Improving Performance of Consensus Protocols

by

Jun Wan

B.S. Tsinghua University (2016)
S.M. Massachusetts Institute of Technology (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2024

Authored by:	Jun Wan Department of Electrical Engineering and Computer Science January 18, 2024
Certified by:	Srinivas Devadas Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Committee on Graduate Students

Improving Performance of Consensus Protocols

by

Jun Wan

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 2024 in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

ABSTRACT

Designing an efficient solution for Byzantine Broadcast (BB) is a central problem for many distributed computing and cryptographic tasks. Some of the most important challenges include improving the round complexity, the communication complexity of the protocol and tolerating strong adversaries.

For round complexity, under the honest majority setting, it is long known that there exist randomized protocols that can achieve BB in expected constant rounds, regardless of the number of nodes n. However, whether we can match the expected constant round complexity in the corrupt majority setting — or more precisely, when $f \geq n/2 + \omega(1)$ — was unknown, where f denotes the number of corrupt nodes. We solve this long-standing question and achieve BB in expected constant rounds, even when 99% of the nodes are corrupted by a weakly adaptive adversary. A weakly adaptive adversary can observe messages sent by honest nodes, adaptively corrupt nodes and inject arbitrary new messages.

Besides a weakly adaptive adversary, it is also important to study the round complexity of BB protocol under a strongly adaptive adversary. A strongly adaptive adversary can examine the original message an honest node would have wanted to send in some round, adaptively corrupt the node in the same round and make it send a completely different message instead. In the corrupt majority setting, no protocol with sublinear round complexity is known. We are the first to construct a BB protocol with sublinear round complexity. Specifically, assuming the existence of time-lock puzzles with suitable hardness parameters and that the decisional linear assumption holds in suitable bilinear groups, we show how to achieve BB in $(\frac{n}{n-f})^2 \cdot \mathsf{poly}\log\lambda$ rounds with $1 - \mathsf{negl}(\lambda)$ probability, where λ is the security parameter.

Another important metric for a BB protocol is the communication complexity. There have been many attempts to achieve sub-quadratic complexity in several directions, both in theory and practice, all with pros and cons. We initiate the study of another attempt: improving the *amortized* communication complexity over a significant long sequence of Byzantine Broadcast executions. We achieve optimal amortized linear complexity under honest majority and amortized quadratic communication complexity under dishonest majority and a strongly adaptive adversary.

Thesis supervisor: Srinivas Devadas
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

I extend my deepest gratitude to my advisor, Srini Devadas. Srini encouraged me to work on projects that interested me. Thanks to Srini's unwavering support, I was able to enjoy unlimited academic freedom and always have fun with my research projects. If I am to describe Srini in one word, I cannot think of a better option than "passionate". His passion for research and attention to detail have a great impact on me, both academically and personally.

My heartfelt thanks also go to my committee members, Elaine Shi and Nancy Lynch. When we struggled to publish our first consensus paper, Elaine taught me the skill to effectively communicate research findings to the consensus research community. Over the years, Elaine generously shared her expertise and insights in consensus and cryptography, which has been invaluable to me. Nancy, one of the pioneers in consensus, offered essential advice for my RQE and thesis, for which I am grateful.

I would also like to thank Ling Ren, who led me into the consensus area. As a senior colleague, Ling's mentorship in my early years of Ph.D. shaped my understanding of consensus protocols. The discussions we shared benefited me greatly.

I am fortunate to have collaborated with Srini Devadas, Elaine Shi, Ling Ren, Zhuolun Xiang, Atsuki Momose, and Hanshen Xiao. This thesis would not have been possible without their collective wisdom and collaboration. Zhuolun and Atsuki enlightened me with their sharp insights into the communication complexities of consensus protocols. Collaborating with Hanshen, on research projects both within and outside of consensus area, has been a remarkable experience. His ability to analyze research problems from a high level and capture the key insights never ceases to impress me.

A warm thank you to all members of the Hornet research group. More than just labmates, you have been wonderful friends. Celebrating my 21st birthday with the group is a memory I cherish. Sharing these years with you all has been a joy.

Finally, I am deeply grateful to my family and friends. The friends I met at MIT have made this academic journey enjoyable and memorable. My parents and sister's support has been my backbone throughout this journey. Their role in my life and this achievement is immeasurable.

Contents

Chapter 1

Introduction

Byzantine Agreement (BA) is one of the most fundamental problems in fault tolerant distributed computing [1]–[3] and of increasing interest given recent advances in cryptocurrencies [4]–[6]. We consider the "broadcast" formulation of Byzantine Agreement, henceforth called Byzantine Broadcast (BB): imagine that there are n nodes among which there is a designated sender. The sender is given an input bit $b \in \{0, 1\}$ and wants to send this bit to every other node. Although up to $f < n - 1$ nodes can be corrupted and deviate arbitrarily from the prescribed protocol, we would like to ensure two key properties: 1) *consistency* requires that all honest nodes must output the same bit (even when the sender is corrupt); and 2) *validity* requires that all honest nodes output the sender's input bit if the sender is honest [1].

Two of the most important metrics for a BB protocol are the *round complexity* and *communication complexity*. The round complexity indicates how long it takes for honest nodes to agree and the communication complexity represents the number of bits sent by honest nodes during the protocol execution. The round and communication complexity depend heavily on our adversarial settings / assumptions, including the adversary model and the ratio of the corrupt users.

There are three common types of adversary models, namely *static adversary*, *weakly adaptive advesrary* and *strongly adaptive adversary*. A static adversary corrupts up to f nodes at the beginning of the protocol and cannot corrupt any honest user once the protocol begins. An adaptive adversary, on the other hand, can corrupt nodes during the protocol as long as the total number of corrupt nodes is bounded by f. Once a node is corrupt, it remains corrupt throughout the entire protocol. Adaptive adversaries are further divided into weakly adaptive and strongly adaptive. When a weakly adaptive adversary corrupts a node v in some round r, it can make the now-corrupt v inject additional messages. However, it cannot erase the message v has already sent in round r. In contrast, a strongly adaptive adversary can both erase messages already sent and inject additional messages.

In this thesis, we show that we can achieve

1. expected constant round complexity under a weakly adaptive adversary and $f = (1 - \varepsilon)n$ for any constant $\varepsilon > 0$.

[1] An alternative formulation, known as Byzantine Agreement, has a different definition for validity. In Byzantine Agreement, each node receives an input bit. If all honest nodes receive the same input bit b, then they must output b. However, Byzantine Agreement is known to be impossible under corrupt majority.

2. polylogarithmical round complexity under a strongly adaptive adversary and $f = (1-\varepsilon)n$ for any constant $\varepsilon > 0$.

3. linear amortized communication complexity under a strongly adaptive adversary and honest majority.

Throughout the thesis, we assume a synchronous network, i.e., honest nodes can deliver messages to each other within a single round. We also assume a trusted setup. Due to the famous lower bound by Lamport et al. [7], some setup assumption is necessary to get consensus under $f \geq n/3$ (even static) corruptions. We first discuss the previous works on round complexity and communication complexity in Sections 1.1 and 1.2, respectively. In each section, we also compare with previous results and provide intepretation of our results.

1.1 Round Complexity

1.1.1 Previous works

The round complexity of Byzantine Broadcast has been studied for decades. Fischer and Lynch [8] showed that any deterministic consensus protocol with f fault tolerance requires at least $f+1$ rounds. Later, Dolev and Strong [9] showed that this is true even assuming (idealized) digital signatures. It is widely understood that *randomization* can help overcome the $(f+1)$-round barrier under the honest majority setting. Specifically, many elegant works have shown expected constant-round protocols assuming honest majority [10]–[13].

For a long while, the community was perplexed about the following natural question: *can we achieve sublinear-round Byzantine Broadcast under dishonest majority*? Garay et al. [14] was the first to demonstrate a positive result although their construction achieves sublinear round complexity only under a narrow parameter regime: specifically, they constructed an expected $\Theta((f-n/2)^2)$-round protocol, and the subsequent work of Fitzi and Nielsen [15] improved it to $\Theta(f-n/2)$ rounds. In other words, these constructions achieve sublinear number of rounds only if $f \leq n/2 + o(n)$. This is somewhat unsatisfying since even for $f = 0.51n$, their results would be inapplicable.

