR protocol for $t+1$ epochs with each epoch coordinated by a distinct leader. The leader proposes his set AcceptList along with the proof. Honest parties output the first committed set to compute the final public key and secret keys. In particular, we obtain the following result:

Theorem 55 (Informal)**.** *Assuming a public-key infrastructure, and a universal structured reference string under q-SDH assumption there exists a protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with $O(\kappa n^3)$ communication and $11 + 7(t + 1)$ rounds.*

Limitations. In this work, we assume that the adversary is static, similar to several DKGs [68, 59, 93, 87, 96, 64] in the literature. Canetti et al. [30] show how to build adaptively secure DKG protocols and several of our techniques could be applicable in realizing their protocol in the point-to-point network setting. Very recently, Bacho et al. [13] gave a relaxed definition of DKG and show that prior DKG protocols such as Gennaro et al [59] are adaptively-secure under this relaxed definition. It could be interesting to see if our protocols are adaptively-secure under their relaxed definition. In addition, our protocols make the q-SDH assumption. This assumption is only used for bilinear accumulators which could be replaced with Merkle tree accumulators resulting in a $\log n$ multiplicative overhead in the communication complexity.

6.2 Related Work

6.2.1 Related Works in Distributed Key Generation Literature

We review the most recent and closely related DKG protocols. An overview of the closely related work is provided in Table 6.1. While a myriad of DKG protocols [93, 59, 30, 87, 64, 96, 47, 61] have been proposed in the synchronous model, all of these protocols assume a broadcast channel. All of these protocols invoke $\Omega(n)$ parallel broadcasts. A natural choice to instantiate the broadcast channels is via Byzantine consensus primitives such as Byzantine Broadcast [43, 4] or Byzantine agreement [71]. To the best of our knowledge, all optimally resilient deterministic Byzantine consensus protocols incur $O(\kappa n^3)$ communication without threshold signatures and $t+1$ rounds [43]. For randomized consensus protocols, the best known protocol with optimal resilience in this setting is Katz and Koo [71] which incurs $O(\kappa n^4)$ communication. Although, randomized consensus protocols terminate in expected constant rounds, n parallel instances of randomized consensus requires expected $O(\log n)$ rounds to terminate [18]. For the sake of simplicity, we assign a communication of $O(\kappa n^4)$ and $O(t)$ rounds for the DKG protocols that use broadcast channel in Table 6.1. Compared to all prior DKG protocols, our protocols do not use broadcast channel and use Byzantine consensus protocols. In fact, our protocols require a single consensus invocation and incur either expected $O(\kappa n^3)$ communication and expected $O(1)$ rounds or worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds. Our protocols are secure against static failures and generate uniform keys for discrete logarithm based cryptosystems.

We also argue that the protocols by Momose and Ren [84] and Tsimos et al. [103] are relevant but not sufficient to achieve our goals. Momose and Ren [84] gave a deterministic BA protocol with $O(\kappa n^2)$ communication with sub-optimal resilience of $t < (1-\epsilon)n/2$ for a small constant ϵ. Using their BA

protocol to instantiate broadcast channels will result in DKG protocols with $O(\kappa n^3)$ communication but with *sub-optimal* resilience and linear round complexity. Similarly, Tsimos et al. [103] present a communication-efficient broadcast protocol RandomBroadcast in the bulletin PKI setting. It works with $t < (1 - \epsilon)$ resilience, $O(\kappa^2 n^2)$ communication, linear round complexity, and negligible error probability. Using RandomBroadcast to instantiate broadcast channels will result in DKG protocols with optimal resilience, $O(\kappa^2 n^3)$ communication, linear round complexity and negligible error probability. In contrast, our protocols have optimal resilience, $O(\kappa n^3)$ communication and expected $O(1)$ rounds (or $O(t)$ rounds).

Pedersen [93] introduced the first efficient DKG protocol for discrete log cryptosystems in the synchronous setting. Their protocol is based on n parallel invocations of Feldman VSS [48]. Gennaro et al. [59] showed that Pedersen's DKG protocol can be biased by an adversary to generate non-uniform keys. To remove the bias, they proposed a new DKG protocol that requires additional secret sharing rounds; hence, is less efficient. Canneti et al. [30] extended Gennaro et al.'s DKG to handle adaptive corruptions.

Neji et al. [87] presented an efficient DKG protocol to remove the bias without the additional secret sharing round. However, in their protocol, honest parties still need to agree on whether to perform reconstruction for a secret shared by a party which requires additional consensus invocation.

Gurkhan et al. [64] presented DKG protocol without a complaint phase by using publicly verifiable secret sharing (PVSS) [31] scheme. However, they tolerate only $\log n$ Byzantine faults and do not generate keys for discrete-logarithms based cryptosystems; reducing its usefulness.

Recently, Groth [61] presents a non-interactive DKG protocol with a refresh procedure that allows refreshing the secret key shares to a new committee. Erwig et al. [47] considers large scale non-interactive DKG protocol and handles mobile Byzantine faults. Both of above protocols assume broadcast channels.

Several other works tackle the DKG problem from different angels. Kate et al. [69] reduced the size of input to the broadcast channel from $O(n)$ to $O(1)$ by using polynomial commitments [70]. Tomescu et al. [102] reduce the computational cost of dealings in Kate et al. [69] at the cost of a logarithmic increase in communication cost. Schindler et al. [96] instantiate the broadcast channel with the Ethereum blockchain. In Table 6.1, we replaced the Ethereum blockchain with Byzantine consensus primitives for fair comparison.

Kate et al. [69] gave the first practical DKG protocol in the partially synchronous communication model which requires $3t + 2f + 1$ parties to tolerate t Byzantine faults and f crash faults.

Kokoris-Kogias et al. [73] gave the first DKG protocol in asynchronous communication model with optimal resilience ($t < n/3$). Their protocol has $O(\kappa n^4)$ communication and $O(t)$ rounds overhead. Abraham et al. [5] gave an improved DKG protocol with $O(\kappa n^3)$ communication and expected $O(1)$ round complexity. However, their protocol uses PVSS and hence does not generate keys for dlog-based cryptosystems. Das et al. [38] gave the dlog-based DKG protocol with $O(\kappa n^3)$ communication and optimal resilience in the asynchronous model. However, their protocol incurs expected $O(\log n)$ round complexity and requires stronger Decisional Composite Residuosity (DCR) assumption. Very recently, Das et al. [37] gave the dlog-based DKG protocol with $O(\kappa n^3)$ communication and optimal resilience in the asynchronous model with discrete-log assumption. However, their construction still incurs $O(\log n)$ round complexity. We note that while DKG protocols have been designed with lesser assumption (i.e., DL assumption in Das et al. [37]) in the asynchronous model tolerating $t < n/3$ Byzantine failures, designing protocols tolerating $t < n/2$ Byzantine failures presents its own unique challenges and does not make our protocols sub-optimal.

Recent works without broadcast channel assumption. We emphasize the importance of investigating protocols in the synchronous model without broadcast channels. Recent works [21, 22] have studied design of efficient randomness beacon protocols in the synchronous model without broadcast channel assumption.

6.2.2 Related Works in Byzantine Agreement Literature

There has been a long line of work in improving communication and round complexity of consensus protocols [71, 50, 4, 104, 27, 8, 84, 100]. We review the most recent and closely related works.

Multi-valued validated Byzantine agreement was first introduced by Cachin et al. [28] to allow honest parties to agree on any externally valid values. Their protocol works in asynchronous communication model and has optimal resilience ($t < n/3$) with $O(n^2\ell + \kappa n^2 + n^3)$ communication for input of size ℓ. Later, Abraham et al. [8] gave an MVBA protocol with optimal resilience and $O((\ell + \kappa)n^2)$ communication in the same asynchronous setting. Lu et al. [79] extended the work of Abraham et al. [8] to handle long messages of size ℓ with a communication complexity of $O(n\ell + \kappa n^2)$. All of these protocols assumed threshold setup. In the absence of threshold setup, the communication complexity blows up by a factor of n in all of these protocols.

To the best of our knowledge, no MVBA protocol has been formulated in the synchronous setting tolerating $t < n/2$ Byzantine faults. Our MVBA protocol incurs $O(n^2\ell + \kappa n^3)$ for inputs of size ℓ and does not assume threshold setup and terminates in expected constant rounds.

Our MVBA protocol can also be used for binary inputs as a Binary Byzantine Agreement (BBA) protocol tolerating $t < n/2$ Byzantine faults and terminating in expected $O(1)$ rounds. Feldman and Micali [50] were the first to give a BBA protocol that terminates in constant expected rounds. Their protocol works in plain authenticated model without PKI and tolerates $t < n/3$ Byzantine faults (which is optimal). In the authenticated setting, Katz and Koo [71] gave a BBA protocol tolerating $t < n/2$ Byzantine faults terminating in expected constant rounds. Their protocol incurs $O(\kappa n^4)$ communication and terminates in expected 4 epochs. We extend the BBA protocol of Katz and Koo [71] and reduce its communication by linear factor while handling multi-valued input by designing a communication optimal gradecast protocol. A simple and efficient BBA tolerating $t < n/3$ Byzantine faults in the authenticated model was given by Micali [82]. Abraham et al. [4] reduced the round complexity of BBA protocol to expected 10 rounds. However, their protocol required a threshold setup to generate a perfect common coin; a perfect common coin ensures all honest parties output the same random value. Compared to their work, our work does not require a threshold setup and executes with a weak common coin.

Very recently, Abraham et al. [1] gave a BBA protocol in the authenticated model without PKI and digital signatures tolerating $t < n/3$ Byzantine faults. Their protocol has an expected communication complexity of $O(n^4 \log n)$ and expected constant rounds.

6.3 Model and Preliminaries

We consider a system consisting of n parties $(P_1, \ldots, P_n)$ with pair-wise reliable, authenticated point-to-point channels, where up to $t < n/2$ parties can be Byzantine faulty. The model of corruption is static i.e., the adversary picks the corrupted parties before the start of protocol execution. The Byzantine parties may behave arbitrarily. A non-faulty party is said to be *honest* and executes the protocol as specified. We assume a synchronous communication model. Thus, if an honest party sends a message at the beginning of some round, the recipient receives the message by the end of that round.

Setup. Let p be a prime number that is $\mathsf{poly}(\kappa)$ bits long, and $\mathbb{G}$ be a group of order p such that it is computationally infeasible except with negligible probability in κ to compute discrete log. Let $\mathbb{Z}_p$ denote its scalar field. Moreover, let g and h denote the generators of $\mathbb{G}$ where $a \in \mathbb{Z}_p$ such that $g^a = h$ is not known to any t subset of the nodes.

We make the standard computational assumption on the infeasibility to compute discrete logarithms

called the discrete-log assumption [59]. In particular, we assume that the adversary is unable to compute discrete logarithms modulo large (based on the security parameter κ) primes.

We make use of digital signatures and PKI to prevent spoofing and replays and to validate messages. Message x sent by a party P_i is digitally signed by P_i's private key and is denoted by $\langle x \rangle_i$. We denote $H(x)$ to represent invocation of the random oracle H on input x. In addition, we use a hash function $H' : \mathbb{G} \rightarrow \{0,1\}^\kappa$ in our leader election protocol.

Equivocation. Two or more messages of the same *type* but with different payload sent by a party is considered an equivocation. In order to facilitate efficient equivocation checks, the sender sends the payload along with signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by the sender.

6.3.1 Definitions

Distributed key generation. A DKG protocol for n parties $(P_1, \ldots, P_n)$ generates private outputs $(x_1, \ldots, x_n)$ called the *shares* and a public output y.

Definition 6.3.1 (Secure DKG for Dlog based cryptosystems [59])**.** *A dlog based DKG protocol that distributes a secret x among n parties through shares $(x_1, \ldots, x_n)$ where x_i is a share output to party P_i is t-secure if in the presence of an adversary that corrupts up to t parties, the following requirements for correctness and secrecy are maintained.*

Correctness.

C1. *All subsets of $t+1$ shares provided by honest parties define the same unique secret key $x \in \mathbb{Z}_p$.*

C2. *All honest parties have same value of public key $y = g^x \in \mathbb{G}$, where $x \in \mathbb{Z}_p$ is secret guaranteed by (C1).*

C3. *x is uniformly distributed in $\mathbb{Z}_p$ (and hence y is uniformly distributed in $\mathbb{G}$).*

Secrecy. *No information on x can be learned by the adversary except for what is implied by the value $y = g^x$.*

More formally, the secrecy condition is expressed in terms of simulatability: for every (probabilistic polynomial-time) adversary $\mathcal{A}$ that corrupts up to t parties, there exists a (probabilistic polynomial-time) simulator $\mathcal{S}$, such that on input an element $y \in \mathbb{G}$, produces an output distribution which is

polynomially indistinguishable from $\mathcal{A}$'s view of a run of the DKG protocol that ends with y as its public key output.

Weak Gradecast. Weak gradecast is a relaxed version of gradecast [49] introduced by Katz and Koo [71].

Definition 6.3.2 (Weak Gradecast [71])**.** *A protocol with a designated sender P_i holding an initial input v is a weak gradecast protocol tolerating $t < n/2$ Byzantine parties if the following conditions hold*

1. *Each honest party P_j outputs a value v_j with a grade $g_j \in \{0, 1, 2\}$.*
2. *If the sender is honest, each honest party outputs v_i with a grade of 2.*
3. *If an honest party P_i outputs a value v with a grade of 2, then all honest parties output value v with a grade of ≥ 1.*

Oblivious leader election. An oblivious leader election protocol elects a common honest leader with some constant probability.

Definition 6.3.3 (Oblivious Leader Election [71])**.** *A protocol for parties $P_1, \ldots, P_n$ is an oblivious leader election protocol with fairness α tolerating t Byzantine failures if each honest party P_i outputs a value $v_i \in [1, n]$ and the following conditions holds with probability at least α:*

There exists a value $j \in [1, n]$ such that (i) each honest party P_i outputs $v_i = j$, and (ii) party P_j is honest.

6.3.2 Primitives

In this section, we present several primitives used in our protocols.

Linear erasure and error correcting codes. We use standard $(t + 1, n)$ Reed-Solomon (RS) codes [95]. This code encodes $t + 1$ data symbols into code words of n symbols using ENC function and can decode the $t + 1$ elements of code words to recover the original data using DEC function. More details on ENC and DEC are provided in Section 2.5.

Cryptographic accumulators. A cryptographic accumulator scheme constructs an accumulation value for a set of values using Eval function and produces a witness for each value in the set using

CreateWit function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. More details on these functions are provided in Section 2.5.

In this chapter, we use *collision free bilinear accumulators* from Nguyen [88] as cryptographic accumulators which generates constant sized witness, but requires q-SDH assumption. Alternatively, we can use Merkle trees [81] (and avoid q-SDH assumption) at the expense of $O(\log n)$ multiplicative communication.

Non-interactive threshold signature scheme. We use (t, n) *non-interactive threshold signature scheme* of Cachin et al. [29] in one of our protocols. The threshold signature scheme is secure against static adversary. The threshold signature scheme of Cachin et al. [29] consists of following interfaces:

- The randomized *key generation* algorithm $\mathsf{KeyGen_{TS}}$ that takes a security parameter κ as input and outputs a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a common public key pk.
- The deterministic signing algorithm $\mathsf{Sign_{TS}}$ that takes as input sk_i and a message m and outputs a signature σ_i on m.
- The deterministic *share verification* algorithm $\mathsf{ShareVerify_{TS}}$ that takes as input public key pk_i, a signature share σ_i and tuple (i, m). It outputs a bit $b \in \{0, 1\}$ indicating whether σ_i is a valid signature share on m under secret key sk_i.
- The deterministic *combining* $\mathsf{Combine_{TS}}$ takes as input a tuple of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, a message m, and a list of $t + 1$ pairs (i, σ_i). It outputs either a signature σ on m or $\perp$, if (i, σ_i) contains ill-formed signature shares.
- The deterministic *verification* algorithm $\mathsf{Verify_{TS}}$ takes as input a signature σ, a message m and a common public key pk. It outputs a bit $b \in \{0, 1\}$ indicating whether σ is a valid signature on m.

Non-Interactive Proof-of-Equivalence of commitments [69]. Given two commitments $\mathcal{C}_{\langle g \rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h \rangle}(s, r) = g^s h^r$ to the same value s for generators $g, h \in \mathbb{G}$ and $s, r \in \mathbb{Z}_p$, a prover proves that she knows s and r such that $\mathcal{C}_{\langle g \rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h \rangle}(s, r) = g^s h^r$. We denote it by $\mathsf{NIZKPK}_{\equiv \mathsf{Com}}(s, r, g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r)) = \pi_{\equiv \mathsf{Com}} \in \mathbb{Z}_p^3$.

$\mathsf{NIZKPK}_{\equiv \mathsf{Com}}$ is generated as follows:

- Pick $v_1, v_2 \in_R \mathbb{Z}_p$, and let $t_1 = g^{v_1}$ and $t_2 = h^{v_2}$.
- Compute hash $c = \mathsf{H}_{\equiv\mathsf{Com}}(g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r), t_1, t_2)$, where $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash function.
- Let $u_1 = v_1 - c \cdot s$ and $u_2 = v_2 - c \cdot r$.
- Send the proof $\pi_{\equiv\mathsf{Com}} = (c, u_1, u_2)$ along with $\mathcal{C}_{\langle g\rangle}(s)$ and $\mathcal{C}_{\langle g,h\rangle}(s,r)$.

The verifier checks this proof (given $\pi_{\equiv\mathsf{Com}}$, g, h, $\mathcal{C}_{\langle g\rangle}(s)$, $\mathcal{C}_{\langle g,h\rangle}(s,r)$) as follows:

- Let $t_1' = g^{u_1}\mathcal{C}_{\langle g\rangle}(s)^c$ and $t_2' = h^{u_2}(\frac{\mathcal{C}_{\langle g,h\rangle}(s,r)}{\mathcal{C}_{\langle g\rangle}(s)})^c$.
- Accept the proof as valid if $c = \mathsf{H}_{\equiv\mathsf{Com}}(g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r), t_1', t_2')$.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator and $\kappa_v = \kappa_v(\lambda)$ denote the size of secret share and witness of a secret. Further, let $\kappa = \mathsf{max}(\kappa_h, \kappa_a, \kappa_v)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_v) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the chapter, we can use the same parameter κ to denote the hash size, signature size, accumulator size and secret share size for convenience.

6.4 Secure DKG with Two Broadcast Rounds

We first present a secure DKG protocol assuming a broadcast channel motivated from Gennaro et al. DKG [59]. The presented DKG reduces the number of required rounds with broadcast to two, which is a significant improvement over [59] requiring three broadcast rounds in the best case and five broadcast rounds otherwise.[2] In later sections, we replace the broadcast channel with a novel consensus primitives to design communication-efficient DKG protocols.

Gennaro et al. [59] presented a secure DKG protocol that produces uniform public keys based on Pedersen's VSS [92]. In their protocol, each party, as a dealer, selects a secret uniformly at random and shares the secret using Pedersen's VSS protocol. Since Pedersen's VSS provides information theoretic secrecy guarantees, the adversary has no information about the public key and hence

[2]Using NIZK similar to us, the number of rounds for Gennaro et al. DKG [59] can be reduced to two in the best case and three otherwise in a rather straightforward manner; however, reducing to two broadcast rounds in all situations is the key challenge here.

cannot bias it. At the end of the secret sharing, a set of qualified parties QUAL who correctly shared their secret is defined. Once the set QUAL is fixed, parties in set QUAL invoke an additional round of secret sharing using Feldman's VSS [48] to generate the final public key. While this approach ensures generation of uniform keys and maintains secrecy, it adds additional overhead as it incurs more latency and communication to perform additional secret sharing. In addition to the above overhead, Pedersen VSS requires three broadcast rounds. In particular, parties post the commitment, complaints and secret shares corresponding to the complaints on to the broadcast channel during the sharing phase.

The protocol in Figure 6.2 improves upon the DKG protocol of Gennaro et al. [59] in the following ways.

Improving latency in the sharing phase. We improve latency by reducing information posted on the broadcast channel by using improved eVSS (iVSS) protocol [21] which requires only 2 broadcast rounds.[3] Reducing the broadcast rounds greatly improves latency as broadcast channels are generally instantiated using Byzantine broadcast or Byzantine agreement protocols which have worst-case linear round complexity.

In iVSS, the dealer posts commitments on the broadcast channel and privately sends the secret shares to each party. Instead of posting the complaints on the broadcast channel, parties multicast blame message if they receive invalid secret shares or receive no secret shares at all. Parties then forward all blame messages to the dealer[4]. The dealer is expected to send secret shares corresponding to the blame messages (i.e., secret shares s_{ij}, s'_{ij} if a P_j sent blame message against dealer P_i). If the dealer sends all secret shares corresponding to the blame message it forwarded, a party sends a vote message to the dealer. Upon receiving $t+1$ vote messages, the dealer posts a vote-certificate containing $t+1$ vote messages. Honest parties consider the dealer to be honest if they see the vote-certificate on the broadcast channel.

Observe that using iVSS scheme, the dealer posts only the commitment and vote-certificate on the broadcast channel. This improves the sharing phase by one broadcast round.

Using commitments to evaluations instead of commitments to coefficients. In VSS such as Pedersen's VSS and Feldman's VSS and thus in [59], commitments to the secret share are commitments to the coefficients of a t-degree polynomial, which imply verifying a share requires $O(t)$

[3] Alternatively, we can use broadcast optimal VSS protocol of Backes et al. [14] which has 2 broadcast rounds. We prefer iVSS protocol for its simplicity.

[4] In an implementation, we can only forward up to t blames instead of all the blames.

Sharing Phase

1. **Deal.** Each party (as a dealer) P_i selects two random polynomials $f_i(y)$, $f'_i(y) \in \mathbb{Z}_p[y]$ of degree t:

$$f_i(y) = a_{i0} + a_{i1}y + \cdots + a_{it}y^t, \qquad f'_i(y) = b_{i0} + b_{i1}y + \cdots + b_{it}y^t$$

   Let $s_i = a_{i0} = f_i(0)$. Party P_i posts $\mathcal{C}_{ik} = g^{f_i(k)}h^{f'_i(k)}$ $\forall k \in \{1, \ldots, n\}$ on the broadcast channel. Party P_i computes the secret shares $s_{ij} = f_i(j)$, $s'_{ij} = f'_i(j)$ and sends s_{ij}, s'_{ij} privately to P_j $\forall j \in [n]$.

2. **Blame.** Each party P_i verifies that the commitment vector contains a t degree polynomial (Equation (6.2)). For $j \in [n]$, check if

$$g^{s_{ji}} \cdot h^{s'_{ji}} = \mathcal{C}_{ji} \tag{6.1}$$

$$\prod_{k=1}^{n} \mathcal{C}_{jk}^{\mathsf{Code}_k} = 1_{\mathbb{G}}, \text{ where } \{\mathsf{Code}_1, \ldots, \mathsf{Code}_n\} \in C^{\perp} \text{ using Equation (6.4)} \tag{6.2}$$

   If the check fails for (dealer) party P_j, send $\langle \mathsf{blame}, j \rangle_i$ to all parties and collect all the blames.