Recently, the frontier of our understanding was again pushed forward by Chan, Pass, and Shi [16]. Assuming trusted setup and standard cryptographic assumptions, their protocol achieves Byzantine Broadcast with probability $1-\delta$ for any $f \leq (1-\epsilon)\cdot n$ in $\mathsf{poly}\log(1/\epsilon, \delta)$ rounds (both in expectation and worst-case), where $\epsilon, \delta \in (0,1)$ are two parameters that the protocol takes as input. Although their work represents exciting progress on a long stagnant front, it remains somewhat unsatisfying. Firstly, it fails to match the asymptotic (expected) round complexity of known honest majority protocols — for honest majority, it is long known how to achieve *expected constant* round complexity [10], [13]. Secondly, to achieve their result, we had to significantly weaken the adversary's capabilities relative to the prior results in this space. All aforementioned works [17]–[19] prior to Chan et al. [16] secured against a *strongly adaptive* adversary. In fact, the strongly adaptive model was the well-accepted model in the early days of distributed consensus and multi-party protocols. We thus ask the following two important questions:

- Can we achieve Byzantine Broadcast in *expected constant* rounds in the corrupt majority setting?

- Are there (randomized) BB protocols with sublinear round complexity, and secure in the presence of a strongly adaptive adversary that is allowed to corrupt a majority of the nodes?

1.1.2 Our contribution

We give an affirmative answer to both questions and present two different solutions that achieve

- expected constant round complexity under a weakly adaptive adversary, and
- sublinear round complexity under a strongly adaptive adversary,

for $f = (1-\epsilon)n$ where $\epsilon \in (0,1)$ may be an arbitrarily small constant. We describe our results in detail in the following two theorems.

Theorem 1.1.1 (Expected constant round BB under weakly adaptive corruption)**.** *Assuming the existence of a trusted setup and the decisional linear assumption in suitable bilinear groups* [2]*, there exists a BB protocol with expected $O((\frac{n}{n-f})^2)$ round complexity for any non-uniform $p.p.t.$ weakly adaptive adversary that can corrupt $f < n-1$ nodes.*

Theorem 1.1.2 (Sublinear round BB under strongly adaptive corruption)**.** *Assuming the existence of a trusted setup, the decisional linear assumption in suitable bilinear groups, as well as the existence of time-lock puzzles [21] with hardness parameter ξ, there exists a protocol that achieves BB in $(\frac{n}{n-f})^2 \cdot \frac{\mathsf{poly}\log\lambda}{\xi}$ number of rounds with probability $1-\mathsf{negl}(\lambda)$ under a strongly adaptive adversary. Here, $\mathsf{negl}(\cdot)$ is a suitable negligible function (of the security parameter λ).*

We compare our results with the state-of-the-art results in Table 1.1 and situate our result in context to help the reader understand how tight the bound is as well as the assumptions we make.

	Garay et al.[18]	Fitzi et al.[15]	Chan et al.[16]	Our result 1	Our result 2
Expected round complexity	$\Theta((2f-n)^2)$	$\Theta(2f-n)$	Same as worst-case	$\Theta((\frac{n}{n-f})^2)$	Same as worst-case
Worst-case round complexity with $1-\delta$ failure probability	$\Theta(\log(\frac{1}{\delta}) + (2f-n)^2)$	$\Theta(\log(\frac{1}{\delta}) + (2f-n))$	$\Theta(\log(\frac{1}{\delta}) \cdot \frac{n}{n-f})$	$\Theta(\frac{\log(1/\delta)}{\log(n/f)} \cdot \frac{n}{n-f})$	$(\frac{n}{n-f})^2 \cdot \frac{\mathsf{poly}\log\lambda}{\xi}$
Adversary Type	Strongly Adaptive	Strongly Adaptive	Weakly Adaptive	Weakly Adaptive	Strongly Adaptive

Table 1.1: A comparison between our results and previous work under dishonest majority.

Result under weakly adaptive adversary (Theorem 1.1.1): To the best of our knowledge, our work is the first to achieve an expected constant-round BB protocol for any $f \geq n/2 + \omega(1)$. Previously, no result was known even for the *static* corruption setting, and even under any setup assumptions. Theorem 1.1.1 says that if the number of honest nodes is an arbitrarily small constant fraction (e.g., 0.01%), we can achieve expected constant rounds. The restriction on the number of honest nodes is necessary in light of an elegant lower bound proven by Garay et al. [18]: they showed that even randomized protocols cannot achieve BB in less than $\Theta(n/(n-f))$ number of

[2]The reader can refer to Groth et al. [20] for the definition.

rounds, *even assuming static corruption* and allowing reasonable setup assumptions. Note that their lower bound says that when almost all nodes can be corrupt except $O(1)$ nodes who remain honest, then even randomized protocols must incur linear number of rounds. Comparing their lower bound and our upper bound side by side, one can see that for the (narrow) regime $n - f = o(n)$, there is still an asymptotical gap between our upper bound and their lower bound. Whether we can construct an upper bound that matches their lower bound in this regime remains open.

We stress, however, that expected constant-round BB under 51% corruption is an open question whose answer has eluded the community for more than three decades, under any assumption, allowing any (reasonable) setup, and even under static corruption. We therefore believe that despite our trusted setup and weakly adaptive restrictions, our result is an important step forward in this line of work.

Result under strongly adaptive adversary (Theorem 1.1.2): The protocol requires a time-lock puzzle. A time-lock puzzle with hardness parameter ξ ensures that the puzzle solution remains hidden from any machine running in time that is at most ξ fraction of the honest evaluation time, even when the machine has access to unbounded polynomial parallelism. As a typical example, consider the case when $\xi \in (0, 1)$ is a constant just like what prior works have assumed [21], [22], and moreover, suppose that $\frac{n}{n-f}$ is also a constant (e.g., 99% may be corrupt) — in this case, our protocol's round complexity is simply poly $\log \lambda$.

To the best of our knowledge, no prior work can achieve sublinear-round BB in the strongly adaptive setting under any reasonable setup assumption, and even for only 51% corruption. In this sense our result significantly improves our understanding of the round complexity of BB.

1.2 Communication Complexity

For the communication complexity, Dolev and Reischuk [23] showed any deterministic Byzantine broadcast protocol costs at least $\Omega(f^2)$ messages in the worst case, which was matched by Berman et al. [24] for $f < n/3$ and then by Momose and Ren [25] for $f < n/2$. However, for large-scale distributed systems, $\Theta(n^2)$ communication complexity may still be expensive. Many attempts have been made to circumvent the impossibility results, mainly in three directions – (1) Randomized solutions [26]–[29], which improve the communication cost to (worst-case or expected) sub-quadratic. But this approach is inherently vulnerable to *strongly adaptive adversaries* [26]. (2) Optimistic solutions [30]–[35] that have sub-quadratic communication under optimistic executions such as synchrony and no failures. But they still incur quadratic costs in the worst case. (3) Extension protocols [36]–[38] can achieve the optimal communication cost $O(nL)$ for an input of sufficient size L. Thus, they can also reduce the cost of multiple parallel broadcasts (or atomic broadcasts [39]) through batching. But they do not allow sequential and causal broadcast invocations (i.e., a decision in an instance may affect the input of the next instance), which is required in many cryptographic protocols assuming *broadcast channel* [40]–[42].

Despite all the efforts in the above three directions, another natural attempt through amortization across multiple (sequential) instances, is somehow overlooked in the literature. In the thesis, we initialize the formal study of the amortized communication complexity of multi-shot Byzantine broadcast. Multi-shot BB consists of sequential broadcasts with clear boundaries (i.e., one instance ends before the next instance starts) performed by possibly different senders. The amortized communication cost of a multi-shot Byzantine broadcast protocol is measured as the average cost of each broadcast instance if the protocol runs sufficiently long. Due to the ever-growing nature of targeted applications such as blockchains, reducing the amortized cost of multi-shot Byzantine broadcast can significantly improve the performance of a long-running system. More formally, supposing the total communication complexity in bits of the multi-shot protocol is $C(L, n, f)$ after L sequential instances of Byzantine broadcast, our goal is to design multi-shot Byzantine broadcast protocols that can minimize

$$\lim_{L\to\infty} \frac{C(L, n, f)}{L}.$$

Our goal is to design amortized linear multi-shot Byzantine broadcast protocols in a synchronous network under honest majority and strongly adaptive adversaries, circumventing the $\Omega(f^2)$ lower bound. We also extend our technique to the dishonest majority case to achieve quadratic amortized cost.