3. **Forward blame.** If more than t blame messages are collected for party P_j as the dealer in the previous step, do not send anything for dealer P_j until the Decide step (Step 6).

   Otherwise, for every $\langle \mathsf{blame}, j \rangle_k$ received from party P_k, forward the blame messages to the dealer P_j.

4. **Open.** Each party P_i, who as a dealer, received $\langle \mathsf{blame}, i \rangle_k$ from any party P_j, sends valid secret shares s_{ik}, s'_{ik} (that verifies Equation (6.1)) to party P_j.

5. **Vote.** If in Step 2, a party P_i received $\leq t$ $\langle \mathsf{blame}, j \rangle_k$ messages and party P_j sent valid secret shares s_{jk}, s'_{jk} for every $\langle \mathsf{blame}, j \rangle_k$ it forwarded to party P_j, send a vote $\langle \mathsf{vote}, j \rangle_i$ to party P_j. Forward the secret shares s_{jk}, s'_{jk} to party P_k.

6. **Decide.** If party P_i, as a dealer, receives $t + 1$ $\langle \mathsf{vote}, i \rangle$ messages, post the vote-certificate on the broadcast channel.

   Each party P_i marks a party P_j qualified if it receives a vote-certificate for party P_j on the broadcast channel; otherwise the party is disqualified. Party P_i builds a set of non-disqualified parties $\mathtt{QUAL}$.

Generating Public key

7. Party P_i sets its share of the secret as $x_i = \sum_{j \in \mathtt{QUAL}} s_{ji}$, and computes $x'_i = \sum_{j \in \mathtt{QUAL}} s'_{ji}$, $\mathcal{C}_{\langle g \rangle}(x_i) = g^{x_i}$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i) = g^{x_i}h^{x'_i}$ and $\pi_{\equiv \mathsf{Com}\,i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g \rangle}(x_i), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$. Party P_i sends $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\,i})$ to all parties.

8. Upon receiving a tuple $(\mathcal{C}_{\langle g \rangle}(x_j), \pi_{\equiv \mathsf{Com}\,j})$, compute $\mathcal{C}_{\langle g,h \rangle}(x_j, x'_j) = g^{x_j}h^{x'_j}$ locally as follows:

$$g^{x_j}h^{x'_j} = \prod_{m \in \mathtt{QUAL}} \mathcal{C}_{mj} \tag{6.3}$$

   Ensure $\pi_{\equiv \mathsf{Com}\,j}$ verifies $\mathsf{NIZKPK}_{\equiv \mathsf{Com}}$ between $\mathcal{C}_{\langle g \rangle}(x_j)$ and $\mathcal{C}_{\langle g,h \rangle}(x_j, x'_j)$.

9. Upon receiving $t+1$ valid g^{x_j} values, perform Lagrange interpolation in the exponent to obtain $y = g^x$. Output y as the public key and x_i as the private key.

Figure 6.2: **Secure distributed key generation in dlog-based cryptosystems**

computations. This results in $O(nt)$ computations per VSS instance in the complaint stage (where every party verifies opening of up to t complaints) and during reconstruction. SCRAPE [31, Section 2.1] showed how to commit (using discrete log commitments) to evaluations instead of coefficients of the polynomial and verify that the committed evaluations are of a degree t polynomial by using the property of coding schemes: if C is the code space for an (n, t) sharing, then the following vector

$$C^{\perp} := \{\mathsf{Code}_1, \ldots, \mathsf{Code}_n; \mathsf{Code}_i = \mathsf{poly}(i) \prod_{j=1, j\neq i}^{n} 1/(i-j) \\ \mathsf{poly}(x) \text{ is a random polynomial of degree } n-t+1\} \quad (6.4)$$

is orthogonal to C. We can check that the Pedersen's commitments to the evaluations are an (n, t) sharing (see Equation (6.1)). If λ is $\log_g h$, then commitments to evaluations form a polynomial $g^f h^{f'} = g^{f+\lambda f'}$ which is another (n, t) polynomial thereby allowing to use the coding technique. This is an information-theoretic technique and therefore does not affect the security of the underlying VSS.

Removing additional secret sharing while generating public key. We remove the additional secret sharing performed using Feldman's VSS by taking an alternate approach [69]. Instead of executing an additional secret sharing, assuming random oracle, we make use of the NIZK proof of equivalence of commitments $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ to generate the public key. This approach does not require additional secret sharing via Feldman's VSS. Once the sharing phase is completed, a set of qualified parties $\mathtt{QUAL}$ is finalized. Then, each party P_i computes its share of the shared secrets i.e., $x_i = \sum_{P_j \in \mathtt{QUAL}} s_{ji}$ and $x'_i = \sum_{P_j \in \mathtt{QUAL}} s'_{ji}$ along with commitments $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$. It then multicasts commitment of its share $\mathcal{C}_{\langle g \rangle}(x_i)$ and the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}i}$ to prove P_i knows x_i and x'_i.

All parties can compute the commitment $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$ locally as shown in Equation (6.3) and verify the correctness of commitment $\mathcal{C}_{\langle g \rangle}(x_i)$ using $\pi_{\equiv\mathsf{Com}i}$. The final public key Y is computed via Lagrange interpolation in the exponent using $t+1$ distinct commitments $\mathcal{C}_{\langle g \rangle}(x_i)$.

6.4.1 Security Analysis

We rely on the following Lemma of [92].

Lemma 56 ([92])**.** *Under the discrete-log assumption, Pedersen's VSS satisfies following properties*

in the presence of a polynomially bounded adversary that corrupts up to t parties.

(i) If the dealer is not disqualified during the sharing phase, then all honest parties hold secret shares that interpolate to unique polynomial of degree t. In particular, any $t+1$ of these shares suffice to reconstruct the secret σ.

(ii) The protocol produces information (i.e., commitments $\mathcal{C}_k$ and secret shares σ_i) that can be used at reconstruction time to test for the correctness of each secret share; thus, reconstruction is possible, even in the presence of malicious parties, from any subset of shares containing at least $t+1$ correct secret shares.

(iii) The view of the adversary is independent of the value of the secret σ, and therefore the secrecy of σ is unconditional.

Note that Lemma 56 also holds when using evaluations instead of coefficients as discussed in Section 6.9. The coding check (see Equation (6.2)) ensures that the shared commitments to evaluations are indeed a t degree polynomial except with $1/p$ probability in $\mathbb{Z}_p$. Since p is sufficiently large ($\mathsf{poly}(\kappa)$), the probability of the check failing is negligible in the security parameter.

Fact 57. *If a dealer P_i receives a vote-certificate, all honest parties must have received their corresponding secret shares s_{ij}, s'_{ij}.*

Proof. Suppose a dealer P_i receives a vote-certificate i.e, $t+1$ vote messages. At least one of the vote messages is sent by an honest party (say P_j). An honest party P_j sends a vote message only when it receives no blame messages or receives up to t blame messages and dealer P_i sent secret shares s_{ik}, s'_{ik} for every $\langle \mathsf{blame}, i \rangle_k$ message it forwarded.

If party P_j received no blame messages, all honest parties must have received their corresponding secret shares s_{ij}, s'_{ij}; otherwise honest parties would have sent blame messages. On the other hand, if party P_j received $f \leq t$ blame messages, $n-t-f$ honest parties must have received their corresponding secret shares; otherwise, these honest parties would have sent blame messages and party P_j would have received more than f blame messages. Since party P_j forwards secret shares s_{ik}, s'_{ik} to party P_k for every $\langle \mathsf{blame}, i \rangle_k$ message it received, all honest parties must have received corresponding secret shares. □

Theorem 58. *Under discrete-log assumption and random oracle, the protocol in Figure 6.2 is a secure protocol for distributed key generation in dlog-based cryptosystem tolerating $t < n/2$ Byzantine faults.*

Let $\mathcal{B}$ be the set of parties controlled by the adversary, and $\mathcal{G}$ be the set of honest parties (run by the simulator $\mathcal{S}$). Without of loss of generality, let $\mathcal{B} = [P_1, P_{t'}]$ and $\mathcal{G} = [P_{t'+1}, P_n]$, where $t' \geq t$. Let $Y \in \mathbb{G}$ be the input public key and $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash table for $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$.

1. Perform Step 1 through Step 6 on the behalf of the uncorrupted parties $P_{t'+1}, \ldots, P_n$ exactly as secure DKG protocol (refer Figure 6.2) until set $\mathtt{QUAL}$ is finalized. At the end of Step 6, the following holds:
   - Set $\mathtt{QUAL}$ is well-defined with at least one honest party in it.
   - The adversary's view consists of polynomials $f_i(y)$, $f'_i(y)$ for $P_i \in \mathcal{B}$, the secret shares s_{ij}, s'_{ij} for $P_i \in \mathtt{QUAL} \cap \mathcal{G}, P_j \in \mathcal{B}$, and the commitments $\mathcal{C}_i$ for $P_i \in \mathtt{QUAL}$.
   - $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in \mathtt{QUAL}$ as it knows $n - t'$ shares for each of those.
2. Perform the following computations for each $i \in \{t+1, \ldots, n\}$ before Step 6 (refer Figure 6.2).
   (a) Compute x_j for party $P_j \in \mathcal{B}$. Similarly, compute x_j for party $P_j \in [P_{t'+1}, P_t]$. Interpolate in the exponent $(0, Y)$ and (j, g^{x_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g \rangle}(x_i^*) = g^{x_i^*}$.
   (b) Compute the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ by generating random challenges $c_i \in \mathbb{Z}_p$ and responses $u_{i,1}, u_{i,2} \in \mathbb{Z}_p$, computing the commitments $t_{i,1} = (g^{x_i^*})^{c_i} g^{u_{i,1}}$ and $t_{i,2} = \frac{\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)^{c_i}}{\mathcal{C}_{\langle g \rangle}(x_i^*)} h^{u_{i,2}}$ and include entry $\langle (g, h, \mathcal{C}_{\langle g \rangle}(x_i^*), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i), t_{i,1}, t_{i,2}), c_i \rangle$ in the hash table $\mathsf{H}_{\equiv\mathsf{Com}}$ so that $\pi_{\equiv\mathsf{Com}} = (c_i, u_{i,1}, u_{i,2})$.
3. In the end, $x = \sum_{P_i \in \mathtt{QUAL}} s_i$ such that $Y = g^x$.

Figure 6.3: Simulator for Secure DKG

Proof. We first prove correctness of the protocol. Observe that all honest parties build the same set of non-disqualified parties $\mathtt{QUAL}$ in Step 6. This is true because the commitment to the shared polynomials and vote-certificates are posted on the broadcast channel and broadcast channel ensures all honest parties output a common value.

Note that if a party $P_j \in \mathtt{QUAL}$, it must have posted its commitment and vote-certificate on the broadcast channel. By Fact 57, all honest parties have received secret shares shared by party P_j. This implies party P_j is not disqualified during the sharing phase. By part (i) of Lemma 56, all honest parties hold correct secret shares and any $t+1$ of these secret shares suffices to reconstruct the secret s_j. This is true for all parties $P_j \in \mathtt{QUAL}$. Since, the secret key x is sum of individual secret s_j contributed by $P_j \in \mathtt{QUAL}$ and each secret s_j can be reconstructed using Lagrange interpolation via a combination of $t+1$ secret shares provided by honest parties, the secret key x can be reconstructed via $t+1$ shares provided by honest parties. This proves property C1 of a secure DKG protocol.

By part (ii) of Lemma 56, there exists information (i.e., commitments) that can be used to verify correctness of each secret share. Observe that each honest party P_j sends g^{x_j} and $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}j}$ at the end of sharing phase. Each party P_i can verify correctness of $\mathcal{C}_{\langle g \rangle}(x_j)$ by

checking Equation (6.3). A valid $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}j}$ proves in zero knowledge that party P_j knows x_j and x'_j thus proving the correctness of g^{x_j}. By using $t+1$ valid g^{x_j}, honest parties can compute the same g^x via Lagrange interpolation in the exponent which is the public key. This proves property C2 of a secure DKG protocol.

Observe that the secret key x is the sum of secrets shared by parties in `QUAL` which contains at least one honest party and honest parties select their secret uniformly at random. This suffices to prove property C3 of a secure DKG protocol.

We now prove secrecy. Our proof of secrecy is based on the proof of secrecy in earlier works [59, 69]. We provide a simulator $\mathcal{S}$ for our secure DKG protocol in Figure 6.3. Without loss of generality, we assume the adversary $\mathcal{A}$ compromises parties $P_1, \ldots, P_{t'}$, where $t' \leq t$, denoted by set $\mathcal{B}$. The rest of the parties $P_{t'+1}, \ldots, P_n$, denoted by set $\mathcal{G}$ are controlled by the simulator.

Informally, the simulator $\mathcal{S}$ with input Y runs as follows. $\mathcal{S}$ will run on the behalf of the honest parties $\mathcal{G}$ Step 1 until Step 6 following exactly the instructions. At this point, the set `QUAL` is well-defined and $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in \mathtt{QUAL}$ as it knows $n - t'$ shares for each of those. Observe that the view of adversary $\mathcal{A}$ that interacts with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ that interacts with honest parties in a regular run of the protocol. In particular, $\mathcal{A}$ sees the following distribution of data:

- Polynomials $f_i(y)$, $f'_i(y)$ for $P_i \in \mathcal{B}$
- Values $f_i(j)$, $f'_i(j)$ for $i \in \mathcal{G}$, $j \in \mathcal{B}$ and values $\mathcal{C}_i$ for $P_i \in \mathtt{QUAL}$

$\mathcal{S}$ will then change the secret shared by one honest party (say P_n) to "hit" the desired public key Y such that the above data distribution observed by $\mathcal{A}$ remains identical. For parties $P_i \in (\mathcal{G} \setminus \{P_n\})$, the input polynomial $f_i(y)$ and $f'_i(y)$ remains identical. Thus, their data distribution remains identical. For party P_n, the input polynomial is modified such that $g^{f^*_n(0)} = g^{s^*_n} = \frac{Y}{\prod_{P_j \in \mathtt{QUAL} \setminus \{P_n\}} g^{s_i}}$ and $f^*_n(j) = s_{nj}$ for $j \in [1, t]$. Define $f'^*(y)$ such that $f^*_n(y) + \lambda f'^*_n(y) = f_n(y) + \lambda f'_n(y)$, where $\lambda = \log_g(h)$. Observe that for these polynomials, the evaluations and commitments seen by parties in $\mathcal{B}$ is identical to the real run of the protocol.

Simulator $\mathcal{S}$ will then compute g^{x_j} for party $P_j \in [P_1, P_t]$ and interpolate in the exponent $(0, Y)$ and (j, g^{x_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g \rangle}(x^*_i) = g^{x^*_i}$ and the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ and publish these values. Observe that these values pass the verification in the real run of protocol.

It remains to be shown that polynomials $f^*_i(y)$ and $f'^*_i(y)$ belong to the right distribution. For

$\texttt{QUAL} \setminus (\mathcal{G} \setminus \{P_n\})$, this is trivially true as they are defined identically to $f_i(y)$ and $f'_i(y)$ which were chosen uniformly at random. For f^*_n, the polynomial evaluates to random values $f_n(j)$ at $j \in [1, t]$ and evaluates to $\log_g(s^*_n)$ required to hit Y. Finally, $f'^*_n(y)$ is defined as $f^*_n(y) + \lambda f'^*_n(y) = f_n(y) + \lambda f'_n(y)$, and since $f'_n(y)$ is chosen to be random, so is $f'^*_n(y)$. □

6.5 Communication Optimal Weak Gradecast

One of the main tools in the design of our communication efficient protocols is our communication optimal *weak gradecast* protocol. Gradecast (aka graded broadcast) is a relaxed version of broadcast introduced by Feldman and Micali [49] which can be obtained in constant number of rounds. Feldman and Micali [49] provided a gradecast protocol tolerating $t < n/3$ Byzantine faults in the *plain authenticated* model without PKI and digital signatures. Later, Katz and Koo [71] provided a slightly weaker gradecast protocol in the *authenticated* model tolerating $t < n/2$ Byzantine faults using PKI and digital signatures. The gradecast protocol of Katz and Koo [71] incurs $O(\kappa n^3)$ communication even for a single bit input in the absence of threshold signatures. In this work, we present a gradecast protocol with a optimal communication complexity of $O(n\ell + \kappa n^2)$ for ℓ bit input.

Our gradecast (refer Figure 6.4) implements weaker gradecast [71] (Definition 6.3.2) which relaxes gradecast [49] when no honest party outputs a grade of 2 and allows honest parties to output different values with a grade of 1. In particular, when an honest party P_j outputs a value v with a grade of 1, our primitive allows other honest parties to output a different value v' with a grade of 1 when no honest party outputs a value with a grade of 2. This weaker gradecast suffices for our purpose. In Section 6.10, we show a quadratic lower bound on the communication complexity of weak gradecast for completeness.

Deliver. We recall the Deliver function (refer Figure 4.2) used by an honest party to efficiently propagate long messages. This function is adapted from Chapter 4 where linear-sized messages are propagated among all honest parties with $O(\kappa n^2)$ communication cost. The Deliver function enables efficient propagation of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are a keyword mtype, long message m, accumulation value z_e corresponding to message m and epoch e in which Deliver function is invoked. The input keyword mtype corresponds to message *type* containing long message m sent by its sender. In order to facilitate efficient leader equivocation, the input keyword mtype, hash of long message m, accumulation value z_e, and epoch e are signed by the sender of message m. We omit epoch

parameter when the Deliver function is not invoked within an epoch. The Deliver function incurs 2 rounds.

Set $o_i = \perp$ and $g_i = \perp$. Each party P_i performs the following operations:

- **Round 1:** If party P_j is the designated sender, then it multicasts its input value v in the form of $\langle \mathsf{gcast}, v, z \rangle_j$ where z is the accumulation value of v.
- **Round 2:** If party P_i receives $pr := \langle \mathsf{gcast}, v, z \rangle_j$ for the first time, then invoke $\mathsf{Deliver}(\mathsf{gcast}, pr, z)$.
- **Round 4:** If party P_i invoked Deliver in round 2 and no party P_j equivocation has been detected so far, set $o_i = v$ and $g_i = 2$. Let v_i be the first value received. If $v_i = \perp$, set $o_i = \perp$ and $g_i = 0$, else if $o_i = \perp$, set $o_i = v_i$ and $g_i = 1$. Output (o_i, g_i).
- **At any round:** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 6.4: **Weak Gradecast with $O(n\ell + (\kappa + w)n^2)$ communication.**

The gradecast protocol is presented in Figure 6.4. In round 1, the designated sender P_j sends value v by multicasting $\langle \mathsf{gcast}, v, z \rangle_j$ where z is the accumulation value for value v. We note that the size of input value v can be large. To facilitate efficient equivocation checks, the sender P_j signs $\langle \mathsf{gcast}, H(v), z \rangle$ and sends v separately. Whenever an equivocation by the sender is detected, multicasting signed hashes suffices to prove equivocation by the sender. The reduction in communication is obtained via the use of efficient erasure coding schemes [95], cryptographic accumulators [15] and multicast of equivocating hashes (if any). Multicasting of equivocating hashes been explored in several efficient BFT protocols [7, 100, 10].

In round 2, if party P_i receives $\langle \mathsf{gcast}, v, z \rangle_j$, it invokes Deliver to propagate long message v. Note that Deliver function requires 2 rounds. Round 3 accommodates steps of Deliver function invoked in rounds 2. In round 4, each party P_i sets its output value and initial grades based on whether they invoked Deliver in round 2 and received any value.

6.5.1 Security Analysis

Lemma 59. *Suppose party P_j is the designated sender. If an honest party invokes* Deliver *in round r for a value m sent by party P_j and no honest party has detected a party P_j equivocation by round $r + 1$, then all honest parties will receive value m by round $r + 2$.*

Proof. Suppose an honest party P_i invokes Deliver at round r for a value m sent by party P_j. Party P_i must have sent valid code words and witness $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e, e \rangle_i$ computed from value m to every party P_k $\forall k \in [n]$ at round r. The code words and witness arrive at all honest parties by round $r + 1$.

Since no honest party has detected a party P_j equivocation by round $r+1$, it must be that either honest parties will forward their code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e, e \rangle$ when they receive the code words sent by party P_i or they already sent the corresponding code word when they either invoked Deliver for value m or received the code word from some other party. In any case, all honest parties will forward their code word corresponding to value m by round $r+1$. Thus, all honest parties will have received $t+1$ valid code words for a common accumulation value z_e by round $r+2$ sufficient to decode value m. □

Theorem 60. *The protocol in Figure 6.4 is a gradecast protocol satisfying Definition 6.3.2.*

Proof. Suppose party P_j is the designated sender with its input value v.

We first consider the case when an honest party P_i outputs value v with a grade $g_i = 2$. Honest party P_i must have invoked Deliver for value v by round 2 and did not detect a party P_j by round 4. This implies no honest party detected a party P_j equivocation by round 3. By Lemma 59, all honest parties receive value v by round 4. In addition, since party P_i invoked Deliver for value v by round 2, all honest parties receive a code word for value v by round 3. Thus, value v is the first value received by all honest parties. Since $v \neq \perp$, all honest parties will output value v with a grade ≥ 1.

Next, we consider the case when the designated sender is honest. Since, the sender is honest, it sends its input value v to all honest parties such that all honest parties receive value v in round 2. Thus, all honest parties invoke Deliver to propagate value v in round 2. Moreover, the honest sender does not equivocate. Thus, all honest parties output value v with a grade of 2 in round 4.

The case where each honest party outputs a value with a grade $\in \{0, 1, 2\}$ is trivial by design. □

Lemma 61 (Communication Complexity)**.** *Let ℓ be the size of the input, κ be the size of accumulator, and w be the size of witness. The communication complexity of the protocol in Figure 6.4 is $O(n\ell + (\kappa + w)n^2)$.*

Proof. At the start of the protocol, the sender multicasts its value of size ℓ to all party P_j $\forall j \in [n]$ along with κ sized accumulator. This step incurs $O(n\ell + \kappa n)$. Invoking Deliver on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, the overall communication complexity is $O(n\ell + (\kappa + w)n^2)$. □

6.6 Recoverable Set of Shares

In Section 6.4, we presented a secure DKG protocol by assuming broadcast channels. In general, broadcast channels are instantiated using Byzantine Broadcast (BB) or Byzantine agreement (BA) protocols. To the best of our knowledge, all known BB and BA protocols tolerating $t < n/2$ Byzantine faults incur $O(\kappa n^3)$ communication in the absence of threshold signatures [43, 4, 71]. The secure DKG protocol required $2n$ broadcasts. Thus, instantiating broadcast channel using BB or BA protocols for our secure DKG protocol trivially incurs $O(\kappa n^4)$ communication. In this section, we present a slightly weaker sharing protocol by appropriately replacing the broadcast channel with multicast and our weak gradecast. This protocol completes in constant rounds and acts as a building block towards constructing the DKG. We call this protocol *Recoverable Set of Shares*.