- Assuming a threshold signature scheme, we aim to achieve $O(\kappa n)$ (κ is a security parameter) amortized communication cost for synchronous multi-shot Byzantine broadcast under $f \leq (1/2 - \varepsilon)n$ for any positive constant ε.
- Assuming a digital signature scheme, we aim to achieve $O(\kappa n^2)$ amortized communication cost for synchronous multi-shot Byzantine broadcast under $f < n$.

Protocol	Network Model	Fault Tolerance	Total Cost (L decisions)	Amortized Cost	Cryptographic Primitives
Berman et al. [24]	synchrony	$f < n/3$	$O(n^2 L)$	$O(n^2)$	None
Momose-Ren [25]	synchrony	$f < n/2$	$O(\kappa n^2 L)$	$O(\kappa n^2)$	threshold sig.
Momose-Ren [25]	synchrony	$f \leq (1/2 - \epsilon)n$	$O(\kappa n^2 L)$	$O(\kappa n^2)$	signature
Our result 3	synchrony	$f \leq (1/2 - \epsilon)n$	$O(\kappa n L + \kappa n^3)$	$O(\kappa n)$	threshold sig.
Dolev-Strong [43]	synchrony	$f < n$	$O((\kappa n^2 + n^3)L)$	$O(\kappa n^2 + n^3)$	multi-sig *
Our result 3	synchrony	$f < n$	$O(\kappa n^2 L + \kappa n^4)$	$O(\kappa n^2)$	signature

Table 1.2: Comparison with existing solutions for multi-shot BB with constant-sized inputs under strongly adaptive adversaries. The original Dolev-Strong broadcast uses signatures and has cost $O(\kappa n^3)$.

1.3 Related Papers

The thesis consists of works from the following three papers:

1. Expected Constant Round Byzantine Broadcast under Dishonest Majority. Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Theory of Cryptography Conference (TCC), 2020.
2. Round-Efficient Byzantine Broadcast under Strongly Adaptive and Majority Corruptions. Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Theory of Cryptography Conference (TCC), 2020.

3. On the Amortized Communication Complexity of Byzantine Broadcast. Jun Wan, Atsuki Momose, Ling Ren, Elaine Shi and Zhuolun Xiang. ACM Symposium on Principles of Distributed Computing (PODC), 2023.

Chapter 2

Technical Roadmap

2.1 Preliminaries

2.1.1 Problem definition.

The problem of Byzantine Broadcast has been widely explored. Suppose there are n *nodes* (sometimes also called *parties*) in a distributed system, indexed from 1 to n, respectively. The communication within the system is modeled by a synchronous network, where a message sent by an honest node in some round r is guaranteed to be delivered to an honest recipient at the beginning of the next round $r + 1$. Among the n nodes in the system, there is a designated sender whose identity is common knowledge. Before the protocol begins, the sender receives an input bit b. All nodes then engage in interactions where the sender aims to send the bit b to everyone. At the end of the protocol, each node u outputs a bit b_u.

Definition 2.1.1 (Single-shot Byzantine Broadcast (BB))**.** A Byzantine broadcast protocol for a set of n nodes with a designated sender with input bit b invoking $\mathsf{bc}(b)$, must satisfy the following properties.

- Consistency. If two honest nodes output b_u and b_v respectively, then $b_u = b_v$.
- Termination. All honest nodes output and terminate.
- Validity. If the designated sender is honest and invokes $\mathsf{bc}(b)$, then all honest nodes output b and terminate.

Although our main definition is for agreeing on a single bit, our approach easily extends to multi-valued BB too. When we measure the communication complexity, we are also interested in the problem of *multi-shot Byzantine broadcast*, which consists of a sequence of single-shot *Byzantine broadcasts* [43].

Definition 2.1.2 (Multi-shot Byzantine Broadcast)**.** A multi-shot Byzantine broadcast protocol for a set of n nodes with a (possibly different) designated sender S_i for each slot $i > 0$ with input bit b invoking $\mathsf{bc}_i(b)$, must satisfy the following properties.

- Consistency. If two honest nodes commit values b_u and b_v respectively at the same slot, then $b_u = b_v$.

- Termination. All honest nodes eventually commit a value at any slot.
- Validity. If the designated sender of slot i is honest and invokes $\mathsf{bc}_i(b)$, then all honest nodes commit b at slot i.
- Sequentiality. For any slot i, the sender S_i is allowed to invoke bc_i after bc_j is committed at all honest nodes for all slots $j < i$.

Note that the multi-shot BB has clear boundaries between each slot due to the *sequentiality*. In contrast to atomic broadcast [39], it supports causal inputs. Namely, the sender of each slot can decide its input depending on the previous decisions. As mentioned, using extension protocols with batching cannot solve the problem with linear cost, since $O(n)$ many slots' senders must input in parallel.

Since we assume a synchronous network, the *round complexity* is clearly defined as the number of rounds required. For the *communication complexity*, it is measured as the bit amount sent by honest nodes. In this thesis, we are interested in the *amortized communication complexity* of multi-shot Byzantine broadcast, defined as follows.

Definition 2.1.3 (Amortized Communication Complexity)**.** Let $C(L, n, f)$ be the communication complexity of a multi-shot Byzantine broadcast protocol for n nodes with f faults to commit L slots. The amortized communication complexity of the protocol is defined to be $\lim_{L\to\infty} \frac{C(L,n,f)}{L}$.

For example, a multi-shot BB protocol that naively runs multiple instances of single-shot BB of cost C_{BB}, will have an amortized communication cost of C_{BB}. Since the current state-of-the-art BB protocol for the honest majority scenario is the Momose-Ren Byzantine broadcast of cost $O(\kappa n^2)$, such a naive approach only gives us an amortized $O(\kappa n^2)$ protocol, which has an $O(n)$ gap from the optimal.

2.1.2 Adversary Models

At any point of time during the protocol's execution, a node can either be *honest* or *corrupt*. Honest nodes correctly follow the protocol, while corrupt nodes are controlled by an adversary and can deviate from the prescribed protocol arbitrarily. We allow the adversary to be *rushing*, i.e., it can observe the messages honest nodes want to send in round r before deciding what messages corrupt nodes send in the same round r.

The adversary can be either *static* or *adaptive*. A static adversary selects and corrupts a set of f nodes before the start of the protocol. As soon as the protocol begins, no further corruption is allowed. An adaptive adversary is less restrained and much stronger. In any round r, it can adaptively corrupt honest nodes after observing the messages they want to send in round r, as long as the total number of corrupted nodes does not exceed an upper bound f. If a node $v \in [n]$ becomes newly corrupt in round r, the adversary can make it inject new messages of its choice in the present round r.

Adaptive adversaries are further divided into weakly adaptive and strongly adaptive. A weakly adaptive adversary cannot perform "after-the-fact removal", i.e., erase the messages v originally wanted to send in round r before it became corrupt. A strongly adaptive adversary, however, can perform "after-the-fact removal".

2.1.3 Signatures

We assume an ideal digital signature scheme with a public-key infrastructure (PKI). A message m signed by node i is denoted $\langle m \rangle_i$. Our protocol in Section 4 also assumes a threshold signature scheme [44]. In a (t, n)-threshold signature scheme, each node i can generate a signature share $\langle m \rangle_i$ on a message m. A set of t distinct signature shares $\{\langle m \rangle_{j_1}, .., \langle m \rangle_{j_t}\}$ on the same message m can be combined into a full signature $\mathsf{thsig}(m)$, which has the same length as a single signature share $\langle m \rangle_*$. An adversary cannot generate the full signature $\mathsf{thsig}(m)$ from less than t signature shares. The threshold signature scheme can be set up either through a trusted dealer, or distributed key generation [45].