In the sharing phase of our secure DKG protocol with broadcast channels (refer Figure 6.2), each honest party outputs a common set `QUAL` consisting of size at least $n-t$ parties such that the secrets shared by parties in set `QUAL` can be reconstructed. In more detail, honest parties have a common decision on which parties correctly shared their secret at the end of the sharing phase. Requiring this agreement was free in the presence of broadcast channels; however, under a point-to-point network, it blows up communication complexity.

Thus, in our protocol, we instead rely on the use of weaker primitive such as gradecast instead of consensus to share secrets. As a result, each honest party P_i may have a different view regarding the acceptance of the shared secret. Thus, each honest party P_i outputs a possibly different set $\mathsf{AcceptList}_i$ of at least $n-t$ parties which they accept to have shared the secret correctly; i.e., party P_i observes the secrets shared by parties in $\mathsf{AcceptList}_i$ can be reconstructed. It is in this regard, we call our protocol *recoverable* set of shares as the secret shared by parties in $\mathsf{AcceptList}_i$ can be reconstructed.

We stress that in recoverable set of shares protocol, honest parties need not agree on a common set and may output a different set of at least $n - t$ parties which they believe have shared the secret properly. To ensure that the final keys for DKG are generated for a common set, parties need to agree on one such set. In the following section, we present a multi-valued validated Byzantine agreement protocol to agree on a common set.

We call an $\mathsf{AcceptList}$ *certified* if it is accompanied by a set of signatures from at least $t + 1$ parties. The set of $t + 1$ signatures on $\mathsf{AcceptList}$ forms the certificate for $\mathsf{AcceptList}$ and denoted as $\mathcal{AC}(\mathsf{AcceptList})$.

Each party P_i performs the following operations:

1. **(Round 1) Distribute.** Each party P_i selects two random polynomials $f_i(y)$, $f'_i(y)$ over $\mathbb{Z}_p$ of degree t:

$$f_i(y) = a_{i0} + a_{i1}y + \cdots + a_{it}y^t, \qquad f'_i(y) = b_{i0} + b_{i1}y + \cdots + b_{it}y^t$$

   Let $s_i = a_{i0} = f_i(0)$. Party P_i generates the commitment $\mathcal{C}_{ik} = g^{f_i(k)}h^{f'_i(k)}\ \forall k \in \{1, \ldots, n\}$. Let $\mathsf{VSS}.\vec{C}_i$ represent $\mathcal{C}_{ik}\ \forall k \in \{1, \ldots n\}$. Party P_i multicasts its proposal $\langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$. Party P_i computes the shares $s_{ij} = f_i(j)$, $s'_{ij} = f'_i(j)$ and sends s_{ij}, s'_{ij} to $P_j\ \forall j \in [n]$.

2. **(Round 2) Blame/Forward.** If party P_i receives commitment $\mathsf{comm}_j := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_j, z_{pj}\rangle_j$ and valid secret share s_{ji}, s'_{ji} (i.e., satisfy Equation (6.1) with $\mathsf{VSS}.\vec{C}_j$), then invoke $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_j, z_{pj}, -)$. If no valid secret shares has been received from party P_j, multicast $\langle \mathsf{blame}, j\rangle_i$ to all parties.

3. **(Round 3) Request open.** Collect all blames received so far. If up to t blame are received for party P_j, forward the blame messages to party P_j.

4. **(Round 4) Open.** Party P_i sends secret shares s_{ik}, s'_{ik} to party P_j, for every blame $\langle \mathsf{blame}, i\rangle_k$ received from party P_j.

5. **(Round 5) Vote.** Upon receiving valid secret shares s_{jk}, s'_{jk} for every $\langle \mathsf{blame}, j\rangle_k$ it forwarded and no party P_j equivocation has been detected, send $\langle \mathsf{vote}, H(\mathsf{comm}_j)\rangle_i$ to party P_j. Forward secret share s_{jk} to party P_k for every $\langle \mathsf{blame}, j\rangle_k$ it received. If no blames for party P_j has been received by round 3 and no party P_j equivocation has been detected, send $\langle \mathsf{vote}, H(\mathsf{comm}_j)\rangle_i$ to party P_j.

6. **(Round 6) Vote cert.** Upon receiving $t+1$ distinct vote messages for comm_i (denoted by $\mathcal{C}(\mathsf{comm}_i)$), invoke weak gradecast (refer Figure 6.4) to propagate $\mathcal{C}(\mathsf{comm}_i)$.

7. **(Round 9) Propose Grade** Let $(o_{j,i}, g_{j,i})$ be the output of weak gradecast with party P_j as the sender. Set $\mathsf{AcceptList}_i[j] = g_{j,i}$. Multicast $\langle \mathsf{accept\text{-}list}, \mathsf{AcceptList}_i\rangle_i$.

8. **(Round 10) Verify and Ack.** Upon receiving $\langle \mathsf{accept\text{-}list}, \mathsf{AcceptList}_j\rangle_j$ from party P_j, if the following conditions hold send $\langle \mathsf{ack}, H(\mathsf{AcceptList}_j)\rangle_i$ to party P_j.

   (a) $|\{h \mid \mathsf{AcceptList}_j[h] = 2\}| \geq n - t$

   (b) If $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_i[h] \geq 1\ \forall h \in [n]$.

9. **(At any round) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 6.5: **Recoverable Set of Shares**

Definition 6.6.1 (Recoverable Set of Shares)**.** *Each party P_i, as a dealer, secret shares a uniformly random input s_i. Each honest party outputs an n element certifed list $\mathsf{AcceptList}_i$ with an entry corresponding to each party as a dealer such that $\mathsf{AcceptList}_i[j] \in \{0, 1, 2\}\ \forall j \in [n]$. A recoverable set of shares protocol tolerating t Byzantine failures satisfies the following properties:*

1. *If dealer P_j is honest, then each honest party P_i outputs $\mathsf{AcceptList}_i[j] = 2$.*
2. *A certified $\mathsf{AcceptList}_i$ must have $|\{h \mid \mathsf{AcceptList}_i[h] = 2\}| \geq n - t$.*
3. *If $\mathsf{AcceptList}_i$ is certified and $\mathsf{AcceptList}_i[j] = 2$, then secret s_j can be recovered from the secret shares s_{ji} received by each honest party P_i.*

Protocol details. At the start of the protocol (refer Figure 6.5), each honest party P_i selects two random t degree polynomials $f_i(y) = \sum_k a_{ik}y^k$ over $\mathbb{Z}_p$ and $f'_i(y) = \sum_k b_{ik}y^k$ over $\mathbb{Z}_p$ such that $f_i(0) = s_i$ and $f'_i(0) = s'_i$. Party P_i generates the commitment $\mathcal{C}_{ik} = g^{f_i(k)}h^{f'_i(k)}$ $\forall k \in \{1, \ldots, n\}$. Let $\mathsf{VSS}.\vec{C}_i$ represent $\mathcal{C}_{ik}$ $\forall k \in \{1, \ldots n.\}$. Party P_i multicasts the commitment in the form of a proposal $\langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$ where z_{pi} is the accumulation value of $\mathsf{VSS}.\vec{C}_i$. In order to facilitate efficient equivocation checks, party P_i signs $\langle \mathsf{propose}, H(\mathsf{VSS}.\vec{C}_i), z_{pi}\rangle$ separately and sends $\mathsf{VSS}.\vec{C}_i$ separately. Party P_i also privately sends secret share s_{ij}, s'_{ij} to party P_j $\forall j \in [n]$.

If a party P_j receives valid secret share s_{ij}, s'_{ij} along with the proposal $\mathsf{comm}_i := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$ by the start of round 2, it invokes $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_i, z_{pi}, -)$ to propagate the commitment $\mathsf{VSS}.\vec{C}_i$; otherwise party P_j multicasts $\langle \mathsf{blame}, i\rangle_j$. Observe that we ignore the epoch e parameter in $\mathsf{Deliver}$ as the current protocol is not executed in an epoch.

Party P_j waits to collect any blame messages sent by other parties. If up to t blame messages are received for P_i, P_j forwards the blame messages to party P_i. Party P_i then privately sends secret shares s_{ik}, s'_{ik} to party P_j, for every blame $\langle \mathsf{blame}, i\rangle_k$ received from party P_j. Upon receiving valid secret shares for all $\langle \mathsf{blame}, i\rangle_k$ it forwarded, party P_j sends a vote $\langle \mathsf{vote}, H(\mathsf{comm}_i)\rangle$ to party P_i and also forwards secret shares s_{ik}, s'_{ik} to party P_k if no party P_i has been detected by round 5. Additionally, if no blame messages are received for P_i by round 3, party P_j sends $\langle \mathsf{vote}, H(\mathsf{comm}_i)\rangle$ to party P_i at round 5.

Party P_i then waits to collect $t+1$ vote messages for $H(\mathsf{comm}_i)$, denoted by $\mathcal{C}(\mathsf{comm}_i)$. A certificate on the comm_i implies that secret s_i shared by party P_i can be reconstructed later. Party P_i then gradecasts $\mathcal{C}(\mathsf{comm}_i)$. Invocation of gradecast on $\mathcal{C}(\mathsf{comm}_i)$ ensures that if the party P_i is honest, all honest parties output a common $\mathcal{C}(\mathsf{comm}_i)$ with a grade of 2 and if an honest party P_k output $\mathcal{C}(\mathsf{comm}_i)$ with a grade of 2, all other honest parties output the certificate with a grade ≥ 1.

Note that all parties (at least all honest parties) are executing the secret sharing phase. Thus, at the end of gradecast step, each honest party outputs at least $n-t$ certificates with a grade of 2 and outputs at most t values with a grade ≤ 2. We call the list of grades for party P_j as $\mathsf{AcceptList}_j$. This list is a set of parties which party P_j observes to have shared their secret properly and each secret can be reconstructed. Party P_j then multicasts its $\mathsf{AcceptList}_j$ to all other parties. Party P_k then checks the validity of $\mathsf{AcceptList}_j$ by checking if (i) $|\{h \mid \mathsf{AcceptList}_j[h] = 2\}| \geq n - t$, and (ii) if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_k[h] \geq 1$ $\forall h \in [n]$. The first check ensures that $\mathsf{AcceptList}_j$ contains at least $n - t$ entries with $\mathsf{AcceptList}_j[h] = 2$. This check trivially satisfies for $\mathsf{AcceptList}$ sent by an honest party as each honest party receives at least $n - t$ certificates with a grade of 2. Later, the DKG protocols use secrets from parties in $\mathsf{AcceptList}_j$ such that $\mathsf{AcceptList}_j[h] = 2$ to

compute the final keys. This is required to ensure security of DKG protocol. The second check ensures that all the secrets corresponding to $\mathsf{AcceptList}_j[h] = 2$ are recoverable; observe that if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_k[h] \geq 1$ due to weak gradecast properties. This implies party P_k has received a $\mathcal{C}(\mathsf{comm}_h)$ from party P_h and $\mathcal{C}(\mathsf{comm}_h)$ implies the secret shared by party P_h can be reconstructed. If the checks pass, party P_k sends $\langle \mathsf{ack}, H(\mathsf{AcceptList}_j)\rangle_k$ to party P_j. A set of $t+1$ ack (ack-cert) messages for $\mathsf{AcceptList}_j$ (denoted by $\mathcal{AC}(\mathsf{AcceptList}_j)$) implies at least one honest party has verified that all the secrets corresponding to $\mathsf{AcceptList}_j[h] = 2$ can be recovered.

The idea of using gradecast to perform secret sharing has been explored before in the works of Feldman and Micali [49, 50] to generate common source of randomness. Compared to their work, our protocols work in authenticated model with $t < n/2$ resilience and invoke a single gradecast per secret sharing. Their protocols work in *unauthenticated* model without PKI with $t < n/4$ [49] and $t < n/3$ [50] resilience and involved multiple invocation of gradecast per secret sharing.

6.6.1 Security Analysis

Lemma 62. *If an honest party sends* vote *for a commitment* comm*, then (i) all honest parties receive* comm*, (ii) all honest parties receive their valid secret shares corresponding to commitment* comm*.*

Proof. Suppose an honest party P_i sends a vote for commitment $\mathsf{comm}_k := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_k, z_{pk}\rangle_k$ at round 5. Party P_i must have received up to t blame messages for party P_k. This implies at least one honest party P_j received valid secret shares $s_{k,j}$, $s'_{k,j}$ and commitment comm_k and invoked $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_k, z_{pk}, -)$ at round 2. Moreover, party P_i did not detect party P_k equivocation by round 5. This implies no honest party detected party P_k equivocation by round 3. By Lemma 59, all honest parties receive the commitment comm_k by round 4. This proves part (i) of the Lemma.

For part (ii), party P_i can send vote message on two occasions: (a) when it does not detect a $\langle \mathsf{blame}, k\rangle$ by round 3 and party k equivocation by round 5, and (b) when party k sent valid secret shares for every $\langle \mathsf{blame}, k\rangle$ message it forwarded and does not detect any party k equivocation by round 5.

In case (a), party P_i did not detect a party k equivocation by round 5 and $\langle \mathsf{blame}, k\rangle$ by round 3. Observe that all honest parties must have received valid secret shares corresponding to the commitment comm_k; otherwise party P_i must have received $\langle \mathsf{blame}, k\rangle$ by round 3 (since honest

parties send $\langle \mathsf{blame}, k \rangle$ if no valid secret shares are received at round 2). Thus, all honest parties receive valid secret shares corresponding to commitment comm_k.

In case (b), party P_i receives valid secret shares from party P_k for every $\langle \mathsf{blame}, k \rangle$ (up to t blame) messages it forwarded and detected no party k equivocation by round 5. Observe that party P_i received $f \leq t$ $\langle \mathsf{blame}, k \rangle$ messages and received valid secret shares for every $\langle \mathsf{blame}, k \rangle$ message it forwarded. This implies at least $n - t - f$ honest parties have received valid shares for commitment comm_k from party P_k; otherwise, party P_i would have received more than f $\langle \mathsf{blame}, k \rangle$ message by round 3. Since, party P_i forwards f received secret shares corresponding to f received $\langle \mathsf{blame}, k \rangle$, all honest parties receive valid secret shares corresponding to commitment comm_k. □

Lemma 63. *If an honest party sends an* ack *for a grade list* $\mathsf{AcceptList}_j$*, then all honest parties have valid secret shares corresponding to* comm_h *for all* h *such that* $\mathsf{AcceptList}_j[h] = 2$.

Proof. Suppose an honest party P_i sends an ack for a grade list $\mathsf{AcceptList}_j$. Then, it must be that if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_i[h] \geq 1$ $\forall h \in [n]$. Party P_i sets $\mathsf{AcceptList}_i[h] \geq 1$ when it receives a vote certificate $\mathcal{C}(\mathsf{comm}_h)$. If there is a vote certificate $\mathcal{C}(\mathsf{comm}_h)$ for value comm_h, then at least one honest party (say party P_k) must have voted for comm_h. By Lemma 62 part (ii), all honest parties have valid secret shares corresponding to commitment comm_h. Thus, all honest parties have valid secret shares corresponding to comm_h for all h such that $\mathsf{AcceptList}_j[h] = 2$. □

Lemma 64 (Liveness)**.** *Each honest party* P_i *will receive an* $\mathsf{ack\text{-}cert}$ *for its grade list* $\mathsf{AcceptList}_i$.

Proof. Consider an honest party P_i. Party P_i will send valid commitment $\mathsf{VSS}.\vec{C}_i$ and secret shares s_{ij}, s'_{ij} to party P_j $\forall j \in [n]$ in round 1. All honest parties will receive their valid secret shares s_{ij}, s'_{ij} and commitment comm_i in round 2. Thus, no honest party will send $\langle \mathsf{blame}, i \rangle$ for party P_i.

Observe that up to t Byzantine parties can always send $\langle \mathsf{blame}, i \rangle$. Honest parties wait until round 3 to collect blame messages for any party. Honest parties forward $\langle \mathsf{blame}, i \rangle$ to party P_i which party P_i receives by round 4. Party P_i forwards valid secret shares to party P_j for every $\langle \mathsf{blame}, i \rangle$ message it received from party P_j which party P_j receives by round 5. Thus, party P_j will send vote for party P_i which party P_i receives by round 6. This implies party P_i collects $t + 1$ distinct vote messages by round 6.

Party P_i invokes weak gradecast to propagate $\mathcal{C}(comm_i)$ which completes by round 9. Due to the properties of weak gradecast, for an honest party P_i, all honest parties set $\mathsf{AcceptList}[i]$ to 2. Thus,

for any honest party P_j, all honest parties set $\mathsf{AcceptList}[j]$ to 2. This implies all honest parties will have $|\{h \mid \mathsf{AcceptList}_j[h] = 2\}| \geq n - t$.

Next, we consider the case when an honest party sets $\mathsf{AcceptList}_i[l] = 2$ for a Byzantine party P_l and receive $\mathcal{C}(\mathsf{comm}_l)$. Due to the properties of weak gradecast, all honest parties receive $\mathcal{C}(\mathsf{comm}_l)$ and set $\mathsf{AcceptList}[l] \geq 1$. Thus, for every $\mathsf{AcceptList}_i[h] = 2$ then $\mathsf{AcceptList}[h] \geq 1$ for all honest parties.

Party P_i multicasts its $\mathsf{AcceptList}_i$ in round 9. Since, $\mathsf{AcceptList}_i$ satisfies both the conditions $|\{h \mid \mathsf{AcceptList}_i[h] = 2\}| \geq n - t$ and $\mathsf{AcceptList}_i[h] = 2$ then $\mathsf{AcceptList}[h] \geq 1$, all honest parties will send ack for $\mathsf{AcceptList}_i$ proposed by party P_i and party P_i will receive $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ the end of round 10. □

Theorem 65. *The protocol in Figure 6.5 is a recoverable set of shares protocol satisfying Definition 6.6.1.*

Proof. Straight forward from Lemma 62, Lemma 63 and Lemma 64 □

Lemma 66 (Communication Complexity)**.** *Let ℓ be the size of commitment comm, κ be the size of secret share and accumulator, and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. At the start of the protocol, each party P_i multicasts comm_i of size ℓ to all party P_j $\forall j \in [n]$ and sends secret share $s_{i,j}$ to party P_j $\forall j \in [n]$. This step incurs $O(n^2\ell + \kappa n^3)$. In the Forward step, parties invoke $\mathsf{Deliver}$ for the first comm_j from party P_j for $j \in [n]$. Invoking $\mathsf{Deliver}$ on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, invoking $\mathsf{Deliver}$ on n commitments incurs $O(n^2\ell + (\kappa + w)n^3)$.

In the Blame step, honest parties may blame up to t Byzantine parties if they do not receive valid secret shares. Multicast of t blame from each party incurs $O(\kappa t n^2)$ communication. In addition, t Byzantine parties always can blame honest parties. Honest parties forward up to t $\langle \mathsf{blame}, j \rangle$ messages to party P_j. This incurs $O(\kappa t n^2)$ communication.

In the Private open step each party can send up to t secret shares to all other parties. This incurs $O(\kappa t n^2)$ for all parties. In the Vote cert step, each party multicasts $O(n)$-sized $\mathsf{vote\text{-}cert}$ to all other parties which incurs $O(\kappa n^3)$ in communication. Invoking $\mathsf{Deliver}$ on an $O(n)$-sized certificate incurs $O(n^2 + (\kappa + w)n^2)$. For n certificate, this step incurs $O(n^3 + (\kappa + w)n^3)$.

In the Propose grade step, each party multicast their grade list of size $O(n)$. Multicast of $O(n)$-sized grade list by n parties incurs $O(n^3)$ communication. Thus, the total communication complexity is $O(n^2\ell + (\kappa + w)n^3)$ bits. □

6.7 Oblivious Leader Election

In this section, we construct an oblivious leader election (OLE) (aka, common coin) protocol that outputs a common honest leader with some constant probability called the *fairness*. In the absence of an existing threshold (DKG) setup, the OLE protocol was designed via n^2 parallel invocations of weaker VSS primitives such as graded VSS [49] or moderated VSS [71] which trivially incurs $\Omega(n^4)$ communication. A recent work [5] designs an OLE protocol tolerating $t < n/3$ Byzantine faults using Aggregatable PVSS [64] for the asynchronous model which incurs $O(\kappa n^3)$ communication. However, Aggregatable PVSS requires additional cryptographic assumptions which is not desirable. In this work, we build an OLE protocol using n parallel invocations of weaker VSS primitives and a non-interactive threshold signature scheme [29]. Note that our OLE protocol does not require a prior threshold (DKG) setup phase despite making use of threshold signatures. Our OLE protocol works in the random oracle model and requires CDH assumption. The resulting protocol incurs a communication complexity of $O(\kappa n^3)$ and constant rounds.

Construction. The starting point of our construction is the threshold coin-tossing scheme of Cachin et al. [29] which makes use of non-interactive threshold signature scheme. The threshold signature scheme requires a prior threshold setup which is essentially a DKG. The threshold setup establishes a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ of verification keys. After the threshold setup phase, each party signs a common message (e.g., an epoch number) with its threshold secret key to obtain a threshold share. A combination of any $t+1$ valid threshold shares is then used to obtain a unique and random threshold signature σ. A random oracle $H'' : \mathbb{G} \rightarrow \{0,1\}$ is then used to generate an unbiased and unpredictable random bit from the threshold signature σ.

Note that the threshold signature scheme requires a prior threshold setup to establish a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ of verification keys. We fulfill this requirement by using the output of recoverable set of shares protocol (Section 6.6) to establish a local threshold setup corresponding to each party. In recoverable set of shares protocol, each party P_i outputs an $\mathsf{AcceptList}_i$ along with $\mathcal{AC}(\mathsf{AcceptList}_i)$. An $\mathsf{AcceptList}_i$ consists of at least $n - t$ entries with grades of 2 and all honest parties will contain secret shares shared by parties in $\mathsf{AcceptList}_i$ whose grades are 2. Thus, each party P_j uses secret shares shared by parties in an $\mathsf{AcceptList}_i$ with

Each party P_i performs the following operations:

1. Each party P_i invokes recoverable set of shares protocol (refer Figure 6.5). Each party P_i outputs ($\mathsf{AcceptList2}_i$ $\mathcal{AC}(\mathsf{AcceptList2}_i)$).
2. Each party P_i invokes weak gradecast to propagate ($\mathsf{AcceptList2}_i, \mathcal{AC}(\mathsf{AcceptList2}_i)$).
3. Let ($o_{j,i}$, $\mathsf{grade}_i[j]$) be the output with party P_j as dealer. Let $o_{j,i}$ contains $\mathsf{AcceptList2}_j$. If $\mathsf{grade}_i[j] \geq 1$, set $\mathsf{local\_dkg}_i[j] = \mathsf{AcceptList2}_j$, $\mathsf{local\_dkg\_grade}_i[j] = \mathsf{grade}_i[j]$.
   - Set $\mathsf{sk}_{j,i} = \sum_{m \in \mathsf{AcceptList2}_j | \mathsf{AcceptList2}_j[m]=2} s_{mi}$, $\mathsf{vk}_{j,i} = g^{\mathsf{sk}_{j,i}}$, and $\mathsf{sk}'_{j,i} = \sum_{m \in \mathsf{AcceptList2}_j | \mathsf{AcceptList2}_j[m]=2} s'_{ji}$.
   - Compute $\mathcal{C}_{\langle g \rangle}(\mathsf{sk}_{j,i})$, $\mathcal{C}_{\langle g,h \rangle}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i})$) and $\pi_{\equiv \mathsf{Com}\, j,i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i}, g, h, \mathcal{C}_{\langle g \rangle}(\mathsf{sk}_{j,i}), \mathcal{C}_{\langle g,h \rangle}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i}))$. Multicast ($\mathsf{vk}_{j,i}, \pi_{\equiv \mathsf{Com}\, j,i}$) to all parties.