When we replace the ideal signature with a real-world instantiation that satisfies the standard notion of "unforgeability under chosen-message attack", all of our theorems and lemmas will follow accounting for an additive, negligibly small failure probability due to the failure of the signature scheme — this approach has been commonly adopted in prior works too and is well-known to be cryptographically sound (even against adaptive adversaries).

2.1.4 Time-Lock Puzzles

In Section 5, we utilize a cryptographic tool called time-lock puzzle [46]–[48] to construct our BB protocol under a dishonest majority and strongly adaptive adversary. Time-lock puzzles were first proposed by Rivest, Shamir and Wagner [47] in 1996. At the high level, a time-lock puzzle with hardness parameter ξ ensures that the puzzle solution remains hidden from any machine running in time that is at most ξ fraction of the honest evaluation time, even when the machine has access to unbounded polynomial parallelism. Time-lock puzzles are a high-level primitive that can be applied to various cryptographic applications such as fair contracts [49] and non-malleable commitments [48]. In 2019, Malavolta and Thyagarajan [21] designed a more efficient protocol that supports homomorphic time-lock puzzles. They also discuss many application scenarios for homomorphic puzzles and formally characterize the security guarantees used in this thesis. We review the notion of time-lock puzzles below.

Definition 2.1.4 (Time-lock puzzles)**.** Let $\mathcal{S}$ be a finite domain of size at most $2^{\mathsf{poly}(\lambda)}$. A time-lock puzzle (TLP) with solution space $\mathcal{S}$ is a tuple of algorithms $(\mathsf{Gen}, \mathsf{Solve})$ as defined below:

- $Z \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{T}, s)$: a probabilistic algorithm that takes a security parameter λ, a time parameter $\mathcal{T}$, and a solution $s \in \mathcal{S}$, and outputs a puzzle Z.
- $s \leftarrow \mathsf{Solve}(Z)$: a deterministic algorithm that takes in a puzzle Z, and outputs a solution s.

Correctness. We require that for every λ, every $s \in \mathcal{S}$, every $\mathcal{T}$, $\mathsf{Solve}(\mathsf{Gen}(1^\lambda, \mathcal{T}, s))$ outputs the correct solution s with probability 1.

Efficiency. We require that on a sequential Random-Access Machine, $\mathsf{Gen}(1^\lambda, \mathcal{T}, s)$ runs in at most $\mathsf{poly}(\lambda, \log \mathcal{T})$ steps for any $s \in \mathcal{S}$; and moreover, $\mathsf{Solve}(Z)$ runs in at most $\mathcal{T}$ number of steps for any Z in the support of $\mathsf{Gen}(1^\lambda, \mathcal{T}, \cdot)$.

ξ-hardness. A TLP scheme $(\mathsf{Gen}, \mathsf{Solve})$ is said to be ξ-hard iff there exists a polynomial $\widetilde{\mathcal{T}}$ such that for all polynomials $\mathcal{T}(\cdot) \geq \widetilde{\mathcal{T}}(\cdot)$ and every $\xi\mathcal{T}$-bounded, non-uniform $p.p.t.$ parallel machine

$\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that for all $\lambda \in \mathbb{N}$ and for all $s_0, s_1 \in \mathcal{S}$ it holds that

$$\left|\Pr\left[\mathcal{A}(\mathsf{Gen}(1^\lambda, \mathcal{T}, s_0)) = 1\right] - \Pr\left[\mathcal{A}(\mathsf{Gen}(1^\lambda, \mathcal{T}, s_1)) = 1\right]\right| \leq \mathsf{negl}(\lambda).$$

2.1.5 Verifiable Random Functions

Another cryptographic tool that we use is the verifiable random function (VRF) [50]. A VRF includes the following (possibly randomized) algorithms:

- $(\mathsf{crs}, \{\mathsf{pk}_u, \mathsf{sk}_u\}_{u\in[n]}) \leftarrow \mathsf{Gen}(1^\lambda)$: takes in a security parameter λ and generates public parameters crs, and a public and secret key pair $(\mathsf{pk}_u, \mathsf{sk}_u)$ for each node $u \in [n]$; each sk_u is of the form $\mathsf{sk}_u := (s_u, \rho_u)$ where s_u is said to be the *evaluation key* and ρ_u is said to be the *proof key* for u.
- $(y, \sigma) \leftarrow \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_u, x)$: we shall assume that $\mathsf{Eval} := (E, P)$ has two sub-routines E and P where $\mathsf{Eval}.E$ is *deterministic* and $\mathsf{Eval}.P$ is possibly randomized. Given the public parameters crs, the secret key $\mathsf{sk}_u = (s_u, \rho_u)$, and input $x \in \{0,1\}^{|x|}$, compute $y := \mathsf{Eval}.E(\mathsf{crs}, s_u, x)$ and $\sigma := \mathsf{Eval}.P(\mathsf{crs}, s_u, \rho_u, x)$, and output (y, σ).
- $\{0,1\} \leftarrow \mathsf{Vf}(\mathsf{crs}, \mathsf{pk}_u, x, y, \sigma)$: receives the public parameters crs, a public key pk_u, an input x, a purported outcome y, and a proof σ, outputs either 0 indicating rejection or 1 indicating acceptance.

For the VRF scheme to satisfy correctness, we require that for any $v \in [n]$, for any input x, the following holds with probability 1:

$$(\mathsf{crs}, \{\mathsf{pk}_u, \mathsf{sk}_u\}_{u\in[n]}) \leftarrow \mathsf{Gen}(1^\lambda),\ (y, \sigma) \leftarrow \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_v, x) \\ \Longrightarrow \mathsf{Vf}(\mathsf{crs}, \mathsf{pk}_v, x, y, \sigma) = 1.$$

Pseudorandomness under Selective Opening

To define pseudorandomness under selective opening, we shall consider two games. The first game is intended to capture that the evaluation outcome, i.e., the y term output by Eval, is pseudorandom even when $\mathcal{A}$ can selectively corrupt nodes and open the first component of the corrupted nodes' secret keys. The second game captures the notion that the proof σ does not reveal anything additional even under an adaptive adversary.

First game: pseudorandomness of the evaluation outcome. We consider a selective opening adversary $\mathcal{A}$ that interacts with a challenger denoted $\mathcal{C}$ in the following experiment $\mathsf{Expt}_b^{\mathcal{A}}(1^\lambda)$ indexed by the bit $b \in \{0,1\}$.

$\underline{\mathsf{Expt}_b^{\mathcal{A}}(1^\lambda)}$:

- First, the challenger $\mathcal{C}$ runs the $\mathsf{Gen}(1^\lambda)$ algorithm and remembers the secret key components $(s_1, \ldots, s_n)$ for later use. Note that $\mathcal{C}$ need not give public parameters to $\mathcal{A}$.
- Next, the adversary $\mathcal{A}$ can adaptively make queries of the following forms:
  - **Evaluate**: $\mathcal{A}$ submits a query (u, x), now $\mathcal{C}$ computes $y \leftarrow \mathsf{Eval}.E(\mathsf{crs}, s_u, x)$ and gives y to $\mathcal{A}$.

- **Corrupt**: $\mathcal{A}$ specifies an index $u \in [n]$ to corrupt, and $\mathcal{C}$ parses $\mathsf{sk}_u := (s_u, \rho_u)$ and reveals s_u to $\mathcal{A}$.
- **Challenge**: $\mathcal{A}$ specifies an index $u^* \in [n]$ and an input x. If $b = 0$, the challenger returns a completely random string of appropriate length. If $b = 1$, the challenger computes $y \leftarrow \mathsf{Eval}.E(\mathsf{crs}, s_{u^*}, x)$ and returns y to the adversary.

We say that $\mathcal{A}$ is compliant iff with probability 1, every challenge tuple (u^*, x) it submits satisfies the following: 1) $\mathcal{A}$ does not make a corruption query on u^* throughout the game; and 2) $\mathcal{A}$ does not make any evaluation query on the tuple (u^*, x).