Figure 6.6: Threshold setup protocol

grades of 2 to compute its secret key $\mathsf{sk}_{i,j}$ and verification key $\mathsf{vk}_{i,j} = g^{\mathsf{sk}_{i,j}}$ to establish local DKG setup $\mathsf{local\_dkg}[i]$ corresponding to party P_i. Note that this establishes a separate threshold setup for each party. With local DKG setup $\mathsf{local\_dkg}[i]$ as the threshold setup for party P_i, parties collectively generate a unique and random threshold signature σ_i. Parties then use a random oracle $H' : \mathbb{G} \rightarrow \{0,1\}^\kappa$ to generate a random coin value assigned to party P_i. A party with highest (or lowest) coin value is selected to be the leader.

Looking ahead, the final DKG is also computed from one of the valid $\mathsf{AcceptList}$ output from the recoverable set of shares. Making use of the secret shares in an $\mathsf{AcceptList}$ output from the recoverable set of shares during this local DKG setup phase will leak the final public key before the final DKG is decided. Note that the final public key can be computed from $t+1$ verification keys. This allows the adversary ability to force the final DKG to have certain final public key. To circumvent this issue, we execute two separate instances of recoverable set of shares in parallel; one instance to setup local DKG instances and the other to setup the final DKG instance. To remove this ambiguity, we call the accept list output from the recoverable set of shares executed for local DKG as $\mathsf{AcceptList2}$ i.e. each party P_i outputs an $\mathsf{AcceptList2}_i$ along with $\mathcal{AC}(\mathsf{AcceptList2}_i)$.

Protocol details. The setup phase of the protocol is presented in Figure 6.6. Each party P_i invokes recoverable set of shares protocol and outputs $\mathsf{AcceptList2}_i$ (along with $\mathcal{AC}(\mathsf{AcceptList2}_i)$). Each party P_i then invokes weak gradecast to propagate ($\mathsf{AcceptList2}_i$, $\mathcal{AC}(\mathsf{AcceptList2}_i)$). At the end of the setup phase, each party P_i sets up the local DKG instance for each party P_j (i.e., $\mathsf{local\_dkg}_i[j]$) as $\mathsf{AcceptList2}_j$ if $\mathsf{local\_dkg\_grade}_i[j] \geq 1$. If $\mathsf{local\_dkg\_grade}_i[j] = 2$, due to weak gradecast properties, all honest parties have a common local DKG instance for party P_j (i.e., $\mathsf{local\_dkg}[j]$). In addition, for an honest party P_j, all honest parties will have a common local DKG instance $\mathsf{local\_dkg}[j]$. Each party P_i also computes required secret keys $\mathsf{sk}_{j,i}$, verification keys $\mathsf{vk}_{j,i}$ for local DKG instance $\mathsf{local\_dkg}_i[j]$ computed from $\mathsf{local\_dkg}_i[j]$ as shown in Figure 6.6.

Let sid be the input of party P_i.
Set $\mathcal{X}_i \leftarrow \emptyset$. Each party P_i performs following operations:

1. Perform $\sigma_{j,i} = \mathsf{Sign}_{\mathsf{TS}}(\mathsf{sk}_{j,i}, (j, \mathsf{sid}))$ and multicast $\sigma_{j,i}$ if $\mathsf{local\_dkg\_grade}_i[j] \geq 1 \ \forall j \in [n]$.
2. Upon receiving a set S of $t+1$ valid signature shares for party P_j, compute $\sigma_j = \mathsf{Combine}_{\mathsf{TS}}(\mathsf{pk}, \mathsf{sid}, S)$ and $\mathcal{X}_i[j] \leftarrow H'(\sigma_j)$.
3. Perform $\ell \leftarrow argmax_h\{\mathcal{X}_i[h] | \mathsf{local\_dkg\_grade}_i[h] = 2\}$. Output P_ℓ.

Figure 6.7: Oblivious Leader Election

The OLE protocol is presented in Figure 6.7. The input to the protocol is a sequence id sid. Once the local DKG instances are setup, each party P_i uses its secret key $\mathsf{sk}_{j,i}$ to sign a common message i.e., (j, sid) (for party P_j) if $\mathsf{local\_dkg\_grade}_i[j] \geq 1$ to obtain a threshold share $\sigma_{j,i}$. A set of $t+1$ valid signature shares corresponding to $\mathsf{local\_dkg}[j]$ is combined to form a single threshold signature σ_j and a hash $H'(\sigma_j)$ generates coin value for party P_j. We note that two or more parties could output the same grade list (i.e, $\mathsf{AcceptList2}$) in the recoverable set of shares protocol; hence their local DKG might be same. However, parties sign a distinct message e.g. (j, sid) for party P_j. Such generated threshold signatures are unique and random regardless of their local DKG instance being common; hence the coin value is also random. Honest parties consider coin values for party P_j only if $\mathsf{local\_dkg\_grade}_i[j] = 2$. Note that if $\mathsf{local\_dkg\_grade}_i[j] = 2$, a threshold signature σ_j will exist for party P_j. This is because all honest parties will have $\mathsf{local\_dkg\_grade}[j] \geq 1$ and a common $\mathsf{local\_dkg}[j]$ due to weak gradecast properties and each honest party P_i will send their signature share $\sigma_{j,i}$. A coin value is then computed as $H'(\sigma_j)$. The party P_ℓ with highest coin value is elected as leader.

Round complexity and communication complexity. The threshold setup phase has a latency of 15 rounds to invoke recoverable set of shares, n parallel instances of weak-gradecast and distribute verification keys. The OLE protocol requires only 1 round to generate threshold signatures. The threshold setup phase invokes recoverable set shares, n parallel weak-gradecasts with an input of size $O(\kappa n)$ and sharing verification keys. This incurs $O(\kappa n^3)$ communication. The threshold signature generation incurs $O(\kappa n^3)$ communication.

6.7.1 Security Analysis

Our coin generation protocol is similar to the threshold coin-tossing scheme of [29]. In Cachin et al. [29], the coin value is a single bit computed from the threshold signature using $H'' : \mathbb{G} \rightarrow \{0, 1\}$. In our scheme, the coin value is a κ bit string computed from the threshold signature using H' :

$\mathbb{G} \rightarrow \{0,1\}^{\kappa}$. We rely on the following Lemma of [29].

Lemma 67 ([29])**.** *In the random oracle model, the coin-tossing scheme of Cachin et al. [29] is secure i.e., satisfies robustness and unpredictability under CDH assumption.*

We note that in the proof of the above Lemma, Cachin et al. [29] show the threshold signature generation is robust and unpredictable. This suffices to show that our coin-tossing scheme is also secure.

Theorem 68. *In the random oracle model and under CDH assumption, the protocol in Figure 6.7 is an oblivious leader election protocol with fairness at least* $\frac{1}{2}$*.*

Proof. We first show termination i.e., honest party P_i will obtain a threshold signature σ_j (and coin value for party P_j) if $\mathsf{local\_dkg\_grade}_i[j] = 2$. This is because all honest parties will have $\mathsf{local\_dkg\_grade}[j] \geq 1$ and a common $\mathsf{local\_dkg}[j]$ due to weak gradecast properties. Thus, each honest party P_k will send their signature share $\sigma_{j,k}$ i.e., a set of $t+1$ valid signature shares will be available sufficient to obtain threshold signature σ_j (and coin value $H'(\sigma_j)$).

By Lemma 67 the threshold signature generation protocol satisfies robustness and unpredictability. Thus, the coin value generated from threshold signature is robust and unpredictable.

Observe that each party P_i signs a distinct message (i.e, (j, sid)) for each part P_j. Thus, the threshold signature σ_j for each party P_j is unique and random even if two or more parties have the same local DKG instance; hence each party P_j will be assigned random coin value ($H'(\sigma_j)$). Since, the coin value assigned to a party is random, the coin value assigned to an honest party will be a global maximum with probability at least $\frac{n-t}{n}$. The probability that coin values of any two parties can be maximum is bounded by $\frac{1}{2^{\kappa}}$. Thus, all honest parties select the coin value corresponding to a common honest leader with probability $\frac{n-t}{n} - \frac{1}{2^{\kappa}} \geq \frac{1}{2}$ when $\kappa = 2 \log n$. □

6.8 Multi-Valued Validated Byzantine Agreement

In Section 6.6, we presented a recoverable set of shares protocol where each honest party P_i outputs a (possibly different) set $\mathsf{AcceptList}_i$ along with $\mathcal{AC}(\mathsf{AcceptList}_i)$–both of which are linear sized. For DKG, all honest parties need to agree on a common set of parties whose secret shares are used to compute final secret keys and a public key. Thus, we need a consensus primitive that takes a different $O(n)$-sized input from each party and outputs a common set which is valid. Here, a valid

set is accompanied by its ack certificate and can potentially also be the input of a Byzantine party. Such a consensus primitive is called a *multi-valued validated Byzantine agreement*.

Multi-valued validated Byzantine agreement (MVBA) was introduced by Cachin et al. [28] to allow honest parties to agree on any externally valid value. Recent works [8, 79] have proposed MVBA protocols for the asynchronous communication model tolerating $t < n/3$ Byzantine faults. To the best of our knowledge, no MVBA protocol have been proposed in the synchronous communication model for $t < n/2$ case. In this chapter, we present a synchronous MVBA protocol tolerating $t < n/2$ Byzantine faults with $O(n^2\ell + \kappa n^3)$ communication for inputs of size ℓ bits and expected constant rounds.

We extend the Binary Byzantine agreement (BBA) protocol of Katz and Koo [71] to MVBA for large ($\ell = \Theta(n)$) input. The BBA protocol of Katz and Koo [71] tolerates $t < n/2$ Byzantine faults and terminates in expected 4 epochs. Their protocol involves invoking n parallel gradecasts; with each gradecast propagating small sized input. As mentioned before, their gradecast protocol incurs $O(\kappa n^3)$ communication for a single bit input; thus, their protocol trivially incurs $O(\kappa n^4)$ communication. We replace their gradecast protocol with our communication optimal gradecast protocol from Section 6.5. Our gradecast protocol incurs only $O(\kappa n^2)$ communication while propagating $O(n)$-sized input. Using our gradecast protocol allows BBA protocol of Katz and Koo [70] to handle large input while simultaneously reducing the communication to $O(\kappa n^3)$.

To circumvent the linear round lower bound for a deterministic BA protocol [43], BA protocols use a common source of randomness called *common coin* to achieve agreement in constant expected rounds. The common coin is *weak* if all honest parties obtain a common honest leader with some constant probability (and with the remaining probability either the common leader is Byzantine or honest parties may disagree on the leader). In Katz and Koo BBA, the weak common coin was obtained by invoking n^2 moderated VSS instances which incurs $\Omega(\kappa n^4)$ communication and blows up the communication complexity. In this work, we replace their weak common coin protocol with our communication efficient leader election protocol from Section 6.7 which outputs a common honest leader with probability at least $\frac{1}{2}$. Our OLE protocol incurs $O(\kappa n^3)$ communication and a single round after an initial setup phase (refer to Figure 6.6) which incurs 15 rounds.

Our MVBA protocol in presented in Figure 6.8. The underlying consensus mechanism is identical to the BBA protocol of Katz and Koo [71]. In round 1, each party P_i invokes weak gradecast protocol to propagate its input v_i. Our weak gradecast protocol incurs 4 rounds. Rounds 2 and 3 accommodates the steps of the weak gradecast protocol. Again in round 4, each party P_i invokes weak gradecast protocol to propagate its updated input v_i. Rounds 5 and 6 accommodates the

Let v_i be party P_i's input and e be the current epoch. Each party P_i sets $\mathsf{lock}_i \leftarrow \perp$. Each party P_i performs following operations.

1. **(Round 1) Propose.** Each party P_i invokes weak gradecast to propagate v_i.
2. **(Round 4) Update.** Let $(v_{j,i}, \mathsf{grade}_i[j])$ be the output with party P_j as the dealer. Let $\mathcal{S}_i^v := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] = 2\}$ and $\tilde{\mathcal{S}}_i^v := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] \geq 1\}$. If $\mathsf{lock}_i = \perp$, then:
   (a) If $|\tilde{\mathcal{S}}_i^v| > t$, update $v_i \leftarrow v$.
   (b) If $|\mathcal{S}_i^v| > t$, set $\mathsf{lock}_i \leftarrow 1$.

   Invoke weak gradecast (refer Figure 6.4) to propagate v_i.
3. **(Round 7) Update2.** Again, let $(v_{j,i}, \mathsf{grade}_i[j])$ be the output with party P_j as the dealer. Define $\mathcal{S}_i^v$ and $\tilde{\mathcal{S}}_i^v$ as above. If $\mathsf{lock}_i = \perp$ and $|\tilde{\mathcal{S}}_i^v| > t$, set $v_i \leftarrow v$. Multicast v_i.
4. **(Round 8) Leader election.** Invoke OLE protocol with input e.
5. **(Round 9) Terminate/Advance Epoch.** Let P_ℓ be the output of leader election protocol.
   (a) If $\mathsf{lock}_i = 0$, output v_i and terminate.
   (b) If $\mathsf{lock}_i = 1$, set $\mathsf{lock}_i = 0$. If $\mathsf{lock}_i = \perp$ and $|\mathcal{S}_i^v| \leq t$, $v_{\ell,i} \neq \perp$ and $\mathsf{ex\text{-}validation}(v_{\ell,i}) = \mathsf{true}$, update $v_i \leftarrow v_{\ell,i}$. Advance to epoch $e + 1$.
6. **(At any round) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 6.8: **MVBA** with $O(n^2\ell + \kappa n^3)$ communication and expected 4 epochs.

steps of the weak gradecast protocol. In round 8, parties invoke the OLE protocol to elect a leader.

Round complexity. By Theorem 68, a common honest leader is selected with probability at least $\frac{1}{2}$ and all honest parties terminate in the next 2 epochs. Thus, the expected number of epochs required is 4 epochs.

6.8.1 Security Analysis

Lemma 69. *If an honest party sets* lock *to* 1 *with a value* v *in epoch* e*, then all honest parties adopt value* v *in epoch* e*.*

Proof. Suppose an honest party P_i sets lock_i to 1 in epoch e. Party P_i must have received value v from a set Q of at least $t + 1$ parties such that $|\mathcal{S}_i^v| > t$. By the properties of weak gradecast, all other honest parties receive value v corresponding to parties in Q with a grade ≥ 1 (i.e., all other honest parties have $\mathsf{grade}[j] \geq 1\ \forall j \in Q$) and $|\tilde{\mathcal{S}}^v| > t$ for all other honest parties and all honest parties adopt value v in the Update step.

Once all honest parties adopt value v in the Update step, they invoke weak-gradecast to propagate

value v at the end of the Update step. Since, honest parties do not equivocate and send value v in a timely manner, all honest parties output value v such that $\mathsf{grade}[j]$ to 2. Thus, $|\tilde{\mathcal{S}}_i^v| > t$ and $|\mathcal{S}_i^v| > t$ in the Update2 step. Since, $|\mathcal{S}_i^v| > t$, no honest party will adopt value v_ℓ selected from the proposal election protocol. Thus, all honest parties adopt value v in epoch e. □

Lemma 70. *If all honest parties start an epoch e with same input v, then all honest parties decide value v and terminate by the end of epoch $e + 1$.*

Proof. Suppose all honest parties start an epoch e with the same input v. All honest parties invoke weak-gradecast with value v in the Propose step. By the properties of weak gradecast, for an honest dealer, all honest parties output a grade of 2. Thus, all honest parties will set $\mathsf{grade}[j] = 2$ for all other honest parties. Thus, for value v, all honest parties have $|\mathcal{S}_i^v| > t$ and $|\tilde{\mathcal{S}}_i^v| > t$ If $\mathsf{lock} = \perp$, honest parties set lock to 1.

Similarly, all honest parties invoke weak-gradecast with value v in the Update2 step. By similar argument, all honest parties will set $\mathsf{grade}[j] = 2$ for all other honest parties i.e., $|\mathcal{S}_i^v| > t$ and $|\tilde{\mathcal{S}}_i^v| > t$ for all honest parties at the of Update 2 step. Moreover, no honest party will adopt the value output from the proposal election protocol.

Honest parties with $\mathsf{lock} = 0$, output v and terminate in epoch e. All the remaining honest parties with $\mathsf{lock} = 1$, set $\mathsf{lock} = 0$ and advances to epoch $e + 1$. In the next epoch, all the remaining honest parties have $\mathsf{lock} = 1$ and will not update its value and stick to value v. At the end of epoch $e + 1$, they set their $\mathsf{lock} = 0$, output value v and terminate. Thus, all honest parties output v and terminate by the end of epoch $e + 1$. □

Theorem 71. *The protocol in Figure 6.8 solves MVBA.*

Proof. We first consider external validity i.e., if an honest party decides a value v, then $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$. Observe that an honest party P_i decides a value v only when its sets $\mathsf{lock}_i = \mathsf{true}$. An honest party sets $\mathsf{lock}_i = \mathsf{true}$ only when it observes $|\mathcal{S}_i^v| > t$. Thus, at least one honest party P_j must have sent value v in Propose step. Honest party P_j sends value v either when its input at the start of the protocol execution is v in which case $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$, or when its updates its value v_j to v at the end of an epoch. In the latter case, party P_j checks if $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$.

Next, we consider agreement. Consider an epoch e and let P_ℓ be the common leader in epoch e elected via OLE protocol. There are two cases to consider.

Case I. $\mathsf{lock}_i = 1$ for at least one honest party P_i with a value v in epoch e. By Lemma 69, all honest party adopt value v in epoch e and enter epoch $e+1$ with same value v. By Lemma 70, all honest parties output value v and terminate by epoch $e+2$.

Case II. $\mathsf{lock}_i = \perp$ for all honest parties in epoch e. If leader P_ℓ is honest, leader P_ℓ sends the same value v_ℓ to all parties. If $|\mathcal{S}_i^v| \leq t$ for all honest parties, then all honest parties adopt the value v_ℓ in epoch e. By Lemma 70, all honest parties output value v_ℓ and terminate in epoch $e+2$.

If $|\mathcal{S}_i^v| > t$ for at least one honest party P_i in the Update2 step, by the properties of weak-gradecast, $|\tilde{\mathcal{S}}^v| > t$ for all honest parties. Thus, all honest parties including leader P_ℓ adopt value v in the Update2 step. If the leader P_ℓ is honest, it sends the same value v to all parties. Honest parties with $|\mathcal{S}_i^v| \leq t$ adopt value v_ℓ which is the same value adopted by party P_i with $|\mathcal{S}_i^v| > t$. Thus, all honest parties have value v at the end of epoch e. By Lemma 70, all honest parties output value v and terminate by epoch $e+2$. □

Lemma 72 (Communication Complexity)**.** *Let ℓ be the size of input v for each party, κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. At the start of the protocol, each party P_i invokes weak gradecast with $O(\ell)$-sized proposal. By Lemma 61, this step incurs $O(n^2\ell + (\kappa + w)n^3)$. Similarly, in the Update2 step, each party invokes weak gradecast with $O(\ell)$-sized proposal. By Lemma 61, this step also incurs $O(n^2\ell + (\kappa + w)n^3)$. The proposal election protocol has a communication complexity of $O(\kappa n^3)$. Thus, the total communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch. □

6.9 Distributed Key Generation

Finally, we present two communication efficient DKG protocols with $O(\kappa n^3)$ communication. The first protocol incurs expected $O(\kappa n^3)$ communication and terminates in expected constant rounds while the second protocol incurs $O(\kappa n^3)$ communication in the worst case and terminates in $t+1$ epochs. The DKG protocols in this section differs from the secure DKG protocol of Section 6.4 in the following ways. First, we replace the broadcast channel with weaker consensus primitives and use a single invocation of consensus instance. Second, in the secure DKG protocol, the final public key and secret keys are computed from the secret shares of all honest parties. In particular, all honest parties belong to set `QUAL` and the public key and secret keys are computed from parties in

1. **Deal/Setup.** Each party P_i invokes recoverable set of shares protocol (refer Figure 6.5). Each party P_i outputs a set $\mathsf{AcceptList}_i$ with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ (i.e., $\mathcal{AC}(\mathsf{AcceptList}_i)$). Each party P_i also invokes threshold setup phase (refer Figure 6.6) in parallel.
2. **MVBA.** Each party P_i invokes MVBA (Figure 6.8) with input $(\mathsf{AcceptList}_i, \mathcal{AC}(\mathsf{AcceptList}_i))$. Let $\mathsf{AcceptList}_k$ be the output of all honest parties.
3. **Generating keys.** Let $x_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s_{ji}$ and $x'_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s'_{ji}$ be the sum of secret shares in $\mathsf{AcceptList}_k$. Compute $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$ and $\pi_{\equiv \mathsf{Com}\, i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g \rangle}(x_i), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$.
   - Multicast $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, i})$ to all parties.
   - Verify the received $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, j})$ as shown in Equation (6.3).
   - Upon receiving $t+1$ valid $\mathcal{C}_{\langle g \rangle}(x_i)$, interpolate them to obtain $y = g^x$. Set y as the public key and x_i as the private key.

Figure 6.9: DKG with expected $O(\kappa n^3)$ communication and expected $O(1)$ rounds

QUAL. In contrast, the DKG protocols in this section compute the final public key and secret keys from a common set of at least $n - t$ parties where at least $n - 2t$ parties are honest (i.e., at least one honest party when $n = 2t + 1$). This suffices to ensure construction of a secure DKG protocol.