If no efficient and compliant adversary can effectively distinguish $\mathsf{Expt}_0^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Expt}_1^{\mathcal{A}}(1^\lambda)$, then we can be sure that the evaluation outcome of the VRF is pseudorandom even with an adaptive adversary.

Second game: zero-knowledge of the proofs. We also need to make sure that the proof part is zero-knowledge even w.r.t. an adaptive adversary. Therefore, we define another game below where the adversary $\mathcal{A}$ tries to distinguish whether it is playing in the real-world experiment or in the ideal-world experiment:

- *Real-world experiment* Real*:* In the real-world experiment, the challenger runs the $\mathsf{Gen}(1^\lambda)$ algorithm and gives the public parameters crs and all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to $\mathcal{A}$, but keeps $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ to itself. Next, $\mathcal{A}$ can adaptively make the following queries:
  - **Evaluate**: $\mathcal{A}$ submits a query (u, x), now $\mathcal{C}$ computes $(y, \sigma) \leftarrow \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_u, x)$ and gives (y, σ) to $\mathcal{A}$.
  - **Corrupt**: $\mathcal{A}$ specifies an index $u \in [n]$ to corrupt. $\mathcal{C}$ reveals not only sk_u to $\mathcal{A}$, but also all the randomness used in the Eval algorithm for any earlier **Evaluate** query pertaining to u.
- *Ideal-world experiment* $\mathsf{Ideal}^{\mathcal{S}_0,\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$*:* First, the challenger $\mathcal{C}$ runs a simulated setup algorithm

$$(s_1, \ldots, s_n) \leftarrow \mathcal{S}_0(1^\lambda);$$
$$(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \tau) \leftarrow \mathcal{S}_1(1^\lambda);$$

  it gives the public parameters crs and all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to $\mathcal{A}$, but keeps the trapdoor τ to itself.

  Next, $\mathcal{A}$ can adaptively make the following queries:
  - **Evaluate**: $\mathcal{A}$ submits a query (u, x), and now the simulator computes $y := \mathsf{Eval}.E(\mathsf{crs}, s_u, x)$, and $\sigma \leftarrow \mathcal{S}_2(\tau, \mathsf{pk}_u, x, y)$ and gives y, σ to $\mathcal{A}$.
  - **Corrupt**: $\mathcal{A}$ specifies an index $u \in [n]$ to corrupt. Let $\mathcal{I}$ denote the indices of the earlier **Evaluate** queries that correspond to the node $u \in [n]$; and moreover, for $i \in \mathcal{I}$, let the i-th query be of the form (u, x_i) and the result be of the form (y_i, σ_i).

    The challenger $\mathcal{C}$ calls $(\rho_u, \{\psi_i\}_{i \in \mathcal{I}}) \leftarrow \mathcal{S}_3(\tau, \mathsf{pk}_u, s_u, \{x_i, \sigma_i\}_{i \in \mathcal{I}})$, and returns the secret key $\mathsf{sk}_u := (s_u, \rho_u)$ as well as $\{\psi_i\}_{i \in \mathcal{I}}$ to $\mathcal{A}$.

Definition 2.1.5 (Pseudorandomness under selective opening)**.** We say that a VRF scheme satisfies pseudorandomness under selective opening iff:

1. for any compliant non-uniform $p.p.t.$ adversary $\mathcal{A}$, its views in $\mathsf{Expt}_0^{\mathcal{A}}(1^\lambda)$ and $\mathsf{Expt}_1^{\mathcal{A}}(1^\lambda)$ are computationally indistinguishable.

2. there exist $p.p.t.$ simulators $(\mathcal{S}_0, \mathcal{S}_1, \mathcal{S}_2, \mathcal{S}_3)$ such that the outcome of $\mathcal{S}_0(1^\lambda)$ is identically distributed as the $(s_0, \ldots, s_n)$ components[1] generated by the real-world $\mathsf{Gen}(1^\lambda)$ algorithm, and moreover, $\mathcal{A}$'s views in the above Real and $\mathsf{Ideal}^{\mathcal{S}_0,\mathcal{S}_1,\mathcal{S}_2,\mathcal{S}_3}$ are computationally indistinguishable.

Unforgeability

We say that a VRF scheme satisfies unforgeability, if there exists a negligible function $\mathsf{negl}(\cdot)$ such that no non-uniform $p.p.t.$ adversary $\mathcal{A}$ can win the following game with more than $\mathsf{negl}(\lambda)$ probability:

- First, the challenger $\mathcal{C}$ runs the $\mathsf{Gen}(1^\lambda)$ algorithm and gives the public parameters crs and all public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to $\mathcal{A}$, but keeps $\mathsf{sk}_1, \ldots, \mathsf{sk}_n$ to itself.
- The adversary $\mathcal{A}$ can adaptively make the following queries:
  - **Evaluate**: $\mathcal{A}$ submits a query (u, x), now $\mathcal{C}$ computes $(y, \sigma) \leftarrow \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_u, x)$ and gives (y, σ) to $\mathcal{A}$.
  - **Corrupt**: $\mathcal{A}$ specifies an index $u \in [n]$ and $\mathcal{C}$ reveals sk_u to $\mathcal{A}$ as well as random coins used in earlier **Evaluate** queries pertaining to u.
- Finally, $\mathcal{A}$ outputs a tuple (u, x, y, σ). It is said to win the game if either $\mathsf{Vf}(\mathsf{crs}, \mathsf{pk}_u, x, y, \sigma) = 1$, but $y \neq y'$ where $(y', \_) := \mathsf{Eval}(\mathsf{crs}, \mathsf{sk}_u, x)$; or if u has not been corrupted before and $\mathcal{A}$ has not made any **Evaluate** query of the form (u, x).

In other words, we want that except with negligible probability, $\mathcal{A}$ cannot forge the VRF outcome and proof on behalf of any honest node on a point that has not been queried; furthermore, even for corrupted nodes, $\mathcal{A}$ cannot forge an VRF outcome and proof such that the evaluation outcome is different from the honest evaluation outcome.

Abraham et al. [26] proved the following theorem where the bilinear group assumptions needed are the same as those adopted by Groth et al. [20].

Theorem 2.1.6 (Existence of adaptively secure VRFs [26])**.** *Assuming standard bilinear group assumptions and a trusted setup, we can construct a VRF scheme satisfying pseudorandomness under selective opening and unforgeability.*

2.1.6 Expander graph.

Our protocol in Section 4 uses an expander (inspired by [25]), which is a graph with sparse edges but overall good connectivity. More formally, an (n, α, β)-expander ($0 < \alpha < \beta < 1$) is a graph of n vertices s.t. for any set S of αn vertices, the number of neighbors of S is more than βn. It is well-known that for any n and any constants α, β such that $0 < \alpha < \beta < 1$, an expander with constant degree exists [25].

[1] Recall that each secret key sk_u output by $\mathsf{Gen}(1^\lambda)$ where $u \in [n]$ is of the form $\mathsf{sk}_u := (s_u, \rho_u)$.

2.2 Trust Graph: a High Level Description

Byzantine Broadcast under dishonest majority is challenging even under static corruption because the standard random committee election technique fails to work. More concretely, in the honest majority setting and assuming static corruption, a well-known random committee election technique can allow us to compile any polynomial-round BB to a poly-logarithmic round BB protocol. However, as already pointed out by Chan et al. [16], this technique is inapplicable to the corrupt majority setting even under a static adversary. [2] Similarly, we also know of no way to extend the recent techniques of Chan et al. [16] to obtain our result. Instead, we devise novel techniques that redesign the consensus protocol from the ground up.

2.2.1 Trust graph maintenance

First, we devise a new method for nodes to maintain *trust graphs* over time. While previous work [51], [52] also used consistency graphs in multiparty protocols and secret sharing, our trust graph is of a different nature from prior work. We are the first to tie the round complexity of distributed consensus with the diameter of a trust graph, and upper bound the diameter.