6.9.1 DKG with $O(\kappa n^3)$ communication and expected $O(1)$ rounds

The DKG protocol uses recoverable set of shares protocol (refer Figure 6.5) to perform secret sharing. The threshold setup protocol (refer Figure 6.6) is also executed at the start of the execution. At the end of the recoverable set of shares, each honest party P_i outputs a (possibly different) set of at least $n - t$ parties ($\mathsf{AcceptList}_i$) which they observe to have correctly shared their secret along with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ ($\mathcal{AC}(\mathsf{AcceptList}_i)$). The $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ serves an external validity function to the MVBA protocol i.e., if there is an $\mathcal{AC}(\mathsf{AcceptList}_i)$ for $\mathsf{AcceptList}_i$, then $\mathsf{ex\text{-}validation}(\mathsf{AcceptList}_i) = \mathsf{true}$. Note that both $\mathsf{AcceptList}_i$ and $\mathcal{AC}(\mathsf{AcceptList}_i)$ are linear sized. Each honest party P_i then invokes MVBA protocol with $(\mathsf{AcceptList}_i, \mathcal{AC}(\mathsf{AcceptList}_i))$ as input. At the end of MVBA protocol, each honest party outputs a common set $\mathsf{AcceptList}_k$. The final secret key and public key is then computed using secret shares shared by parties h such that $\mathsf{AcceptList}_k[h] = 2$ using the reconstruction protocol in Figure 6.2.

Latency and communication complexity. The recoverable set of shares protocol has a round complexity of 10 rounds and $O((\kappa + w)n^3)$ communication. The threshold setup protocol incurs a communication of $O((\kappa + w)n^3)$ and 15 rounds; but is executed in parallel and completes before the OLE protocol is invoked in the MVBA protocol. Thus, it does not increase overall round complexity of the protocol. The MVBA protocol incurs expected 4 epochs (with each epoch being

9 rounds) and $O((\kappa + w)n^3)$ communication where the size of input is $O(\kappa n)$. The reconstruction phase requires $O(\kappa n^2)$ communication and a single round. Thus, the protocol incurs $O((\kappa + w)n^3)$ communication and expected 47 rounds.

6.9.2 DKG with worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds

While the above protocol terminates in expected 4 epochs in the best case, it has probabilistic termination and may require a linear number of epochs in the worst case with a communication of $O(\kappa n^4)$. As an alternate solution, we present a DKG protocol with guaranteed termination in $t+1$ epochs with $O(\kappa n^3)$ communication in the worst case. The protocol is presented in Figure 6.10. In the protocol, honest parties execute the recoverable set of shares protocol and each honest party P_i outputs a (possibly different) set of at least $n - t$ parties ($\mathsf{AcceptList}_i$) which they observe to have correctly shared their secret along with an ack-cert for $\mathsf{AcceptList}_i$ ($\mathcal{AC}(\mathsf{AcceptList}_i)$). The tuple ($\mathsf{AcceptList}_i$, $\mathcal{AC}(\mathsf{AcceptList}_i)$) is input into a leader-based Byzantine fault tolerant state machine replication (BFT SMR) protocol from Chapter 4 to agree on a common set. We present a brief overview of the BFT SMR.

1. **Deal.** Each party P_i invokes recoverable set of shares protocol (refer Figure 6.5). Each party P_i output a set $\mathsf{AcceptList}_i$ with an ack-cert for $\mathsf{AcceptList}_i$.
2. **BFT SMR.** Each party P_i participates in BFT SMR [21] with input $\mathsf{AcceptList}_i$ and $\mathcal{AC}(\mathsf{AcceptList}_i)$. The BFT SMR protocol is executed in round-robin manner with first $t+1$ leaders. Let $\mathsf{AcceptList}_k$ be the first committed value of all honest parties.
3. **Generating keys.** Let $x_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s_{ji}$ and $x'_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s'_{ji}$ be the sum of secret shares in $\mathsf{AcceptList}_k$. Compute $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$ and $\pi_{\equiv \mathsf{Com}\, i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g \rangle}(x_i), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$.
   - Multicast $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, i})$ to all parties.
   - Verify the received $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, j})$ as shown in Equation (6.3).
   - Upon receiving $t+1$ valid $\mathcal{C}_{\langle g \rangle}(x_i)$, interpolate them to obtain $y = g^x$. Set y as the public key and x_i as the private key.

Figure 6.10: DKG with worst-case $O(\kappa n^3)$ communication and $t+1$ epochs

BFT SMR from Chapter 4. The BFT SMR protocol is a communication efficient rotating-leader SMR protocol with $O(\kappa n^2)$ communication per epoch even for $O(n)$-sized input. The BFT SMR protocol has optimal resilience i.e., tolerates $t < n/2$ Byzantine faults. The leaders are rotated in each epoch, where an epoch is a duration of 7 rounds. When the leader of an epoch is honest, all honest parties commit the proposed value in the same epoch, whereas, when the leader of the epoch is Byzantine, some honest parties may require linear number of epochs to commit the proposed

value. The BFT SMR utilizes the "block-chaining" paradigm i.e., each proposal is represented in the form of a block which explicitly extends a block B proposed earlier by including hash of previous block B. In this paradigm, when a block B is committed, all its ancestors are also committed.

In this DKG protocol, we execute the BFT SMR protocol for $t+1$ epochs. In each epoch, the epoch leader is expected to propose its (AcceptList, $\mathcal{AC}$(AcceptList)). If the epoch leader is honest, all honest parties commit the proposed set in the same epoch; otherwise honest parties may require linear number of epochs when the leader is Byzantine to commit the proposed value or commit no value at all if the Byzantine leader does not propose. Since the BFT SMR protocol is executed for $t+1$ epochs, there will be at least one honest leader; thus all honest parties commit at least one set. Honest parties output the first committed set and perform reconstruction using this set to generate the final secret key and public key.

Latency and communication complexity. The recoverable set of shares protocol incurs a latency of 10 rounds and $O(\kappa n^3)$ communication. The BFT SMR protocol incurs $O(\kappa n^2)$ communication per epoch; $O(\kappa n^3)$ communication for $t+1$ epochs. The length of each epoch is 7 rounds. The reconstruction phase requires $O(\kappa n^2)$ communication and a single round . Thus, the protocol incurs $O(\kappa n^3)$ communication in the worst-case and $11 + 7 * (t+1)$ rounds.

6.10 A Lower Bound on the Communication Complexity of Weak Gradecast

In this section, we show a quadratic communication lower bound for the weak gradecast protocol. The proof of this lower bound is a trivial extension of the communication lower bound for Byzantine broadcast by Dolev and Reischuk [41].

Lemma 73. *There does not exist a protocol for weak gradecast tolerating t Byzantine parties with a communication complexity of at most $t^2/4$ messages.*

Proof. Suppose for the sake of contradiction, there exists such a protocol. Consider the parties being partitioned into the following two sets: A: a set of $\lceil t/2 \rceil$ parties, and B: all remaining parties which includes the designated sender r.

We consider two executions W1 and W2 where the third property of weak gradecast (i.e., if an honest party outputs a value v with a grade of 2, all other honest parties output value v with a

grade ≥ 1) is violated in the W2. In the first execution (W1), all parties in A are Byzantine. Parties in A do not communicate with each other. Towards B, parties in A execute honestly except they ignore the first $\lceil t/2 \rceil$ messages from parties in B. The designated sender $r \in A$ sends value v to all parties. Since, the maximum faults in W1 is $\lceil t/2 \rceil$ and the designated sender is honest, all honest parties decide value v with a grade of 2.

Since the communication complexity of the protocol is at most $t^2/4$, there must exist a party (say s) in A that receives at most $t/2$ messages from parties in B; otherwise the communication complexity will be more than $t^2/4$. Let B_s be the set of all parties that send messages to party s in W1.

In the second execution (W2), all parties in $A \setminus \{s\}$ are Byzantine and all parties in B_s are Byzantine which includes the designated sender r. The total number of Byzantine parties is $(\lceil t/2 \rceil - 1) + \lceil t/2 \rceil \leq t$ which is within allowed fault threshold t. The designated sender r sends value v. The parties in B_s execute the protocol in the same way as in W1 except they do not send any messages to party s. Parties in $A \setminus \{s\}$ execute the protocol in the same way as in W1. Party s in W1 behave as an honest party which did not receive the first $\lceil t/2 \rceil$ messages which is similar to party s in W2 which receives no messages. Thus, parties in $B \setminus B_s$ cannot distinguish W1 and W2. Thus, they decide value v with a grade of 2. Since, party s does not receive any messages in W2, it does not decide v with a grade ≥ 1. This violates the third property of weak gradecast where if an honest party outputs a value v with a grade of 2, then all honest parties need to output a value v with a grade of ≥ 1. A contradiction. □

Theorem 74. *Let $\mathcal{CC}(\ell)$ be the communication complexity of weak gradecast for ℓ bit input. Then $\mathcal{CC}(\ell) = \Omega(n\ell + n^2)$*

Proof. Since each party must learn ℓ bit input, the protocol needs $\Omega(n\ell)$ bits (The argument follows from [56]). From Lemma 73, weak gradecast requires $\Omega(n^2)$ even for a single bit input. Thus, $\mathcal{CC}(\ell) = \Omega(n\ell + n^2)$ for ℓ bit input. □

Chapter 7

Communication and Round Efficient Parallel Broadcast Protocols

7.1 Introduction

Parallel broadcast is a primitive where all parties wish to broadcast ℓ bit messages in parallel. It is an important building block, central to many cryptographic protocols like verifiable secret sharing, multi-party computation [16, 19], where all parties often broadcast ℓ bit messages in parallel in the same round. Design of efficient protocols for parallel broadcast is of paramount importance as any improvements for parallel broadcast also results in improvement of these primitives. In this work, we focus on improving the communication complexity (i.e., reducing the number of bits honest parties exchange) and the round complexity (i.e., the time required to reach a decision) of parallel broadcast in the synchronous authenticated model with PKI and digital signatures tolerating $t < n/2$ Byzantine failures under various setup assumptions.

Existing works on parallel broadcast either naïvely run n instances of Byzantine agreement (or Byzantine broadcast) primitives (increasing the communication complexity by undesirable factor of n) [18, 1] or incur high round complexity along with strong cryptographic assumptions [103]. While existing solutions for parallel broadcast have optimal fault tolerance of $t < n$ [43] or nearly optimal fault tolerance of $t < (1 - \epsilon)n$ [103], they incur high communication and $\Omega(t)$ round complexity. This work investigates the communication complexity and round complexity of parallel broadcast protocol when the fault tolerance is $t < n/2$. To be specific, we ask the following question:

Can we design a parallel broadcast protocol with $o(\kappa n^4)$ communication (κ denotes a security parameter) and a good round complexity while tolerating $t < n/2$ Byzantine faults?

We answer this question affirmatively by showing two parallel broadcast protocols each with $O(n^2\ell + \kappa n^3)$ communication for inputs of size ℓ bits and termination in constant expected rounds. Thus, for inputs for size $\ell = \Omega(n)$ bits, our protocols have no asymptotic overhead. Our first protocol works in the authenticated model with PKI and digital signatures and is secure against a static adversary. Our second protocol relies on threshold setup assumption to obtain security against a (strongly rushing) adaptive adversary.

7.1.1 Key Technical Ideas and Results

Parallel broadcast is a primitive where all parties wish to broadcast ℓ bit messages in parallel [91]. It can be implemented naïvely by invoking n instances of Byzantine broadcast [43] or Byzantine agreement [4] primitives in parallel in a "black box" manner. However, this technique increases the communication complexity by an undesirable factor of n. Moreover, invoking n concurrent instances of randomized Byzantine agreement protocol [4] (that terminates in expected $O(1)$ rounds) terminates in expected $O(\log n)$ rounds [18]; thus increasing the round complexity. Our work focuses on improving communication complexity while keeping a constant expected round complexity.

Towards communication efficient parallel broadcast. Instead of relying on n instances of expensive Byzantine Broadcast (or Byzantine Agreement) primitive, we obtain parallel broadcast using a combination of n instances of a gradecast primitive [71, 101] and only one instance of (validated) agreement protocol. To ensure an overall communication complexity of $O(n^2\ell + \kappa n^3)$ for inputs of size ℓ bits, we improve the communication complexity of gradecast to $O(n\ell + \kappa n^2)$ and the validated agreement protocol to $O(n\ell + \kappa n^2)$ in expectation. In the following, we will first describe our improvements to each of the primitives, before describing parallel broadcast.

Gradecast with multiple grades. Gradecast is a relaxed version of broadcast introduced by Feldman and Micali [49] where parties output a value along with a grade. Basic versions of gradecast [71, 101] have grades in the range of $\{0, 1, 2\}$. We rely on a version of gradecast that supports grades in the range $\{0, 1, 2, 3, 4\}$ (we will explain later the need for this version of gradecast). At a high level, our gradecast with multiple grades provides the following guarantees: (i) the grades of all honest parties are maximum i.e., 4 when the sender is honest, (ii) honest parties may output

different grades when the sender is Byzantine; but the grades of any two honest parties differ by at most 1, (iii) when an honest party outputs a value with a grade of 2, all honest parties output the same value with grade of ≥ 1, (iv) two honest parties may output different values with a grade of 1 when no honest party has a grade of 2.

We give a construction with a communication complexity of $O(n\ell + \kappa n^2)$. The key technique we employ to design communication efficient gradecast is to have parties multicast smaller chunks of messages (via extension techniques [86]) only once and then "silently" waiting to detect any conflicting messages while simultaneously increasing the grades when no conflicting messages are detected.

While gradecast with grades up to 4 suffices for our purpose, we generalize it to arbitrary number of grades $\{0, 1, \ldots, g^*\}$ where g^* is the maximum supported grade. We note that Garay et al. [57] also formulated gradecast with multiple grades and gave a construction with a communication complexity of $O(g^*(\ell + \kappa)n^2)$ for ℓ bit input. We give a slightly relaxed definition.[1] We obtain the following result:

Theorem 75. *Assuming a public-key infrastructure, digital signatures and a universal structured reference string under q-SDH assumption, there exists a g^*-gradecast protocol tolerating $t < n/2$ Byzantine faults with $O(n\ell + \kappa n^2)$ communication for an input of size ℓ bits and a round complexity of $3g^* - 2$.*

Graded parallel broadcast: Composing n instances of gradecast with multiple grades and ensuring validated output. Parties invoke gradecast with multiple grades with each party as a sender to propagate their input and output an n-element list of grades (GradeList) corresponding to each party as sender. Note that GradeList of two honest parties may be different; especially the grades corresponding to a Byzantine sender. Looking ahead, our aim is to feed the GradeList of each party into a multi-valued validated Byzantine agreement (MVBA) protocol to agree on a common GradeList and output the final vector based on the grades in the agreed GradeList to solve the parallel broadcast problem.

Note that in MVBA, the output GradeList can be an input of any party, including a Byzantine party as long as the output GradeList meets some validity conditions. However, a Byzantine party may set arbitrary grades corresponding to honest senders. In order to restrict a Byzantine party from setting arbitrary grades corresponding to honest senders in its GradeList, we define the notion

[1] Our relaxation allows honest parties to output different values with a grade of 1 when no honest party has a grade of 2 while they require honest parties to output the same value with a grade of 1.

of *valid* GradeList. A *valid* GradeList is one that has been verified by at least one honest party. An honest party verifies a given GradeList by checking against its own GradeList and ensuring that the grades corresponding to a sender differ by at most 1. This restricts the grades corresponding to honest parties to be in a specific range in a *valid* GradeList.

Given this notion of valid GradeList, let us see why we need a gradecast that supports grades in the range $\{0, 1, 2, 3, 4\}$ where honest parties output a common value with the highest grade of 4 when the sender is honest. Consider a Byzantine party who may set arbitrary grades corresponding to an honest sender in its GradeList. For its GradeList to be valid, it must set a grade of at least 3 for its GradeList to be verified by an honest party. Then we can compute the final output vector (to solve parallel broadcast) by considering values that have grades at least 3 in the agreed valid GradeList. This ensures honest inputs are always included in the final output vector. Note that the Byzantine party may also set a grade of at least 3 corresponding to a Byzantine sender in its GradeList. The GradeList will be verified as long as an honest party has a grade of at least 2 corresponding to this Byzantine sender. Note that our gradecast protocol ensures that all honest parties have output the same value when an honest party sets a grade of at least 2. This ensures consistency in the final output vector.

To see why gradecast protocol that supports fewer grades does not work, let us conider a consider a gradecast where the maximum grade is 3. We consider a GradeList of a Byzantine party who may set a grade of 2 corresponding to an honest sender (to ensure the GradeList is verified). In this version, in order to ensure honest inputs are included in the final vector, we need to output values with grades of at least 2 in the agreed GradeList. However, the Byzantine party may also set a grade of 2 corresponding to Byzantine sender for which no honest party has a grade of 2; different honest parties may have different values in this case. Thus, this violates consistency.

We formally define the process of invoking n parallel instances of gradecast with multiple grades and obtaining (possibly different) valid GradeList as *graded parallel broadcast*. We obtain the following result,

Theorem 76. *Assuming a public-key infrastructure, digital signatures and a universal structured reference string under q-SDH assumption, there exists a graded parallel broadcast protocol tolerating $t < n/2$ Byzantine faults with $O(n^2\ell + \kappa n^3)$ communication for an input of size ℓ bits and constant rounds.*

Agreeing on a common valid GradeList using efficient multi-value validated Byzantine agreement. We make use of a single instance of multi-valued validated Byzantine agreement

Table 7.1: **Comparison of related works on MVBA with ℓ-bit input**

	Model	Resilience	Communication	Latency	Adversary
Shrestha et al. [101]	PKI	$t < n/2$	$E(O(n^2\ell + \kappa n^3))$	$E(O(1))$	static
This work	threshold setup	$t < n/2$	$E(O(n\ell + \kappa n^2))$	$E(O(1))$	adaptive

$E(.)$ implies "in expectation".

(MVBA) to agree on a common GradeList. In MVBA, each party starts with a different externally valid input (possibly large) and outputs a common value; the output value can be input of any party as long as it is externally valid. To the best of our knowledge, the MVBA protocol from Chapter 6 which works in the authenticated model with PKI and digital signatures is the only known synchronous MVBA protocol. That protocol is secure against a static adversary tolerating $t < n/2$ faults with $O(n^2\ell + \kappa n^3)$ communication in expectation and expected $O(1)$ rounds.

In order to improve communication complexity and provide security against an adaptive security, in this chapter, we design an MVBA protocol secure against a (strongly rushing) adaptive adversary tolerating $t < n/2$ Byzantine faults. Our MVBA protocol incurs $O(n\ell + \kappa n^2)$ communication in expectation and terminates in expected constant rounds but assumes threshold setup and relies on adaptively-secure threshold signature scheme [78]. Following the communication lower bound results of Abraham et al. [2] and Fitzi et al. [56], our MVBA protocol has optimal communication complexity. Specifically, we show the following result:

Theorem 77. *Assuming a public-key infrastructure, digital signatures, threshold setup and a universal structured reference string under q-SDH assumption, there exists a multi-valued validated Byzantine agreement protocol tolerating $t < n/2$ Byzantine faults with $O(n\ell + \kappa n^2)$ communication in expectation for inputs of size ℓ bits, termination in expected $O(1)$ rounds and security against a (strongly rushing) adaptive adversary.*

The starting point of our MVBA construction is the Byzantine synod protocol of Abraham et al. [4] which is secure against a (strongly rushing) adaptive adversary and incurs $O((\ell + \kappa)n^2)$ communication in expectation and terminates in expected constant rounds. We present a brief overview of their protocol to understand $O(n^2\ell)$ term.

In their protocol, parties first multicast their ℓ-bit proposals and collect acknowledgements from at least $t + 1$ parties. A proposal is said to be "prepared" if it collects acknowledgements from $t + 1$ parties. Each of the parties then propose these prepared proposals. This is followed by a leader election phase where they always obtain a common leader. With probability at least $1/2$,

this leader is honest. Once the leader is elected, parties only consider the prepared proposal of the leader. If such a proposal exists and there are no equivocating prepared proposals from the leader, the proposal is committed; otherwise parties perform a "view-change" to restart the process. Having parties create prepared proposals before a leader election prevents an adaptive adversary from corrupting the elected party and creating equivocating proposals. For proposals of size ℓ bits each, a multicast of n proposals trivially incurs $O(n^2\ell)$ communication even with the use of extension techniques [86].

Our MVBA protocol inherits the underlying consensus mechanism of their protocol and improves the dissemination of the proposals to obtain $O(n\ell + \kappa n^2)$ communication. Our solution uses Reed-Solomon erasure codes [95] to decode large messages into n code words and cryptographic accumulators [88] to verify the correctness of the code words.

In our protocol, each party P_i encodes its ℓ bit proposal to n code words $(s_{i,1}, \ldots, s_{i,n})$ via Reed-Solomon erasure codes and sends a code word $s_{i,j}$ to party P_j $\forall j \in [n]$ along with a cryptographic witness to verify the correctness of the code word $s_{i,j}$. Each party P_j, upon receiving a valid code word $s_{i,j}$, sends an acknowledgment to party P_i. Party P_i considers its proposal "prepared" once it receives $t + 1$ acknowledgments. We stress that a party receives a single valid code word corresponding to the proposal; and not the full proposal. This differs from extension techniques [86] where all parties receive the full proposal. For all n proposals each of size ℓ bits, this process only incurs $O(n\ell + \kappa n^2)$ communication. Having proposals prepared in this manner still gives that same advantages against an adaptive adversary with reduced communication.

Later in the protocol, when the prepared proposal is selected during leader election phase, the full proposal needs to be retrieved before committing it. An original proposal can be decoded with $t + 1$ valid code words for the proposal. Note, however that a "prepared" proposal does not imply sufficient code words required to decode the proposal will be available. A Byzantine party may send a valid code word corresponding to its proposal to a single honest party and collect t acknowledgements from Byzantine parties to have its proposal prepared. Thus, having a proposal prepared does not guarantee its availability. We consider such proposals as "bad". When such a bad proposal is selected during the leader election phase, we "wait" for a few rounds to detect recoverability of the proposal and perform view-change when we are unable to decode the selected proposal i.e., we rely on synchrony to detect and filter out bad proposals. Once an honest leader is elected, its prepared proposal can be decoded and committed.