The vertices in the trust graph represent nodes in the BB protocol; and an edge between u and v indicates that u and v mutually trust each other. Initially, every node's trust graph is the complete graph; however, during the protocol, if some nodes misbehave, they may get removed completely or get disconnected from other nodes in honest nodes' trust graphs. On the other hand, honest nodes will forever remain direct neighbors to each other in their respective trust graphs.

There are a few challenges we need to cope with in designing the trust graph mechanism. First, if a node v misbehaves in a way that leaves a cryptographic evidence implicating itself (e.g., double-signing equivocating votes), then honest nodes can distribute this evidence and remove v from their trust graphs. Sometimes, however, v may misbehave in a way that does not leave cryptographic evidence: for example, v can fail to send a message it is supposed to send to u, and in this case u cannot produce an evidence to implicate v. In our trust graph mechanism, we allow u to complain about v without providing an evidence, and a receiver of this complaint can be convinced that at least one node among u and v is corrupt (but it may not be able to tell which one is corrupt). In any case, the receiver of this complaint may remove the edge (u, v) from its trust graph. We do not allow a node u to express distrust about an edge (v, w) that does not involve itself — in this way a corrupt node cannot cause honest nodes to get disconnected in their trust graphs.

A second challenge we are faced with is that honest nodes may not have agreement for their respective trust graphs — in fact, reaching agreement on their trust graphs may be as hard as the BB problem we are trying to solve in the first place. However, if honest nodes always share their knowledge to others, we can devise a mechanism that satisfies the following *monotonicity* condition: any honest node's trust graph in round $t > r$ is a subgraph of any honest node's trust graph in round r. In our protocol, we will have to work with this slightly imperfect condition to generate complete agreement.

Finally, although an honest node is convinced that besides their direct neighbors in its own

[2] As Chan et al. [16] point out, the random committee election approach fails to work for corrupt majority (even for static corruption), because members outside the committee cannot rely on a majority voting mechanism to learn the outcome.

trust graph, no one else can be honest, it still must wait to hear what nodes multiple hops away say during the protocol. This is because their direct neighbors may still trust their own neighbors, and the neighbors' neighbors may care about their own neighbors, etc. For information to flow from a node v that is r hops away from u (in u's trust graph), u needs to wait for up to r rounds. This explains why the diameter of the trust graph is critical to the round complexity of our protocol. We will devise algorithms to ensure that honest nodes' trust graphs have *small diameter*. To maintain small diameter, we devise a mechanism for nodes to post-process their trust graphs: for example, although a node u may not have direct evidence against v, if many nodes complain about v, u can be indirectly convinced that v is indeed corrupt and remove v.

2.2.2 The Gossip building block

A common technique in the consensus literature is to bootstrap full consensus from weaker primitives, often called "reliable broadcast" or "gradecast" depending on the concrete definitions [12], [13], [53]. Typically, these weaker primitives aim to achieve consistency whether the sender is honest or not; but they may not achieve liveness if the sender is corrupt [12], [13], [53]. Based on a weaker primitive such as "reliable broadcast" or "gradecast", existing works would additionally rely on random leader election to bootstrap full consensus. Roughly speaking, every epoch a random leader is chosen, and if the leader is honest, liveness will ensue. Additionally, relying on the consistency property of this weaker primitive, with enough care we can devise mechanisms for ensuring consistency within the same epoch and across epochs — in other words, honest nodes must make the same decision whether they make decisions in the same epoch or different epochs.

In our work, we devise a Gossip building block which is also a weakening of full consensus and we would like to bootstrap consensus from this weaker primitive. Our definition of Gossip, however, is tied to the trust graph and departs significantly from prior works. Specifically, Gossip allows a sender $s \in [n]$ to send a message to everyone: *if s wants to continue to remain in an honest node u's trust graph, u must receive some valid message from s at the end of the protocol*, although different honest nodes may receive inconsistent messages from s if s is corrupt. At a high level, the sender s has three choices:

1. it can either send the same valid message to all honest nodes;
2. (**technical challenge*) or it can fail to send a valid message to some honest node, say u, — in this case u will remove s from its trust graph immediately and in the next round all honest nodes will remove s from their trust graphs;
3. or u can send equivocating messages to different honest nodes, but in the next round honest nodes will have compared notes and discovered the equivocation, and thus they remove s from their trust graphs.

The first case will directly lead to progress in our protocol. In the second and third cases, s will be removed from honest nodes' trust graphs; we also make progress in the sense that s can no longer hamper liveness in the future.

An important technical challenge for designing the Gossip protocol lies in the second case above: in this case, u may not have a cryptographic evidence to implicate s and thus u cannot directly convince others to remove s. However, in this case, it turns out that u can be convinced that some of its direct neighbors must be corrupt, and it will instead convince others to remove the edge (u, v) for every direct neighbor v that it believes to be corrupt. Once these edges are removed, s will land in a "remote" part of the graph such that honest nodes can be convinced that it is corrupt and remove it altogether.

2.2.3 Connection with Communication Complexity

While trust graphs are designed to improve round complexity, they can also be used to improve communication complexity through a new query technique. One of the biggest problems with communication complexity in consensus is the relay operation. A common step in BA or BB protocols is that whenever a node receives a certain message, it will relay it to all other nodes in the next round. For example, in the TrustCast protocol, whenever an honest node receives a valid message from the sender, it will forward it to all other nodes. This guarantees that if an honest node receives this message, all honest nodes would receive it in the next round. However, this also means that the relayed message will be sent over the network multiple times, resulting in a $\Theta(n)$ factor increase in the communication complexity.

Note that when all nodes are honest, this relay step is unnecessary. Similarly, if all nodes know exactly which nodes are honest, then the relayed step can be greatly simplified. Our intuition is to utilize the trust graph structure to simplify the relay step. We let nodes who haven't received the message query other nodes, instead of letting those who have already received relay. In Section 5, we will show how to utilize this idea to reduce the communication complexity for both honest-majority and dishonest majority scenarios.

2.3 Intuition for Strongly Adaptive Adversary under Dishonest Majority

Let us now describe our intuition for dealing with a strongly adaptive adversary under dishonest majority. For simplicity, in this informal technical roadmap, we assume that the adversary may adaptively corrupt an arbitrarily large constant fraction of the nodes. In other words, we assume that $f = (1 - \epsilon)n$ in the remainder of this section for an arbitrarily small constant $\epsilon \in (0, 1)$. Our full protocol in the subsequent sections will be stated formally for more general parameters.

The most natural starting point appears to be the recent work by Chan et al. [16] which achieves BB with polylogarithmic round complexity in a weakly adaptive corruption model even under the presence of majority corruptions. Unfortunately, their protocol is prone to an explicit attack that breaks consistency assuming a strongly adaptive adversary.

2.3.1 Chan et al. Breaks under a Strongly Adaptive Adversary

To aid understanding, we first describe Chan et al.'s approach [16] at a high level. Their main idea is a new method of committee election relying on an adaptively secure Verifiable Random Function (VRF) [50], [54] which they call "bit-specific" committee election. More concretely, during setup, a VRF public-key and secret-key pair denoted $(\mathsf{pk}_u, \mathsf{sk}_u)$ is selected for every node $u \in [n]$ and the corresponding public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_n$ are published in the sky. Recall that a designated sender wants to broadcast a bit to all other nodes. Using the VRF, we can define two committees called the 0-committee and the 1-committee, each responsible for voting for the 0-bit and the 1-bit, respectively. Specifically, the b-committee consists of all nodes whose VRF evaluation outcome on the input b is smaller than an appropriate difficulty parameter. The difficulty parameter is chosen such that each committee's size is polylogarithmic in expectation. Now, committee members for each bit b will engage in poly-logarithmically many rounds of voting based on certain voting rules; moreover, all nodes, including committee members and non-committee members, keep relaying the votes they have seen. We will not go into the details of the voting rules, but what is important here is that the security of their scheme critically relies on the fact that the committee members remain secret until they actually cast a vote for the corresponding bit b. More specifically, after the setup phase, each node knows whether it is a member of the 0-committee (or the 1-committee resp.) but this knowledge is kept secret until the node has cast a vote for the bit 0 (or the bit 1 resp.). Further, when a vote for b is cast, the vote is attached with a VRF proof that vouches for the fact that the voter is a member of the b-committee.

In Chan et al.'s scheme [16], if somehow the adversary could predict who is in which committee, then security could be broken, since the adversary would have enough budget to corrupt all members of the 0-committee (or the 1-committee resp.) before their vote gets propagated to everyone. However, since the VRF scheme satisfies pseudorandomness under adaptive corruptions, essentially the adversary cannot effectively guess which nodes are in either committee until the nodes actually cast a vote for the corresponding bit b, divulging the fact that they are in the b-committee. Even though upon observing a node u's vote for a bit b, the adversary can act instantly and corrupt the voter u (who must be a member of the b-committee), it is already too late — u's vote is guaranteed to propagate to all other nodes since a weakly adaptive adversary cannot retroactively erase the vote u had already sent prior to corruption.

Now, if the adversary is actually strongly adaptive, then an explicit attack is to wait till a node u casts a vote for b, act immediately and corrupt u in the same round, and cause u's vote to be delivered to a subset of the honest recipients but not all of them — without going into details, such an attack would break the consistency of Chan et al.'s protocol.

It might be tempting to try to fix the above problem with naive solutions along the following vein: have all honest nodes first commit to their messages (which is either a valid vote or a dummy message), wait for a while, and then open their messages to reveal whether it is a vote. However, such naive attempts do not fundamentally fix the problem, because a strongly adaptive adversary can always immediately erase a vote as soon as it is opened.

2.3.2 A Strawman Scheme

A first strawman idea is to use time-lock puzzles to transiently hide message contents and thereby defeat the agility of the strongly adaptive adversary. Imagine that in some round, some members of the b-committee want to cast a vote for b (henceforth called voters), and other nodes do not want to cast votes (henceforth called non-voters). Recall that the adversary cannot predict a-priori which nodes are members of the b-committee, but if the votes are cast in the clear, then the voter immediately reveals itself to be a b-committee member.

Our idea is 1) for voters to lock the votes temporarily in a time-lock puzzle and send the resulting puzzle rather than the clear-text vote; and 2) for non-voters to send chaff of the same length, also temporarily locked in puzzles. Even if the adversary may have unbounded parallelism, it cannot distinguish within one round of time which nodes are voters and which ones are non-voters.

Although the adversary can adaptively corrupt a subset of nodes and prevent their puzzles from being delivered, such adaptive corruption is basically performed in a blindfolded manner. Finally, if a node is not corrupt in the round in which the puzzle is sent, essentially the puzzle is let through and honest nodes can solve it later given enough time and obtain the message locked inside.

In the strawman scheme, each voting round is prolonged to the time needed for a single node to solve *all* puzzles (plus one more round for sending the puzzles). If every honest node had n parallel processors, then it could solve all puzzles in parallel consuming only a constant number of rounds. However, it is quite unreasonable to assume that all honest nodes have so much parallelism — in particular, note that the amount of parallelism must scale linearly with the number of nodes. Therefore, we would like a better solution where the honest nodes may run on sequential machines, and yet the adversary is allowed unbounded polynomial parallelism.

2.3.3 Our Approach

In our approach, all nodes propagate their own puzzle to everyone else in the first round. If a node remains honest till the end of the first round, then its puzzle is guaranteed to be received by all honest nodes — henceforth we call such puzzles *honest puzzles*. Puzzles sent by nodes that are corrupt before the end of the first round are said to be *corrupt puzzles*.

We now repeat logarithmically many iterations: in each iteration, all nodes share the workload of solving the puzzles, and send solutions to each other. After logarithmically many iterations, we will show that except with negligible probability, all honest puzzles will have been solved and their solutions will be received by all honest nodes (but we do not guarantee that corrupt puzzles are also solved).

Making this idea work, however, is non-trivial, and we are faced with several challenges. One difficulty arises from the fact that the honest nodes do not have common knowledge of the set of puzzles at the start of the protocol, since corrupt nodes can reveal their puzzles only to a subset of the honest nodes. Similarly, at the end of each iteration, honest nodes also do not have common knowledge of which set of puzzles have been solved. Therefore, nodes must coordinate and share the workload of solving puzzles, regardless of their different views of what the remaining puzzles are. Our idea is for nodes to randomly select a somewhat small subset of puzzles to solve in every iteration but how to choose a random subset is somewhat tricky.

Idealized randomized process assuming perfect knowledge of leftover honest puzzles. Although we use randomness to overcome the inconsistency in nodes' views of the remaining puzzle set, it still helps to first think of how a random strategy might work in a perfect, imaginary world where everyone always knows which are the set of *leftover* honest puzzles at the beginning of each iteration — here, a puzzle is said to be *leftover* if no honest node has solved it and propagated its solution yet. In such a perfect world, we can use the following randomized strategy: in the first iteration, there are at most n honest puzzles to start with. Now, everyone chooses each of the n puzzles with some probability p_1 such that the expected number of puzzles each node solves is $p_1 \cdot n$. Note that the adversary can examine which puzzles' solutions each node is propagating at the end of the first iteration, and then adaptively decide on a set of nodes to corrupt, and make these nodes fail to propagate their puzzle solutions. If all honest nodes that tried to solve a specific puzzle Z are corrupt, then Z will be leftover. A smart adversary can try to pick a set of nodes to corrupt such that the set of leftover honest puzzles is maximized. One can prove that as long as $p_1 \cdot n$ is a sufficiently large constant, even if the adversary chooses the worst-case set of size $(1-\epsilon)n$ to corrupt, there cannot be more than $n/2$ leftover honest puzzles except with negligible probability. In other words, the adversary cannot simultaneously deny too many honest puzzles from being solved. Now, in the second iteration, there are at most $n/2$ honest puzzles left, and we can repeat the same but setting $p_2 = 2p_1$, i.e., the probability of sampling each honest puzzle doubles. Again, one can show that after two iterations, there cannot be more than $n/4$ leftover honest puzzles except with negligible probability, and this goes on for logarithmically many rounds at which point all honest puzzles are solved except with negligible probability.

Working with imperfect knowledge and corrupt puzzles. Our actual protocol needs to somehow "embed" the above idealized random process in a world where there can be corrupt puzzles, and moreover, honest nodes do not have a consistent view of which puzzles are solved. This turns out to be tricky — for example, the simplest idea is for nodes to always pick puzzles from the set of puzzles they have seen by the end of the first round (recall that during the first round nodes propagate their own puzzles to each other). However, if in the first round, the adversary discloses $\Theta(n)$ corrupt puzzles (henceforth denoted Q) only to one honest node u, then no one else will be helping to solve these corrupt puzzles Q; and if the probability p keeps doubling in each iteration, u will need to solve $\Theta(n)$ puzzles in the last iteration. So we want u to be able to propagate Q to others, so that when others receive them, they can start solving them too. But this introduces a new issue: the adversary can suddenly disclose a new set of $\Theta(n)$ puzzles late in the protocol, i.e., when the probability p has doubled logarithmically many times and is close to 1. In this case, the same node would have to solve too many puzzles.

To overcome the above issues, our approach adjusts the sampling probability based on the puzzle's *age*. Roughly speaking, we define a puzzle's age as how many iterations ago the puzzle was first seen. Given a puzzle of age α, we will sample it with probability $p_1 \cdot 2^{\alpha}$. In other words, the older the puzzle is, the more likely it will get sampled, and the probability of being sampled doubles with every iteration. Finally, if a node v is detected to have double-signed more than one puzzle, all of v's puzzles will henceforth be ignored.

We will prove later in the technical sections that except with negligible probability, in every iteration, the number of leftover puzzles of age α in the union of the honest nodes' views is upper bounded by $n/2^{\alpha-1}$, as long as each iteration is long enough such that honest nodes can indeed solve all puzzles they have sampled. Note that since puzzles of age α are each sampled with probability $p_1 \cdot 2^{\alpha}$, the expected number of puzzles of age α that are chosen is $\Theta(np_1) = \Theta(1)$. By the Chernoff bound, we can show that except with $\mathsf{negl}(\lambda)$ probability, no more than $\mathsf{poly}\log\lambda$ puzzles of each age α are chosen. Since the protocol runs for logarithmically many iterations, there can be at most logarithmically many different ages. Therefore, in each iteration, every node solves only polylogarithmically many puzzles (except with negligible probability).