Efficient parallel broadcast. Finally, we obtain efficient protocols for parallel broadcast using the above primitives. In particular, we use the graded parallel broadcast and MVBA protocol to

Table 7.2: **Comparison of related parallel broadcast protocols**

	Model	Resilience	Communication	Latency	Adversary
Tsimos et al. [103]	PKI	$t < (1-\epsilon)n$	$\tilde{O}(\kappa^2 n^3 \ell)$	$O(t \log t)$	adaptive
Tsimos et al. [103]	trusted PKI	$t < (1-\epsilon)n$	$\tilde{O}(\kappa^4 n^2 \ell)$	$O(\kappa \log t)$	adaptive
Abraham et al. [1]	unauthenticated	$t < n/3$	$O(n^2\ell) + E(O(n^4 \log n))$	$E(O(1))$	static
This work + [101]	PKI	$t < n/2$	$O(n^2\ell) + E(O(\kappa n^3))$	$E(O(1))$	static
This work	threshold setup	$t < n/2$	$O(n^2\ell + \kappa n^3) + E(O(\kappa n^2))$	$E(O(1))$	adaptive

Tsimos et al. [103] and Abraham et al. [1] do not assume q-SDH assumption. Tsimos et al. [103] has $\tilde{O}$ in the communication complexity which hides a $\log n$ factor unrelated to the q-SDH assumption. Without q-SDH setup assumption, our protocols would have $\log n$ multiplicative factor in the communication complexity. $E(.)$ implies "in expectation".

achieve parallel broadcast protocol. Specifically, we obtain the following main result:

Theorem 78. *Assuming a public key infrastructure and digital signatures, if we have a graded parallel broadcast tolerating $t < n/2$ Byzantine faults with a communication complexity of x and round complexity of y, and a MVBA protocol tolerating $t < n/2$ Byzantine faults with a communication complexity of a and a round complexity of b, we can have a parallel broadcast protocol tolerating $t < n/2$ Byzantine faults with a communication complexity of $x+a$ and a round complexity of $y+b$.*

We obtain different results for parallel broadcast depending on the variant of the validated Byzantine agreement used. Our first parallel broadcast protocol uses the MVBA protocol from Chapter 6 which is a secure against a static adversary with $O(n^2\ell + \kappa n^3)$ communication in expectation and expected $O(1)$ rounds. Using this MVBA protocol, we obtain the following corollary:

Corollary 79. *Assuming a public-key infrastructure, digital signatures, and a universal structured reference string under q-SDH assumption there exists a protocol secure against static adversary that solves parallel broadcast tolerating $t < n/2$ Byzantine faults with $O(n^2\ell) + E(O(\kappa n^3))$ communication and expected $O(1)$ rounds.*

Our second parallel broadcast protocol uses our MVBA protocol (Theorem 71). We obtain the following corollary:

Corollary 80. *Assuming a public-key infrastructure, digital signatures, threshold setup, and a universal structured reference string under q-SDH assumption there exists a protocol that solves parallel broadcast tolerating $t < n/2$ Byzantine faults with $O(n^2\ell+\kappa n^3)+E(O(\kappa n^2))$ communication, termination in expected $O(1)$ rounds and security against a (strongly rushing) adaptive adversary.*

Observe that our second parallel broadcast has $O(n^2\ell + \kappa n^3) + E(O(\kappa n^2))$ communication. In the common case, the protocol terminates in expected constant number of rounds with total communication complexity of $O(n^2\ell + \kappa n^3)$. In the worst case, when the protocol runs for linear number of rounds, this protocol still incurs $O(n^2\ell+\kappa n^3)$ communication; thus this protocol incurs $O(n^2\ell+\kappa n^3)$ communication even in the worst-case.

Related Work. Table 7.1 and Table 7.2 presents comparisons with recent results in MVBA and parallel broadcast literature. We present a detailed discussion in Section 7.7.

7.2 Model and Preliminaries

We consider a system consisting of n parties $(P_1, \ldots, P_n)$ in a reliable, authenticated all-to-all network, where up to $t < n/2$ parties can be Byzantine faulty. The Byzantine parties may behave arbitrarily. We consider two kinds of adversaries: (i) a static adversary, and (ii) a strongly rushing adaptive adversary. A static adversary corrupts parties before the start of the protocol execution whereas a strongly rushing adaptive adversary can adaptively decide which t parties to corrupt at any time during protocol execution. In addition, due to "strongly-rushing" nature of the adversary, the adversary is capable of corrupting a party P_h after observing message sent by party P_h in round r and remove round r messages sent by party P_h before they reach other honest parties and send round r messages after corrupting it [4]. A party that is not faulty throughout the execution is considered to be *honest* and executes the protocol as specified.

We assume a synchronous communication model. Thus, if an honest party sends a message at the beginning of some round, the recipient receives the message by the end of that round. We make use of digital signatures and a public-key infrastructure (PKI) to prevent spoofing and replays and to validate messages. Message x sent by a node P_i is digitally signed by P_i's private key and is denoted by $\langle x \rangle_i$. In addition, we use $H(x)$ to denote the invocation of the random oracle H on input x.

7.2.1 Definitions

Gradecast with multiple grades. Gradecast with multiples grades was originally introduced by Garay et al. [57] that supports arbitrary number of grades. We present a slightly different definition of gradecast with multiple grades.

Definition 7.2.1 (Gradecast with multiple grades)**.** *A protocol with a designated sender P_i holding an initial input v is a g^*-gradecast protocol tolerating t Byzantine faults if the following conditions hold:*

1. *Each honest party P_j outputs a value v_j with a grade $g_j \in \{0, 1, \ldots, g^*\}$.*
2. *If the sender is honest, each honest party P_j outputs v with a grade $g_j = g^*$.*
3. *If two honest parties P_j and P_k output values with grades g_j and g_k respectively, then $|g_j - g_k| \leq 1$.*
4. *If an honest party P_j outputs a value v with a grade $g_j > 1$, then all honest parties output value v.*

7.2.2 Primitives

In this section, we present several primitives used in our protocols.

Linear erasure and error correcting codes. We use standard $(t+1, n)$ Reed-Solomon (RS) codes [95]. This code encodes $t+1$ data symbols into code words of n symbols using ENC function and can decode the $t+1$ elements of code words to recover the original data using DEC function. More details on ENC and DEC functions are provided in Section 2.5.

Cryptographic accumulators. A cryptographic accumulator scheme constructs an accumulation value for a set of values using Eval function and produces a witness for each value in the set using $\mathsf{CreateWit}$ function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. More details on these functions are provided in Section 2.5.

In this chapter, we use *collision free bilinear accumulators* from Nguyen [88] as cryptographic accumulators which generates constant sized witness, but requires q-SDH assumption. Alternatively, we can use Merkle trees [81] (and avoid q-SDH assumption) at the expense of $O(\log n)$ multiplicative communication.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator. Further, let $\kappa = \mathsf{max}(\kappa_h, \kappa_a)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_a) = \Theta(\lambda)$.

Throughout the chapter, we will use the same parameter κ to denote the hash size, signature size and accumulator size for convenience.

7.3 Gradecast with Multiple Grades

In this section, we present a communication efficient gradecast protocol that supports multiple grades. Gradecast (aka graded broadcast) is a relaxed version of broadcast introduced by Feldman and Micali [49]. In gradecast, parties output a value along with a grade. Informally, the grade output by a party is an indicator of the "confidence" in the output produced by it. Thus, when the grade output by an honest party is *high*, other honest parties are expected to output the same value (even though their grade may be lower). When the grades are *lower*, there may be some amount of disagreement between the output values of different honest parties too. This is in contrast to broadcast which requires honest parties to reach a unanimous decision.

The gradecast protocol of Feldman and Micali [49] supports three grades $\{0, 1, 2\}$ and their protocol tolerates $t < n/3$ Byzantine faults in the *plain* authenticated model without PKI. Later, Garay et al. [57] generalized the gradecast protocol to the case of an arbitrary number of grades $\{0, 1, \ldots, g^*\}$ where g^* is the maximum supported grade. They gave a protocol in the authenticated model with PKI and digital signatures tolerating $t < n$ Byzantine faults and a communication complexity of $O(g^*(\ell + \kappa)n^2)$ for input of size ℓ bits and a round complexity of $2g^* + 1$. In this work, we present a slightly different definition of the gradecast with multiple grades and show a construction that satisfies this definition with a communication complexity of $O(n\ell + \kappa n^2)$ for input of size ℓ bits.

Our definition of gradecast with multiple grades differs from the definition of Garay et al. [57] in the following ways. First, our definition allows honest parties to output different values with a grade of 1 when no honest party outputs a grade > 1 while their definition restricts honest parties to output the same value with a grade of 1. Second, our definition requires the grades of any two honest parties to differ by at most 1, i.e., for any two honest parties P_j and P_k, we require $|g_j - g_k| \leq 1$ while the definition of Garay et al. [57] only requires $g_k \geq g_j - 1$ when $g_j \geq 2$.

Next, we construct a protocol $\mathsf{M\text{-}Gradecast}(v, g^*)$ where v is the sender's value and g^* is the maximum supported grade. $\mathsf{M\text{-}Gradecast}(v, g^*)$ works in the authenticated model with PKI and digital signatures and tolerates $t < n/2$ Byzantine faults.

Deliver. As a building block, we first present a Deliver function (refer Figure 4.2) used by an

honest party to efficiently propagate long messages. This function is adapted from Chapter 4 where linear-sized messages are propagated among all honest parties with $O(\kappa n^2)$ communication cost. The Deliver function enables efficient propagation of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are a keyword mtype, long message m and an accumulation value z_e corresponding to message m. The input keyword mtype corresponds to message *type* containing long message m sent by its sender. In order to facilitate efficient leader equivocation, the input keyword mtype, hash of long message m and accumulation value z_e are signed by the sender of message m. The Deliver function incurs 2 rounds.

Equivocation. Two or more messages of the same *type* but with different payload sent by a party is considered an equivocation. In order to facilitate efficient equivocation checks, the sender sends the payload along with signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by the sender.

Set $o_i = \perp$ and $g_i = \perp$. Each party P_i performs the following operations:

- **Round 1:** If party P_j is the designated sender, then it multicasts its input value v in the form of $\langle \mathsf{gcast}, v, z \rangle_j$ where z is the accumulation value of v.
- **Round $2h$ ($h \in [1, g^* - 1]$):** If party P_i receives $pr := \langle \mathsf{gcast}, v, z \rangle_j$ for the first time, then invoke Deliver(gcast, pr, z).
- **Round $2g^*$:** If party P_i invoked Deliver and no party P_j equivocation has been detected so far, set $o_i = v$ and $g_i = 2$. Let v_i be the first value received. If $v_i = \perp$, set $o_i = \perp$ and $g_i = 0$, else if $o_i = \perp$, set $o_i = v_i$ and $g_i = 1$.
- **Round $2g^* + h$ ($h \in [1, g^* - 2]$):** If party P_i invoked Deliver for value v by Round $2g^* - 2(h + 1)$ and no party P_j equivocation has been detected so far, set $g_i = g_i + 1$. At Round $3g^* - 2$, output (o_i, g_i).
- **At any round:** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 7.1: **M-Gradecast(v, g^*) with $O(n\ell + (\kappa + w)n^2)$ communication.**

The M-Gradecast(v, g^*) protocol is presented in Figure 7.1. In round 1, the designated sender P_j sends value v by multicasting $\langle \mathsf{gcast}, v, z \rangle_j$ where z is the accumulation value for value v. We note that the size of input value v can be large. In order to facilitate efficient equivocation checks, the sender P_j signs $\langle \mathsf{gcast}, H(v), z \rangle$ and sends v separately. Whenever an equivocation by the sender is detected, multicasting signed hashes suffices to prove equivocation by the sender. Note that the size of the signed message $\langle \mathsf{gcast}, H(v), z \rangle$ is $O(\kappa)$ bits. Thus, all-to-all multicast of the signed message $\langle \mathsf{gcast}, H(v), z \rangle$ incurs only $O(\kappa n^2)$ communication. The reduction in communication is obtained via the use of efficient erasure coding schemes [95], cryptographic accumulators [15] and multicast of equivocating hashes (if any). Multicasting of equivocating hashes been explored in several communication efficient BFT protocols [100, 10, 22].

During rounds $2h$ for $h \in [1, g^* - 1]$, if party P_i receives $\langle \mathsf{gcast}, v, z \rangle_j$ for the first time, it invokes

Deliver to propagate long message v, i.e., if party P_i invoked Deliver in round 2, it does not invoke Deliver again in later rounds. Note that Deliver function requires 2 rounds. Rounds $2h+1$ for $h \in [1, g^*-1]$ accommodates steps of Deliver function invoked in rounds $2h$ for $h \in [1, g^*-1]$. We note that although parties may invoke Deliver to propagate long message v in different rounds, they forward their code words only the first time. For example, if a party P_i invoked Deliver in round 2 and an honest party P_k received its first valid code word (s_k, w_k) in round 3 for accumulator z, it forwards the code word to all parties in round 3. Later, if some other party (say party P_h) invokes Deliver in round 4 and party P_k receives code word (s_k, w_k) again in round 5, party P_k does not forward (s_k, w_k) again. This helps in keeping communication complexity to $O(n\ell + \kappa n^2)$.

In round $2g^*$, each party P_i sets its output value and initial grades. If party P_i invoked Deliver for value v at any prior rounds, and it did not detect any equivocation so far, it sets $o_i = v$ and $g_i = 2$. We note that an honest party decodes long messages corresponding to the first valid code word they receive even though it detects equivocation as long as it receives $t+1$ valid code words. Let v_i be the first value received. If $v_i = \perp$, it sets $o_i = \perp$ and $g_i = 0$. Otherwise if $o_i = \perp$, set $o_i = v_i$ and $g_i = 1$ irrespective of the equivocation.

In round $2g^* + h$ for $h \in [1, g^* - 2]$, each party P_i updates their grade g_i based on when they invoked Deliver and if they have detected any equivocation so far.

Optimal communication complexity. Our M-Gradecast(v, g^*) incurs $O(n\ell + \kappa n^2)$ communication for input of ℓ bits. In Chapter 6, we showed a communication lower bound of $\Omega(n\ell + n^2)$ for weak-gradecast problem where grades are in the range $\{0, 1, 2\}$ for input of size ℓ bits. The communication lowerbound can trivially be extended to show the optimal communication complexity of our M-Gradecast(v, g^*) protocol.

7.3.1 Security Analysis

Claim 81. *Suppose party P_j is the designated sender. If an honest party invokes* Deliver *in round r for a value m sent by party P_j and no honest party has detected a party P_j equivocation by round $r+1$, then all honest parties will receive value m by round $r+2$.*

Proof. Suppose an honest party P_i invokes Deliver at round r for a value m sent by party P_j. Party P_i must have sent valid code words and witness $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e \rangle_i$ computed from value m to every party P_k $\forall k \in [n]$ at round r. The code words and witness arrive at all honest parties

by round $r+1$.

Since no honest party has detected a party P_j equivocation by round $r+1$, it must be that either honest parties will forward their code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e \rangle$ when they receive the code words sent by party P_i or they already sent the corresponding code word when they either invoked Deliver for value m or received the code word from some other party. In any case, all honest parties will forward their code word corresponding to value m by round $r+1$. Thus, all honest parties will have received $t+1$ valid code words for a common accumulation value z_e by round $r+2$ sufficient to decode value m. □

Theorem 82. *The protocol in Figure 7.1 is a g^*-gradecast protocol satisfying Definition 7.2.1.*

Proof. Suppose party P_j is the designated sender with its input value v. Let g^* be the maximum grade.

We first consider the case when an honest party P_i outputs value v with a grade $g_i = 2$ and no honest party outputs a value with a grade > 2. Honest party P_i must have invoked Deliver for value v by round $2g^* - 2$ and did not detect a party P_j by round $2g^*$. This implies no honest party detected a party P_j equivocation by round $2g^* - 1$. By Claim 81, all honest parties receive value v by round $2g^*$. In addition, since party P_i invoked Deliver for value v by round $2g^* - 2$, all honest parties receive a code word for value v by round $2g^* - 1$. Thus, value v is the first value received by all honest parties. Since $v \neq \perp$, all honest parties will output value v with a grade ≥ 1.

Next, we consider the case when an honest party P_i outputs a value v with a grade $g_i > 2$. Without loss of generality, assume g_i is the highest grade output by any honest party. Let $h = g_i - 2$. Since, party P_i outputs value v with a grade $g_i > 2$, it must have invoked Deliver to propagate value v by round $2g^* - 2(h+1)$ and did not detect any party P_j equivocation by round $2g^* + h$. This implies no other honest party detected a party P_j equivocation by round $2g^* + h - 1$. With $h \geq 1$, $2g^* + h - 1 > 2g^* - 2(h+1) + 1$. Thus, by Claim 81, all other honest parties receive value v by round $2g^* - 2h$. The honest parties that did not invoke Deliver by round $2g^* - 2(h+1)$ will invoke Deliver for value v by round $2g^* - 2h$. Since no other honest party detected a party P_j equivocation by round $2g^* + h - 1$, all honest parties will set a grade of 2 in round $2g^*$. In addition, all honest parties will set a grade of at least $2 + h - 1 = g_i - 1$ by round $2g^* + h - 1$. Thus, all honest parties will output value v with a grade at least $g_i - 1$.

This also proves that if an honest party P_i outputs a value v with a grade $g_i > 1$, then all honest parties output value v.

Next, we consider the case when the designated sender is honest. Since, the sender is honest, it sends its input value v to all honest parties such that all honest parties receive value v in round 2. Thus, all honest parties invoke Deliver to propagate value v in round 2. Moreover, the honest sender does not equivocate. Thus, all honest parties set a grade of 2 in round $2g^*$ and set a grade of $2 + g^* - 2 = g^*$ in round $3g^* - 2$.

The case where each honest party outputs a value with a grade $\in \{0, 1, \ldots, g^*\}$ is trivial by design. □

Lemma 83 (Communication Complexity)**.** *Let ℓ be the size of the input, κ be the size of accumulator, and w be the size of witness. The communication complexity of the protocol in Figure 7.1 is $O(n\ell + (\kappa + w)n^2)$.*

Proof. At the start of the protocol, the sender multicasts its value of size ℓ to all party P_j $\forall j \in [n]$ along with κ sized signed message containing accumulator and hash of large message. This step incurs $O(n\ell + \kappa n)$. An honest party invokes Deliver only on the first value it receives where it sends a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ to each party. Moreover, each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, for all honest parties, this process incurs $O(n\ell + (\kappa + w)n^2)$ and the overall complexity is $O(n\ell + (\kappa + w)n^2)$. □

7.4 Graded Parallel Broadcast

In this section, we present a new primitive that we call *Graded Parallel Broadcast*. Graded parallel broadcast is a relaxation of parallel broadcast [103] and uses gradecast with multiple grades to propagate its input. In this work, we consider an instance of gradecast with multiple grades where the grades can be in the range $\{0, 1, \ldots, 4\}$. In our construction, each party P_i uses M-Gradecast$(., 4)$ to propagate its input v_i and output an n-element list of values along with an n-element list of grades ($\mathsf{GradeList}_i$). Looking ahead, our aim is to have each party P_i feed its output of graded parallel broadcast (i.e., $\mathsf{GradeList}_i$) into a Byzantine consensus primitive to agree on a common $\mathsf{GradeList}_h$. The agreed $\mathsf{GradeList}_h$ can be a Byzantine parties' input too. However, a Byzantine party may set arbitrary grades in its GradeList corresponding to an honest sender and prevent honest input from appearing in the final output. In order to prevent this scenario, we restrict a Byzantine party from setting arbitrary grades and consider only a *valid* GradeList. A *valid* GradeList has (i) at least $n - t$ entries of grade 4, i.e., $|\{h \mid \mathsf{GradeList}[h] = 4\}| \geq n - t$, (ii) $\mathsf{GradeList}[i] \in \{3, 4\}$

corresponding to honest sender P_i. Note that for an honest sender P_k, each honest party P_i sets a grade $\mathsf{GradeList}_i[k] = 4$. Thus, a valid $\mathsf{GradeList}$ must have at least $n - t$ entries of 4. Moreover, due to the properties of $\mathsf{M\text{-}Gradecast}(., 4)$, the grades of two parties for the same sender can differ by at most 1. Since each honest party sets a grade of 4 for an honest sender P_k, a Byzantine party must set a grade of at least 3 corresponding to an honest sender P_k for its $\mathsf{GradeList}$ to be valid. In the final parallel broadcast protocol, we consider all values with grades in the range $\{3, 4\}$ corresponding to agreed $\mathsf{GradeList}$.

In graded parallel broadcast, we ensure that a valid $\mathsf{GradeList}$ is *certified*, i.e., it is accompanied by a set of signatures from at least $t + 1$ parties. A set of $t + 1$ signatures on $\mathsf{GradeList}$ forms the certificate for $\mathsf{GradeList}$ and denoted as $\mathcal{AC}(\mathsf{GradeList})$.

Definition 7.4.1 (Graded Parallel Broadcast)**.** *Each party P_i, as a sender, sends its input v_i. Each honest party P_j outputs an n-element list of values along with a n-element list $\mathsf{GradeList}_j$ with an entry corresponding to each party as a sender such that $\mathsf{GradeList}_j[h] \in \{0, 1, 2, 3, 4\}\ \forall h \in [n]$. A graded parallel Broadcast protocol tolerating t Byzantine failures satisfies the following properties:*

1. *If sender P_i is honest, then each honest party P_j sets $\mathsf{GradeList}_j[i] = 4$.*
2. *A certified $\mathsf{GradeList}_k$ must have $|\{h \mid \mathsf{GradeList}_k[h] = 4\}| \geq n - t$.*
3. *If the sender P_i is honest and $\mathsf{GradeList}_k$ is certified, then $\mathsf{GradeList}_k[i] \in \{3, 4\}$.*
4. *If $\mathsf{GradeList}_k$ is certified and $\mathsf{GradeList}_k[i] \in \{3, 4\}$, then all honest parties have received a common value v_i.*

Each party P_i with its initial input v_i performs following operations:

1. **(Round 1) Propose.** Each party P_i invokes $\mathsf{M\text{-}Gradecast}(v_i, 4)$.
2. **(Round 10) Propose Grade.** Let $(o_{j,i}, g_{j,i})$ be the output of $\mathsf{M\text{-}Gradecast}$ of party P_i with party P_j as sender. Set $\mathsf{GradeList}_i[j] = g_{j,i}$. Multicast $\langle \mathsf{grade\text{-}list}, \mathsf{GradeList}_i \rangle_i$.
3. **(Round 11) Verify and Ack.** Upon receiving $\langle \mathsf{grade\text{-}list}, \mathsf{GradeList}_j \rangle_j$ from party P_j, if the following conditions hold send $\langle \mathsf{ack}, H(\mathsf{GradeList}_j) \rangle_i$ to party P_j.
   (a) $|\{h \mid \mathsf{GradeList}_j[h] = 4\}| \geq n - t$
   (b) $|\mathsf{GradeList}_j[h] - \mathsf{GradeList}_i[h] < 2|\ \forall h \in [n]$.

Figure 7.2: Graded Parallel Broadcast with $O(n^2\ell + (\kappa + w)n^3)$ communication

Protocol Details. Each party P_i uses $\mathsf{M\text{-}Gradecast}(., 4)$ to propagate its input v_i. At the end of $\mathsf{M\text{-}Gradecast}(., 4)$ invocation, each honest party P_i outputs an n element list of values along with

n element list of grades, denoted by $\mathsf{GradeList}_i$, with an entry corresponding to each party as a sender.

Party P_i then multicasts its $\mathsf{GradeList}_i$ to all other parties. Party P_j then checks the validity of $\mathsf{GradeList}_i$ by checking if (i) $|\{h \mid \mathsf{GradeList}_i[h] = 4\}| \geq n-t$, and (ii) $|\mathsf{GradeList}_j[h] - \mathsf{GradeList}_i[h] < 2|$ $\forall h \in [n]$. The first check ensures that $\mathsf{GradeList}_i$ contains at least $n-t$ entries with $\mathsf{GradeList}_i[h] = 4$. Note that for an honest sender P_k, each honest party P_i outputs a value with $\mathsf{GradeList}_i[k] = 4$. Thus, a valid $\mathsf{GradeList}$ must have at least $n - t$ entries of 4. In addition, due to the properties of $\mathsf{M\text{-}Gradecast}(., 4)$, the grades of any two parties corresponding to a sender differs by at most 1. Thus, a valid $\mathsf{GradeList}$ must satisfy $|\mathsf{GradeList}_j[h] - \mathsf{GradeList}_i[h] < 2|$ $\forall h \in [n]$. This check also prevents a Byzantine party from setting too low grades corresponding to an honest sender; otherwise its $\mathsf{GradeList}$ would not be certified. Thus, a Byzantine party must set a grade of at least 3 corresponding to an honest sender for its $\mathsf{GradeList}$ to be certified.

If the checks pass, party P_j sends $\langle \mathsf{ack}, H(\mathsf{GradeList}_i) \rangle_j$ to party P_i. A set of $t + 1$ ack ($\mathsf{ack\text{-}cert}$) messages for $\mathsf{GradeList}_i$ (denoted by $\mathcal{AC}(\mathsf{GradeList}_i)$) implies at least one honest party has verified $\mathsf{GradeList}_i$.

7.4.1 Security Analysis

Theorem 84. *The protocol in Figure 7.2 is a graded Parallel Broadcast protocol satisfying Definition 7.4.1.*

Proof. If the sender P_i is honest, it propagates its input v_i using $\mathsf{M\text{-}Gradecast}$. By Theorem 82, each honest party P_j output $\mathsf{GradeList}_j$ with $\mathsf{GradeList}_j[i] = 4$.

Next, we consider a certified grade list $\mathsf{GradeList}_k$. The only way $\mathsf{GradeList}_k$ gets certified is if at least one honest party P_j sends an ack for it. If an honest party P_j sends an ack for a grade list $\mathsf{GradeList}_k$, then it must be that (i) $|\{h \mid \mathsf{GradeList}_k[h] = 4\}| \geq n - t$ and (ii) $|\mathsf{GradeList}_k[h] - \mathsf{GradeList}_j[h] < 2|$ $\forall h \in [n]$. Trivially, this implies a certified $\mathsf{GradeList}_k$ must have $|\{h \mid \mathsf{GradeList}_k[h] = 4\}| \geq n - t$. This also implies that if $\mathsf{GradeList}_k[h] \in \{3, 4\}$, $\mathsf{GradeList}_j[h]$ must be at least 2. By the properties of $\mathsf{M\text{-}Gradecast}$ (Definition 7.2.1), if an honest party outputs a value v_h with a grade of > 1, all honest parties output a common value v_h. Thus, all honest parties have common value v_h for all h such that $\mathsf{GradeList}_k[h] = \{3, 4\}$.

Next, we consider the grades in $\mathsf{GradeList}_k[j]$ for an honest sender P_j. We know from the fact that

for an honest sender P_j, by the properties of M-Gradecast$(v, 4)$, all honest parties will set a grade of 4. An honest party P_i will send an ack for $\mathsf{GradeList}_k$ only if $|\mathsf{GradeList}_k[j] - \mathsf{GradeList}_i[j] < 2|$. This implies $\mathsf{GradeList}_k[j]$ must be at least 3 i.e. $\mathsf{GradeList}_k[j] \in \{3, 4\}$. □

Lemma 85 (Communication Complexity)**.** *Let ℓ be the size of commitment* comm*, κ be the size of secret share and accumulator, and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. In the Propose step, each party P_i invokes M-Gradecast$(., 4)$ protocol. By Lemma 83, the communication complexity of one invocation of M-Gradecast protocol is $O(n\ell + (\kappa + w)n^2)$. Thus, this step incurs $O(n^2\ell + (\kappa + w)n^3)$.

In the Propose grade step, each party multicast their GradeList of size $O(n)$. Multicast of $O(n)$-sized GradeList by n parties incurs $O(n^3)$ communication. In the Verify and Ack step, each party sends at most n ack messages. This step incurs $O(\kappa n^2)$ communication. Thus, the total communication complexity is $O(n^2\ell + (\kappa + w)n^3)$ bits. □

7.5 Multi-valued Validated Byzantine Agreement

In this section, we present an efficient protocol for multi-valued validated Byzantine agreement (MVBA) secure against a strongly rushing adaptive adversary. MVBA protocol allows honest parties to agree on any externally valid input; the agreed value can be the input of a Byzantine party as long as it is externally valid. The problem of multi-valued validated Byzantine agreement has been extensively studied in the asynchronous model. In the synchronous model, we gave an MVBA protocol (in Chapter 6) in the authenticated model with PKI and digital signatures tolerating $t < n/2$ Byzantine faults and secure against a static adversary. The MVBA protocol from Chapter 6 incurs a communication complexity of $O(n^2\ell + \kappa n^3)$ communication in expectation for input of size ℓ bits and terminates in expected $O(1)$ rounds.

In this work, we improve upon our result from Chapter 6 by a linear factor in communication and also design an MVBA protocol secure against a strongly rushing adaptive adversary. We make threshold setup assumptions and rely on adaptively-secure threshold signature scheme due to Loss and Moran [78] to perform a perfect leader election where all honest parties obtain a common leader all the time. This assumption provides us with three major advantages (i) the leader election can be performed in $O(\kappa n^2)$ communication, (ii) we can obtain security against an adaptive adversary,

Each party P_i with its input v_i performs following operations:

- **Round 1:** Each party P_i partitions its input v_i into $t+1$ data symbols and encode the $t+1$ data symbols into n code words $(s_{i,1}, \ldots, s_{i,n})$ using ENC function. Compute accumulation value z_{v_i} using Eval function and witness $w_{i,j}$ $\forall s_{i,j} \in (s_{i,1}, \ldots, s_{i,n})$ using $\mathsf{CreateWit}$ function. Send $\langle \mathsf{codeword}, s_{i,j}, w_{i,j}, z_{v_i} \rangle_i$ to party P_j $\forall j \in [n]$.
- **Round 2:** If party P_i receives the first valid code word $\langle \mathsf{codeword}, s_{j,i}, w_{j,i}, z_{v_j} \rangle_j$ for the accumulator z_{v_j}, send an $\langle \mathsf{ack}, z_{v_j} \rangle_i$.
- **Round 3:** Upon receiving $t+1$ distinct $\langle \mathsf{ack}, z_{v_i} \rangle_*$ message, create a threshold signature, denoted as $\mathcal{AC}(z_{v_i})$.

Figure 7.3: Proposal Dispersal with $O(n\ell + \kappa n^2)$ communication

and (iii) since the leader election is perfect (i.e, all honest parties observe a common leader), we only need to ensure the leader's proposal is propagated among all parties; this allows us to obtain $O(n\ell + \kappa n^2)$ communication in expectation and security against an adaptive adversary.

The starting point of our construction is the adaptively-secure Byzantine synod protocol of Abraham et al. [4] which has a communication complexity of $O((\ell + \kappa)n^2)$ for ℓ bit input values and termination in expected 16 rounds. Our MVBA protocol inherits the underlying consensus mechanism of their protocol and improves the dissemination of the proposals to obtain $O(n\ell + \kappa n^2)$ communication. Our solution uses Reed-Solomon erasure codes [95] to decode large messages into n code words and cryptographic accumulators [88] to verify the correctness of the code words. Section 7.1.1 presents the key ideas behind our improvement.

Epoch. Our protocol progresses through a series of numbered epochs. Each epoch lasts for 8 rounds.

Certified values and ranking. A certificate on a value v_i consists of $t+1$ distinct signatures in an epoch e and is represented by $\mathcal{C}_e(v_i)$. Certificates are ranked by epochs, i.e., values certified in a higher epoch has a higher rank. During the protocol execution, each party keeps track of all certified blocks and keeps updating the highest ranked certified block to its knowledge. Parties lock on the highest ranked certified values and do not vote for values other than the locked values to ensure safety of a commit.

7.5.1 Protocol Details

We first present a protocol used by all parties to efficiently distribute their long ℓ bit input v_i at the cost of $O(n\ell + \kappa n^2)$ communication. This protocol is executed before the MVBA protocol (refer Figure 7.4).

Proposal dispersal. In proposal dispersal protocol (refer Figure 7.3), each party makes use of erasure coding techniques and cryptographic accumulators to efficiently distribute its long message. Each party P_i partitions its input v_i into $t+1$ data symbols. The $t+1$ data symbols are then encoded into n code words $(s_{i,1}, \ldots, s_{i,n})$ using ENC function and a corresponding accumulation value z_{v_i} is computed. Then, the cryptographic witness $w_{i,j}$ is computed for each code word $s_{i,j} \in (s_{i,1}, \ldots, s_{i,n})$ using $\mathsf{CreateWit}$. Then, the code word and witness pair $(s_{i,j}, w_{i,j})$ is sent to the partyP_j $\forall j \in [n]$ along with the accumulation value z_{v_i}.

When a party P_j receives the first valid code word $s_{i,j}$ for an accumulation value z_{v_i} such that the witness $w_{i,j}$ verifies the code word $s_{i,j}$, it sends an $\langle \mathsf{ack}, z_{v_i} \rangle_j$ to party P_i. When party P_i receives $t+1$ ack messages for z_{v_i}, it forms an $\mathsf{ack\text{-}cert}$ for value v_i, denoted as $\mathcal{AC}(z_{v_i})$. Note that an $\mathsf{ack\text{-}cert}$ for value v_i does not imply all honest parties have received valid code words corresponding to value v_i; this only implies at least one honest party has received a valid code word corresponding to value v_i. When the sender P_i is honest, then all honest parties will receive a valid code word corresponding to value v_i which is sufficient to decode value v_i. In the MVBA protocol that follows, each party P_i proposes accumulation value z_{v_i} along with $\mathcal{AC}(z_{v_i})$ and honest parties only consider proposals containing an $\mathsf{ack\text{-}cert}$. Collecting an $\mathsf{ack\text{-}cert}$ for a proposal is similar to having a proposal prepared in the Byzantine synod protocol of Abraham et al. [4]. However, it does not guarantee that all honest parties will be able to decode the proposed value.

In the proposal dispersal protocol, each party P_j receives only a single code word $s_{i,j}$ corresponding to value v_i. For n proposals each of size ℓ bit, this protocol incurs $O(n\ell + (\kappa + w)n^2)$ bits where κ is the size of accumulator and w is the size of the accumulator *witness*.

MVBA Protocol. At the start of the MVBA protocol (refer Figure 7.4), no party has a certificate for any proposed value; thus each party P_i sends a status message with an empty certificate. Consequently, $\mathcal{CC}_i = \perp$ for each party P_i and each party P_i multicasts its own value $(z_{v_i}, \mathcal{AC}(z_{v_i}))$ in the propose step of the first epoch. In subsequent epochs, parties send proposals corresponding to the highest ranked certificate known to them. Note that a valid proposal is accompanied by an $\mathsf{ack\text{-}cert}$ which can only be formed during a proposal dispersal phase; this is because a $\mathsf{ack\text{-}cert}$ consists of at least $t+1$ ack for z_{v_i} and honest parties send ack for z_{v_i} only in the proposal dispersal phase. In the MVBA protocol, all parties send their proposals first and a leader is elected in a later round. This prevents an adaptive adversary from corrupting the elected party and sending valid equivocating proposals afterwards; this is because $\mathsf{ack\text{-}cert}$ for an equivocating proposal cannot form afterwards.

In round 3, parties participate in the adaptively-secure threshold coin tossing scheme due to

Each party P_i with its input v_i executes the proposal disperal protocol (refer Figure 7.3) and outputs $\mathcal{AC}(z_{v_i})$. Then, each party P_i performs the following operations for each epoch e:

1. **(Round 1) Status.** Multicast the highest ranked certificate known to party P_i in the form of $\langle \mathsf{status}, \mathcal{C}_{e'}(z_{v_h}), \mathcal{AC}(z_{v_h})\rangle_i$.
2. **(Round 2) Propose.** Let $\mathcal{CC}_i := \mathcal{C}_{e'}(z_{v_h})$ be the highest ranked certificate known to party P_i at the end of Status round. If $\mathcal{CC}_i \neq \perp$, set $\mathsf{val}_i = (z_{v_h}, \mathcal{AC}(z_{v_h}))$; otherwise set $\mathsf{val}_i = (z_{v_i}, \mathcal{AC}(z_{v_i}))$. Each party P_i multicasts $\langle \mathsf{propose}, \mathsf{val}_i, \mathcal{CC}_i, e\rangle_i$.
3. **(Round 3) Elect.** Each party P_i participates in threshold coin tossing scheme from [78]. Let L_e be leader of epoch e.
4. **(Round 4) Forward.** Upon receiving the first valid proposal $\langle \mathsf{propose}, (z_{v_h}, \mathcal{AC}(z_{v_h})), \mathcal{CC}_{L_e}, e\rangle_{L_e}$ forward the proposal. If $\mathcal{CC}_i \leq \mathcal{CC}_{L_e}$, party P_i forwards a valid code word $\langle \mathsf{codeword}, s_{h,i}, w_{h,i}, z_{v_h}, e\rangle_i$ consistent with accumulator z_{v_h} sent by party P_h during proposal dispersal phase (if party P_i received a code word for z_{v_h}).
5. **(Round 5) Decode.** Upon receiving $t+1$ valid code words for the accumulator z_{v_h}, decode v_h using DEC function if party P_i has not already received v_h in earlier epochs. Send $\langle \mathsf{codeword}, s_{h,j}, w_{h,j}, z_{v_h}, e\rangle_i$ to party P_j $\forall j \in [n]$.
6. **(Round 6) Forward2.** If party P_i receives the first valid code word $\langle \mathsf{codeword}, s_{h,i}, w_{h,i}, z_{v_h}, e\rangle_*$ for the accumulator z_{v_h}, forward the code word to all the parties.
7. **(Round 7) Vote.** If party P_i receives v_h by round 5, $\mathcal{CC}_i \leq \mathcal{CC}_{L_e}$, $\mathsf{ex\text{-}validation}(v_h) = \mathsf{true}$ and no equivocating proposal by L_e has been detected so far in epoch e, multicast a vote in the form of $\langle \mathsf{vote}, e, H(z_{v_h})\rangle_i$.
8. **(Round 8) Commit.** Upon receiving $t+1$ distinct vote for z_{v_h} (denoted by $\mathcal{C}_e(z_{v_h})$), multicast $\mathcal{C}_e(z_{v_h})$, commit v_h and multicast $\langle \mathsf{terminate}, e, H(v_h)\rangle_i$.
9. **(At any time) Terminate.** Upon receiving $t+1$ $\langle \mathsf{terminate}, e, H(v_h)\rangle_*$ messages, multicast it, output v_h and terminate.
10. **(At any time) Equivocation.** Multicast the equivocating proposals signed by L_e. Stop performing epoch e operations.

Figure 7.4: **MVBA** with $O(n\ell + \kappa n^2)$ bits communication per epoch and expected $O(1)$ epochs

Loss and Moran [78] to randomly select a common leader L_e for epoch e. The leaders are elected uniformly at random, a common honest leader is elected with probability at least $\frac{1}{2}$. Let $\langle \mathsf{propose}, (z_{v_h}, \mathcal{AC}(z_{v_h})), \mathcal{CC}_{L_e}, e \rangle$ be $L'_e s$ proposal for epoch e. If $\mathcal{CC}_i \leq \mathcal{CC}_{L_e}$, party P_i forwards a code word $(s_{h,i}, w_{h,i})$ corresponding to the L_e's proposal for z_{v_h} if party P_i has received $(s_{h,i}, w_{h,i})$ either during the proposal dispersal phase or in earlier epochs. We note again that an ack-cert on accumulation value z_{v_h} (i.e., $\mathcal{AC}(z_{v_h})$) does not imply that all honest parties have received valid code words corresponding to value v_h during proposal dispersal phase. Thus, all honest parties may not forward their code word corresponding to value v_h in round 4.

In round 5, if party P_i receives $t+1$ valid code words for the accumulator z_h, it decodes value v_h using DEC function. Party P_i again encodes value v_h and sends code word $(s_{h,j}, w_{h,j})$ to party P_j $\forall j \in [n]$. In round 6, party P_j forwards the valid code word $(s_{h,j}, w_{h,j})$ to all parties if it has not already forwarded the code word $(s_{h,j}, w_{h,j})$ in round 4. Multicasting code words in 5 and forwarding codewords in rounds 5 and 6 ensures that if an honest party successfully decodes v_h, all honest parties will receive value v_h by the end of round 6.

Note that the elected leader could be Byzantine and that leader might not have sent valid code words to all honest parties during proposal dispersal phase and all honest parties may not have received valid code words corresponding to value v_h although an $\mathcal{AC}(z_{v_h})$ exists. Thus, it is possible that no honest party receives $t+1$ valid code words for accumulator z_{v_h} required to decode value v_h in round 5. In such a case, we ensure no honest party commits value v_h. In our protocol, we require that an honest party be able to decode value v_h in timely manner before voting for value v_h and later commit it. In particular, we rely on synchrony assumption to detect "bad" proposals and prevent it from getting committed.

Thus, party P_i votes for value v_h only if it decodes value v_h by round 5. Party P_i also checks if it did not detect equivocating proposals made by leader L_e in epoch e. This check ensures that if an honest party votes for a value v_h in round 7, all honest parties receive value v_h by round 7. In addition, party P_i also checks the proposed value is externally valid (i.e., $\mathsf{ex\text{-}validation}(v_h) = \mathsf{true}$) and the leader L_e is proposing with the highest ranked certificate. This ensures the safety of a committed value in earlier epochs.

An honest party P_i commits value v_h if it receives $t+1$ distinct votes for v_h. It multicasts the vote certificate and $\langle \mathsf{terminate}, e, H(v_h) \rangle$. In the next round, all honest parties will receive the vote certificate and not vote for lower ranked certificates in future epochs. In addition, if an honest party receives $t+1$ distinct $\langle \mathsf{terminate}, e, H(z_{v_h}) \rangle$ in a round, all honest parties receive the termination certificate, output value v_h and terminate in the next round.

Optimal communication complexity. Each party needs to learn ℓ bit input; thus, a protocol must incur $\Omega(n\ell)$ communication [56]. In Abraham et al. [2], they show $\Omega(n^2)$ communication is required even for a randomized Byzantine agreement protocol secure against a strongly adaptive adversary. Thus, our MVBA protocol has optimal communication complexity of $O(n\ell + \kappa n^2)$ in expectation.

Round complexity. In an epoch, an honest leader is elected with probability at least $\frac{1}{2}$. All honest parties commit and terminate in the same epoch when an honest leader is elected. Thus, the protocol terminates in expected 2 epochs. The proposal dispersal phase requires 2 rounds. Multicast of the termination certificate requires one more additional round. Thus, the protocol terminates in 19 rounds in expectation.

7.5.2 Security Analysis

Claim 86. *If an honest party votes for value v_h at round 7, then all honest parties receive value v_h by round 7.*

Proof. Suppose an honest party P_i votes for value v_h at round 7 in epoch e. Then party P_i must have decoded value v_h by round 5 and did not detect equivocating proposals by leader L_e by round 7. Party P_i must have sent valid code words and witness $\langle \mathsf{codeword}, \mathsf{mtype}, s_{h,k}, w_{h,k}, z_{v_h} \rangle_i$ computed from value v_h to every party P_k $\forall k \in [n]$ at round 5. The code words and witness arrive at all honest parties by round 6. In addition, no honest party detected an equivocating proposal by round 6 in epoch e.

Since no honest party detected an equivocating proposal by round 6 in epoch e., it must be that either honest parties will forward their code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_{h,k}, w_{h,k}, z_{v_h} \rangle$ when they receive the code words sent by party P_i or they already sent the corresponding code word in round 5 or received the code word from some other party. In any case, all honest parties will forward their code word corresponding to value v_h by round 6. Thus, all honest parties will have received $t+1$ valid code words for a common accumulation value z_{v_h} by round 7 sufficient to decode value v_h. □

Lemma 87. *If an honest party commits value v_h in epoch e, then (i) an equivocating certificate does not exist in epoch e, and (ii) all honest parties receive $\mathcal{C}_e(z_{v_h})$ by the end of epoch e.*

Proof. Suppose an honest party P_i commits value v_h in epoch e. Party P_i must have received at least $t+1$ vote messages for value v_h at round 8 in epoch e. At least one honest party (say party

P_j) must have voted for value v_h at round 7 in epoch e. Party P_j votes for value v_h when it receives value v_h by round 5, invokes Deliver for value v_h and does not detect any equivocating proposal by leader L_e by round 7. By Claim 86, all honest parties receive value v_h by round 7. Thus, no honest party votes for a conflicting value and an equivocating certificate does not exist in epoch e. This proves part (i) of the Lemma.

For part (ii), note that party P_i multicasts $\mathcal{C}_e(z_{v_h})$ when it commits value v_h in round 8. Thus, all honest parties receive $\mathcal{C}_e(z_{v_h})$ by end of round 8. By part (i) of the Lemma, an equivocating certificate does not exist. Thus, all honest parties will receive $\mathcal{C}_e(z_{v_h})$ by the end of epoch e. □

Theorem 88 (Safety)**.** *If two honest parties commit v and v', then $v = v'$.*

Proof. Suppose an honest party P_i commits value v in epoch e. By Lemma 87, all honest parties receive $\mathcal{C}_e(v)$ by the end of epoch e and no equivocating certificate exists in epoch e. Thus, no honest party votes for values other than v in any epoch $e' > e$ and an equivocating certificate cannot form in epochs higher than $e' > e$. Thus, it must be that if two honest party commits to v and v', then $v = v'$ □

Theorem 89 (Termination)**.** *If the leader L_e of epoch e is honest, all honest parties terminate by epoch e.*

Proof. Suppose the leader L_e of epoch e is honest. Leader L_e will send the same proposal $(z_{v_h}, \mathcal{AC}(z_{v_h}))$ to all parties by extending the highest ranked certificate known to all honest parties. Thus, each honest party P_i will forward valid code word (s_i, w_i) corresponding to value v_h to all parties in round 4 and all honest parties will receive $t + 1$ valid code words sufficient to decode value v_h in round 5. Thus, each honest party P_i will vote for value v_h in epoch 7, receive $\mathcal{C}_e(z_{v_h})$ by round 8 and commit v_h. In addition each honest party P_i will multicast $\langle \mathsf{terminate}, e, H(v_h)\rangle_i$, receive $t + 1$ distinct terminate, terminate by the end of round 8 of epoch e. □

Theorem 90. *The protocol in Figure 6.8 is a multi-valued validated Byzantine agreement protocol satisfying Definition 2.3.1*

Proof. For a value v_h to be decided at least one honest party must vote for it. For an honest party to vote for value v_h it must be that ex-validation(v_h) = true. The proofs for safety and termination follows immediately from Theorem 88 and Theorem 89. □

Lemma 91 (Communication Complexity)**.** *Let ℓ be the size of the input, κ be the size of accumulator, and w be the size of witness. The communication complexity of the MVBA protocol is $O(n\ell + (\kappa + w)n^2))$ in expectation.*

Proof. In the proposal dispersal phase, each party sends a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ to all other parties. In addition, each party sends κ-sized ack message to all other parties. Thus, this phase incurs $O(n\ell + (\kappa + w)n^2)$ communication.

In the protocol in Figure 6.8, the status step incurs $O(\kappa n^2)$ as each party sends $O(\kappa)$-sized threshold signature to all other parties. The propose step also incurs $O(\kappa n^2)$ communication. The leader election phase in round 3 incurs $O(\kappa n^2)$ communication. In the Forward step (round 4) each party multicasts code word of size $O(\ell/n)$, witness of size w bits, accumulator of size $O(\kappa)$ bits with a total communication complexity of $O(n\ell+(\kappa+w)n^2)$ bits. Similarly, in Decode step and Forward2, each party sends code word of size $O(\ell/n)$, witness of size w bits, accumulator of size $O(\kappa)$ bits with a total communication complexity of $O(n\ell + (\kappa + w)n^2)$ bits.

In Vote step, each party multicasts $O(\kappa)$-sized vote message to all other parties, this incurs $O(\kappa n^2)$ communication. In the commit step, each party multicasts $O(\kappa)$-sized vote certificate and $O(\kappa)$-sized terminate messages. All-to-all multicast of $O(\kappa)$-sized termination certificate also incurs $O(\kappa n^2)$ communication. Thus, the protocol incurs $O(n\ell+(\kappa+w)n^2))$ communication in an epoch.

Note that the protocol terminates in expected constant epochs. Thus, the communication complexity of the protocol is $O(n\ell + (\kappa + w)n^2))$ in expectation. □

7.6 Parallel Broadcast

Finally, we present two communication efficient parallel broadcast protocols tolerating $t < n/2$ Byzantine faults with a communication complexity of $O(n^2\ell + \kappa n^3)$ for input of size ℓ bits and expected $O(1)$ rounds under various setup assumptions. The first protocol is in the authenticated model with PKI and digital signatures. It is secure against a static adversary. The second protocol is secure against an adaptive adversary, but assumes threshold setup and uses adaptively-secure threshold signature scheme.

We present parallel broadcast protocols with expected constant rounds in Figure 7.5. In this protocol, each party P_i first uses graded parallel broadcast to propagate their input v_i and output a n-element list of values along with a n-element grade list $\mathsf{GradeList}_i$ accompanied by $\mathcal{AC}(\mathsf{GradeList}_i)$.

The tuple ($\mathsf{GradeList}_i$, $\mathcal{AC}(\mathsf{GradeList}_i)$) is then input to an MVBA protocol to agree on a common certified $\mathsf{GradeList}_h$. The ack certificate on $\mathsf{GradeList}$ servers as the external validity function. Parties then output $\mathcal{V}$ with $\mathcal{V}[j] = v_j$ if $\mathsf{GradeList}_h[j] \in \{3, 4\}$ $\forall j \in [n]$. We give two variants of the protocol depending upon the MVBA protocol being considered.

1. **Graded parallel broadcast.** Each party P_i invokes graded parallel broadcast protocol (refer Figure 7.2) with its input v_i and outputs an n element list of values along with ($\mathsf{GradeList}_i$, $\mathcal{AC}(\mathsf{GradeList}_i)$).
2. **MVBA.** Each party P_i participates in MVBA with input $\mathsf{GradeList}_i$ and $\mathcal{AC}(\mathsf{GradeList}_i)$. Let $\mathsf{GradeList}_h$ be the output of the MVBA protocol.
3. **Output.** Set $\mathcal{V}[j] = v_j$ if $\mathsf{GradeList}_h[j] \in \{3, 4\}$ $\forall j \in [n]$. Output $\mathcal{V}$.

Figure 7.5: Parallel broadcast with $O(n^2\ell + \kappa n^3)$ communication and expected $O(1)$ rounds

Using MVBA protocol from Chapter 6. In Chapter 6, we gave an MVBA protocol in the authenticated model with PKI and digital signatures with security against a static adversary. The MVBA protocol incurs $O(\kappa n^3)$ communication in expectation when $\ell = O(n)$ (i.e., the size of ($\mathsf{GradeList}, \mathcal{AC}(\mathsf{GradeList})$) and terminates in expected $O(1)$ rounds. Using this MVBA protocol, gives us a parallel broadcast protocol secure against static adversary in the authenticated model with PKI and digital signatures. The resulting parallel broadcast protocol has $O(n^2\ell) + E(O(\kappa n^3))$ communication and terminates in expected constant rounds.

Using MVBA from Section 7.5. In the second variant, we make use of our MVBA protocol from Section 7.5. Using our MVBA protocol, gives us a parallel broadcast protocol secure against a (strongly rushing) adaptive adversary. The graded parallel broadcast protocol has a communication complexity of $O(n^2\ell + \kappa n^3)$ and the MVBA protocol has a communication complexity of $O(\kappa n^2)$ when $\ell = O(n)$ (the size of ($\mathsf{GradeList}, \mathcal{AC}(\mathsf{GradeList})$). Thus, the resulting parallel broadcast protocol will have $O(n^2\ell + \kappa n^3) + E(O(\kappa n^2))$ communication and terminates in expected $O(1)$ rounds.

7.6.1 Security Analysis

Theorem 92. *The protocol in Figure 7.5 is a parallel broadcast protocol satisfying Definition 2.4.1.*

Proof. By Theorem 90, all honest parties eventually terminate with a common $\mathsf{GradeList}_h$ where external validity function is presence of $\mathcal{AC}(\mathsf{GradeList}_h)$. Termination follows from termination property of the underlying MVBA protocol.

By the properties of graded parallel broadcast(Theorem 84), all honest parties receive the same value v_j such that $\mathsf{GradeList}_h[j] \in \{3, 4\}$. Since, all honest parties compute final vector $\mathcal{V}$ based on common $\mathsf{GradeList}_h$. Thus, agreement holds.

In addition, the grades corresponding to honest parties in $\mathsf{GradeList}_h$ are in the range {3,4}. Thus, validity holds. □

7.7 Related Work

7.7.1 Related Works in Parallel Broadcast Literature

The problem of parallel broadcast (aka, interactive consistency) was originally introduced by Pease et al [91]. In the same work, they show two variants of the protocol (i) a protocol with $t < n/3$ resilience in the *plain* authenticated model or unauthenticated model, and (ii) a protocol with $t < n$ resilience in the authenticated model with authenticators. Both of their protocols had exponential communication complexity and $\Theta(t)$ round complexity.

Ben'or and El-Yaniv [18] showed how to achieve expected $O(1)$ rounds for the interactive consistency problem tolerating $t < n/3$ Byzantine faults in the plain authenticated model. In their solution, they invoked $O(n \log n)$ instances of the BA protocol due to Feldman and Micali [49] in a "black-box" fashion to achieve expected $O(1)$ round parallel broadcast protocol. Their construction has a very high communication as each instance of BA protocol of Feldman and Micali [49] has $O(n^6 \log n)$ communication (without q-SDH setup assumption) even for a single bit.

Very recently, Abraham et al. [1] gave an efficient protocol in the plain authenticated model tolerating $t < n/3$ Byzantine faults and security against an adaptive adversary. Their protocol incurs $O(n^2\ell + n^4 \log n)$ communication (without q-SDH setup assumption) in expectation for input of size ℓ bits and expected $O(1)$ rounds.

In the authenticated model with PKI and digital signatures, the notion of parallel broadcast was recently explored by Tsimos et al. [103]. They show two variants of the protocol each tolerating $t < (1 - \epsilon)n$ Byzantine faults and security against an adaptive adversary. The first protocol works in the authenticated model with PKI and digital signatures and incurs $\tilde{O}(\kappa^2 n^3)$ communication for single bit input and $O(t \log t)$ rounds. Their second protocol has stronger setup assumptions. In particular, they require a trusted dealer to setup the keys and relies on *bit-specific* committee election [2] to reduce communication. In addition, their protocol requires parties to erase their

signatures once a message has been sent. Their protocol incurs $\tilde{O}(\kappa^4 n^2)$ communication for single bit input and $O(\kappa \log t)$ rounds.

Closely related technique. In a recent work [1], Abraham et al. gave a parallel broadcast protocol in the unauthenticated model tolerating $t < n/3$ Byzantine faults with a communication complexity of $O(n^2\ell) + E(O(n^4 \log n))$ and termination in expected $O(1)$ rounds. Their protocol relies on the idea of Fitzi and Garay [55] where multiple BA sub-protocols are run in parallel when only a single leader election is invoked per iteration for all the sub-protocols. In their construction, each party first propagates their ℓ bit input via a gradecast protocol where each gradecast invocation costs $O(n\ell + n^3 \log n)$; the total communication complexity of n parallel gradecast is $O(n^2\ell + n^4 \log n)$. It is followed by parallel invocation of n instances of BA protocol where each BA protocol has a communication complexity of $O(n^3 \log n)$ bits for a single bit input. In addition, their leader election protocol has a communication complexity of $O(n^4 \log n)$ bits. The resulting protocol has a communication complexity of $O(n^2\ell) + E(O(n^4 \log n))$ for input of size ℓ bits.

We note that their technique is relevant but not sufficient to achieve our goal. In the authenticated model with PKI and digital signatures, to the best of our knowledge, the MVBA protocol from Chapter 6, when used as a BA protocol, is the most efficient protocol in the setting which has a communication complexity of $O(\kappa n^3)$ in expectation and termination in expected constant rounds. Parallel invocation of $O(n)$ instances of this BA protocol would result in $O(\kappa n^4)$ communication in each round. In contrast, our parallel broadcast in the setting incurs $O(n^2\ell) + E(O(\kappa n^3))$ communication.

With threshold setup assumption, to the best of our knowledge, the BA protocol due to Abraham et al. [4] is the most efficient protocol which has a communication complexity of $O(\kappa n^2)$ in expectation and termination in expected constant rounds. Following the technique of Abraham et al. [1], we can use $\mathsf{M\text{-}Gradecast}(., 2)$ to propagate ℓ bit input at the total communication complexity of $O(n^2\ell + \kappa n^3)$. Then, parallel invocation of $O(n)$ instances of binary BA protocol due to Abraham et al. [4] along with a single leader election protocol across all BA instances will result in expected $O(\kappa n^3)$ communication and termination in expected constant rounds. The total communication complexity of the protocol following their technique is $O(n^2\ell) + E(O(\kappa n^3))$ and termination in expected constant rounds. In the same setting, our protocol incurs $O(n^2\ell + \kappa n^3) + E(O(\kappa n^2))$ communication and expected constant rounds. In the worst case, when the protocol runs for linear number of rounds, the protocol following their technique would incur $O(n^2\ell + \kappa n^4)$ communication, while our protocol incurs $O(n^2\ell + \kappa n^3)$ communication.

7.7.2 Related Works in MVBA Literature

Multi-valued validated Byzantine agreement was first introduced by Cachin et al. [28] to allow honest parties to agree on any externally valid values. Their protocol works in asynchronous communication model and has optimal $t < n/3$ resilience with $O(n^2\ell + \kappa n^2 + n^3)$ communication for input of size ℓ. Later, Abraham et al. [8] gave an MVBA protocol with optimal resilience and $O(n^2\ell + \kappa n^2)$ communication in the same asynchronous setting. Lu et al. [79] extended the work of Abraham et al. [8] to handle long messages of size ℓ with a communication complexity of $O(n\ell + \kappa n^2)$. All of these protocols assume threshold setup, are secure against an adaptive adversary and terminate in expected $O(1)$ rounds. We provide technical differences with MVBA protocol of Lu et al. [79].

Comparison with MVBA protocol of Lu et al. [79]. In the MVBA protocol due to Lu et al. [79], they use $(t+1, n)$ RS codes with $t < n/3$ to distribute ℓ bit proposal during proposal dispersal phase. In their protocol, they collect an ack-cert consisting of $2t+1$ ack messages. If there is an ack-cert for a proposal, this implies at least $t+1$ honest parties have received valid code words for the proposal. This is sufficient to decode the proposal since the protocol uses $(t+1, n)$ RS codes. Thus, in their protocol, ack-cert for a proposal implies honest parties have sufficient valid code words to decode the original proposal and honest parties can agree on any proposal with an ack-cert. This is in contrast to our protocol since honest parties may not be able to decode the proposal even though the proposal is accompanied by an ack-cert. Our protocol relies on synchrony to filter out such "bad" proposals.

Comparison with MVBA protocol from Chapter 6. To the best of our knowledge, the MVBA protocol from Chapter 6 is the first MVBA protocol in the synchronous model tolerating $t < n/2$ Byzantine faults secure against static adversary. The protocol from Chapter 6 works in the plain PKI model without threshold setup and incurs $O(n^2\ell + \kappa n^3)$ for inputs of size ℓ and expected constant rounds. In this chapter, we present an MVBA protocol with better communication and security against a strongly rushing adaptive adversary. Our protocol relies on threshold setup assumption and incurs $O(n\ell + \kappa n^2)$ for inputs of size ℓ bits and terminates in expected $O(1)$ rounds.

Chapter 8

Conclusion and Future Work

This thesis focused on improving the communication complexity and the round complexity of several synchronous Byzantine consensus primitives under various settings. Chapter 3 studied latency of optimistically responsive synchronous consensus protocols and presented consensus protocols that commits with the best-possible latency under all conditions. We also provided a prototype implementation along with evaluation results. Chapters 4 and 5 studied the communication complexity of consensus protocols in the absence of threshold setup. In this setting, we presented two new efficient consensus protocols that incur quadratic communication per decision and optimistically responsive latency during optimistic conditions.

Chapter 6 presented a new framework to solve the DKG problem and provided two new protocols following this framework. Both of these protocols incur cubic communication with either expected constant rounds or linear round complexity. Finally, Chapter 7 studied the communication complexity and the round complexity of parallel broadcast. We showed a generic reduction from parallel broadcast to graded parallel broadcast and validated Byzantine consensus. Using this reduction, we presented two parallel broadcast protocols with cubic communication and expected $O(1)$ rounds.

Future work. The lower bound on the communication complexity to solve the DKG problem still remains an open research question. It is an interesting direction to either show a cubic communication lower bound on the DKG problem or to obtain a protocol with subcubic communication. Similarly, Chapter 7 presented efficient protocols for parallel broadcast when the fault tolerance is $t < n/2$. It is an interesting direction to design efficient protocols for parallel broadcast when the fault tolerance is $t \geq n/2$.

Bibliography

[1] Ittai Abraham, Gilad Asharov, Shravani Patil, and Arpita Patra. Asymptotically free broadcast in constant expected time via packed vss. In *Theory of Cryptography: 20th International Conference, TCC 2022, Chicago, IL, USA, November 7–10, 2022, Proceedings, Part I*, pages 384–414. Springer, 2023.

[2] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of Byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 317–326, 2019.

[3] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous Byzantine consensus. *arXiv preprint arXiv:1704.02397*, 2017.

[4] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. In *International Conference on Financial Cryptography and Data Security*, pages 320–334. Springer, 2019.

[5] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. *arXiv preprint arXiv:2102.09041*, 2021.

[6] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *IACR Cryptol. ePrint Arch.*, 2018:1153, 2018.

[7] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 654–667, 2020.

[8] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

[9] Ittai Abraham, Kartik Nayak, Ling Ren, and Zhuolun Xiang. Good-case latency of Byzantine broadcast: A complete categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, PODC'21, page 331–341, New York, NY, USA, 2021. Association for Computing Machinery.

[10] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous bft. In *25th International Conference on Principles of Distributed Systems*, 2022.

[11] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Communication and round efficient parallel broadcast protocols. *Cryptology ePrint Archive*, 2023.

[12] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires t+ 1 rounds. *Information Processing Letters*, 71(3-4):155–158, 1999.

[13] Renas Bacho and Julian Loss. On the adaptive security of the threshold bls signature scheme. In *CCS'2022*, pages 193–207, 2022.

[14] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 590–609. Springer, 2011.

[15] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.

[16] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round mpc with identifiable abort and public verifiability. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II*, pages 562–592. Springer, 2020.

[17] Michael Ben-Or. Another advantage of free choice (extended abstract) completely asynchronous agreement protocols. In *Proceedings of the second annual ACM symposium on Principles of distributed computing*, pages 27–30, 1983.

[18] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[19] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*, pages 351–371. 2019.

[20] Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science*, pages 313–321. Springer, 1992.

[21] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 3502–3524, 2021.

[22] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. OptRand: Optimistically responsive distributed random beacons. In *Proceedings of the 30th Network and Distributed System Security Symposium (NDSS)*, 2023.

[23] Manuel Blum. Coin flipping by telephone a protocol for solving impossible problems. *ACM SIGACT News*, 15(1):23–27, 1983.

[24] Alexandra Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *International Workshop on Public Key Cryptography*, pages 31–46. Springer, 2003.

[25] Dan Boneh and Xavier Boyen. Short signatures without random oracles and the sdh assumption in bilinear groups. *Journal of cryptology*, 21(2):149–177, 2008.

[26] Gabriel Bracha. Asynchronous Byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[27] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains.* PhD thesis, University of Guelph, 2016.

[28] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[29] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, 2005.

[30] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Annual International Cryptology Conference*, pages 98–116, 1999.

[31] Ignacio Cascudo and Bernardo David. Scrape: Scalable randomness attested by public entities. In *ACNS*, pages 537–556. Springer, 2017.

[32] Miguel Castro, Barbara Liskov, et al. Practical Byzantine Fault Tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[33] Miguel Castro, Rodrigo Rodrigues, and Barbara Liskov. Base: Using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems (TOCS)*, 21(3):236–269, 2003.

[34] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pala: A simple partially synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:981, 2018.

[35] T-H Hubert Chan, Rafael Pass, and Elaine Shi. Pili: An extremely simple synchronous blockchain. *IACR Cryptology ePrint Archive*, 2018:980, 2018.

[36] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. Spurt: Scalable distributed randomness beacon with transparent setup. Technical report, Cryptology ePrint Archive, Report 2021/100. https://eprint. iacr. org/2021/100, 2021.

[37] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive, Paper 2022/1389, 2022.

[38] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 2518–2534. IEEE, 2022.

[39] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Advances in Cryptology, 9th Annual International Cryptology Conference*, volume 435, pages 307–315, 1989.

[40] Danny Dolev, Joseph Y Halpern, Barbara Simons, and Ray Strong. Dynamic fault-tolerant clock synchronization. *Journal of the ACM (JACM)*, 42(1):143–185, 1995.

[41] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for Byzantine agreement. *Journal of the ACM (JACM)*, 32(1):191–204, 1985.

[42] Danny Dolev, Ruediger Reischuk, and H Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM (JACM)*, 37(4):720–741, 1990.

[43] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[44] Drand. Drand - a distributed randomness beacon daemon.

[45] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM)*, 35(2):288–323, 1988.

[46] Paolo D'Arco and Douglas R Stinson. On unconditionally secure robust distributed key distribution centers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 346–363, 2002.

[47] Andreas Erwig, Sebastian Faust, and Siavash Riahi. Large-scale non-interactive threshold cryptosystems through anonymity. *IACR Cryptology ePrint Archive*, 2021:1290, 2021.

[48] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[49] Paul Feldman and Silvio Micali. Optimal algorithms for Byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 148–161, 1988.

[50] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous Byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[51] Michael J Fischer and Nancy A Lynch. A lower bound for the time to assure interactive consistency. Technical report, GEORGIA INST OF TECH ATLANTA SCHOOL OF INFORMATION AND COMPUTER SCIENCE, 1981.

[52] Michael J Fischer, Nancy A Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

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

[54] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement.* PhD thesis, ETH Zurich, 2002.

[55] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *Proceedings of the twenty-second annual symposium on Principles of distributed computing*, pages 211–220, 2003.

[56] Matthias Fitzi and Martin Hirt. Optimally efficient multi-valued Byzantine agreement. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 163–168, 2006.

[57] Juan A Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS'07)*, pages 658–668. IEEE, 2007.

[58] Juan A Garay and Yoram Moses. Fully polynomial Byzantine agreement for n¿ 3 t processors in t+ 1 rounds. *SIAM Journal on Computing*, 27(1):247–290, 1998.

[59] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[60] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.

[61] Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, 2021:339, 2021.

[62] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. Sbft: a scalable decentralized trust infrastructure for blockchains. *arXiv preprint arXiv:1804.01626*, 2018.

[63] Yue Guo, Rafael Pass, and Elaine Shi. Synchronous, with a chance of partition tolerance. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*, pages 499–529. Springer, 2019.

[64] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 147–176. Springer, 2021.

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

[66] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 322–340. Springer, 2005.

[67] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. *Proceedings of Foundations of Computer Security—FCS*, 4:117–130, 2004.

[68] Aniket Kate and Ian Goldberg. Distributed key generation for the internet. In *29th IEEE International Conference on Distributed Computing Systems*, pages 119–128, 2009.

[69] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *IACR Cryptol. ePrint Arch.*, 2012:377, 2012.

[70] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*, pages 177–194. Springer, 2010.

[71] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[72] Justin Kim, Vandan Mehta, Kartik Nayak, and Nibesh Shrestha. Brief announcement: Making synchronous bft protocols secure in the presence of mobile sluggish faults. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 375–377, 2021.

[73] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1751–1767, 2020.

[74] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative Byzantine fault tolerance. *ACM SIGOPS Operating Systems Review*, 41(6):45–58, 2007.

[75] Jae Kwon. Tendermint: Consensus without mining. *Draft v. 0.6, fall*, 1(11), 2014.

[76] Torus Lab. Torus: Globally accessible public key infrastructure for everyone. `https://tor.us/`, 2021.

[77] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[78] Julian Loss and Tal Moran. Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. *Cryptology ePrint Archive*, 2018.

[79] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous Byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, pages 129–138, 2020.

[80] J-P Martin and Lorenzo Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215, 2006.

[81] Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

[82] Silvio Micali. Byzantine agreement, made trivial, 2016.

[83] Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-bft: Optimistically responsive synchronous consensus with optimal latency or resilience. *Cryptology ePrint Archive*, 2020.

[84] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In *35th International Symposium on Distributed Computing*, 2021.

[85] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260, 2008.

[86] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for Byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[87] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and communication networks*, 9(17):4585–4595, 2016.

[88] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Cryptographers' track at the RSA conference*, pages 275–292. Springer, 2005.

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

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

[91] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[93] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *EUROCRYPT'91*, page 522–526, 1991.

[94] Michael O Rabin. Randomized Byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409. IEEE, 1983.

[95] Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

[96] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R Weippl. Ethdkg: Distributed key generation with ethereum smart contracts. *IACR Cryptol. ePrint Arch.*, 2019:985, 2019.

[97] Fred B Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[98] Elaine Shi. Streamlined blockchains: A simple and elegant approach (a tutorial and survey). In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 3–17. Springer, 2019.

[99] Victor Shoup. Practical threshold signatures. In *EUROCRYPT 2000*, volume 1807, pages 207–220. Springer, 2000.

[100] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[101] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. Synchronous distributed key generation without broadcasts. *Cryptology ePrint Archive*, 2021.

[102] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE, 2020.

[103] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. Gossiping for communication-efficient broadcast. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part III*, pages 439–469. Springer, 2022.

[104] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.