Other subtleties. So far, we have implicitly assumed that there is a way to convince another node that some purported puzzle solution is indeed the correct solution, since otherwise the adversary can convince honest nodes to accept wrong solutions. To make sure this is indeed the case, honest nodes sign the message they want to distribute and then lock both the message and the signature inside a puzzle. In this way, a valid solution for a correctly constructed puzzle would be verifiable. However, this is not enough since the adversary can still construct bad puzzles, and when honest nodes solve them, they cannot convince others that the solution is valid. This issue can be fixed if we simply attach a zero-knowledge proof to the puzzle vouching for the fact that it correctly encodes a message and a signature from the purported sender.

Putting it all together. In summary, to obtain Byzantine Broadcast, we first construct a *delayed-exposure message distribution* mechanism called Distribute. In a Distribute protocol, at the beginning every node u receives an input message m_u, and each node would like to distribute its input messages to others. If a node u remains honest at the end of the first round of the protocol, we say its input m_u is an *honest* message.

The Distribute protocol guarantees the following: 1) *liveness*, i.e., all honest messages must be delivered to every honest node at the end of the protocol; and 2) *momentary secrecy*, i.e., by the end of the first round, no probabilistic polynomial-time adversary, even when allowed unbounded parallelism, could have learned any information about the honest messages (although eventually, given sufficiently long polynomial time, the adversary could solve the puzzles and learn the input messages). The protocol works as follows (described below for the special case when $f = (1-\epsilon)n$ where $\epsilon \in (0,1)$ is a constant):

- **Round 1**: Every node $u \in [n]$ computes a signature σ_u on its input message m_u, and computes a puzzle Z_u that encodes (m_u, σ_u). Further, u computes a non-interactive zero-knowledge proof denoted π_u that the puzzle Z_u indeed encodes a pair where the second term is a valid signature on the first one (w.r.t. u's public key).

  Node u now propagates (Z_u, π_u) to everyone along with a signature on the pair.

- **Repeat $\Theta(\log n)$ iterations:** Each iteration has duration $\mathcal{T}_{\text{solve}} \cdot \mathsf{poly}\log\lambda + 1$ where $\mathcal{T}_{\text{solve}}$ is the time it takes for a sequential machine to solve a single puzzle. During each iteration, a node samples each puzzle of age α to solve with independent probability $\min(p_1 \cdot 2^{\alpha}, 1)$, and once solved it propagates the solution (m, σ) to others if σ is a valid signature for m from the purported sender of m.

  At any time, if a node v is detected to have double-signed two different puzzles, all puzzles signed by v will henceforth be ignored.

- **Output.** Finally, for each node $v \in [n]$, if a valid pair (m, σ) has been observed where σ is a valid signature on m under v's public key, then (m, σ) is output as the message received from v; if no such valid pair has been seen, output $\perp$ as the received message from v.

Intuitively, the security properties of Distribute ensure that honest messages will indeed be delivered, and moreover, the adversary cannot base its corruption decisions based on the contents of the messages, and must make corruptions blindly. To get Byzantine Broadcast, we can now plug in the Distribute protocol to distribute batches of votes in the protocol by Chan et al. [16]. Just like in the strawman scheme in Section 2.3.2, here, nodes who do not want to transmit batches of votes must transmit chaff using the Distribute protocol. Through this transformation, we effectively constrain a strongly adaptive adversary such that its capability is roughly the same as a weakly adaptive adversary. We defer the details of the Byzantine Broadcast (BB) protocol to the subsequent technical sections.

Challenges in proving security. Although the intuition is clear, formally reasoning the security of our BB protocol is actually rather subtle due to the use of cryptography. Ideally, we would like to abstract the cryptography away and reason about the core randomized process captured by the protocol in an ideal-world execution. Unfortunately, the real-world protocol does *not* securely emulate the most natural ideal-world protocol that captures the core randomized process. For example, one concrete challenge is the following: we would like to argue that the first-round messages of a Distribute protocol can be simulated by a simulator without knowing the actual input messages of the honest nodes. Unfortunately, the simulated messages can only fool the adversary for a small amount of time, and the adversary could eventually discover that they were being simulated.

To tackle this challenge, our actual proof defines a sequence of hybrids from the real-world experiment with cryptography, to an ideal-world experiment without cryptography. Instead of arguing that the adversary's view is computationally indistinguishable in adjacent hybrids, we argue that the probability of certain bad events happening in the next hybrid is an upper bound of the probability in the previous hybrid (ignoring negligible differences). This means that bad events cannot happen with higher probability in the real-world than in the ideal-world. Finally, since the ideal-world execution does not involve cryptography, we can argue through a probabilistic argument that the probability of relevant bad events is negligibly small.

Finally, another subtlety in both our construction and proofs is the fact that the adversary is strongly adaptive; and therefore we need to rely on cryptographic primitives with suitable adaptive notions of security.

Remark 1 (On how general our Distribute primitive is)**.** *One natural question is whether our* Distribute *primitive can be used to upgrade any weakly adaptive protocol to a strongly adaptive one preserving its security properties. Our result does not imply such a general weakly to strongly adaptive compiler, partly due to the technical challenges mentioned earlier, specifically, the fact that we cannot prove that the real-world protocol emulates some natural ideal-world protocol. As an exciting future direction, it would be great to understand how general our* Distribute *primitive is, i.e., for which class of protocols it is applicable. Further, another exciting question is whether we can get a general weakly to strongly adaptive compiler.*

Chapter 3

Expected Constant Round BB for Dishonest Majority and Weakly Adaptive Corruption

3.1 Trust Graph Maintenance

We first introduce our trust graph ideas in detail. At a very high level, the novelty of our approach lies in the way parties maintain and make use of an undirected *trust graph* over time. In a trust graph, the vertices correspond to all or a subset of the parties participating in the consensus protocol. An edge (u, v) in the trust graph intuitively means that the nodes $u \in [n]$ and $v \in [n]$ mutually trust each other. Since a node in the graph corresponds to a party in the system, to avoid switching between the words "node" and "party", we will just use the word "node".

Initially, every honest node's trust graph is the complete graph over the set $[n]$, i.e., everyone mutually trusts everyone else. However, over the course of the protocol, a node may discover misbehavior of other nodes and remove nodes or edges from its own trust graph accordingly. We will assume that at any point of time, *an honest node u's trust graph must be a single connected component containing u* — effectively u would always discard any node disconnected from itself from its own trust graph.

Notations. Throughout the thesis, we will use G_u^r to denote the node u's updated trust graph in round r (after processing the graph-messages received in round r and updating the trust graph). Sometimes, if the round we refer to is clear, we may also write G_u omitting the round r. We also use $N(v, G)$ to denote the set of neighbors of v in the graph G. In cases where the graph G we refer to is clear, we just abbreviate it to $N(v)$. For convenience, we always assume that a node is a neighbor of itself. Therefore, $v \in N(v)$ always holds.

Finally, we follow our previous notations where n is the number of nodes in the system, f is the upper bound for the number of corrupt nodes and $h = n - f$ is the lower bound for the number of honest nodes.

Important invariants of the trust graph. A very natural requirement is that corrupt nodes can never cause honest nodes to suspect each other; in fact, we want the following invariant:

> **Honest clique invariant:** at any time, in any honest node's trust graph, all honest nodes form a clique. This implies that all honest nodes must forever remain direct neighbors to each other in their trust graphs.

The round complexity of our protocol is directly related to the diameter of honest nodes' trust graphs. Thus, we want to make sure that honest nodes' trust graphs have small diameter. To understand this more intuitively, we can consider an example in which three nodes, u, v, and s execute Byzantine Broadcast with s being the sender. All three nodes behave honestly except that s drops all messages to u. In this case, although u is convinced that s is corrupt and thus removes the edge (u, s) from its trust graph, it cannot prove s's misbehavior to v. Since v still has reasons to believe that s might be honest, v will seek to reach agreement with s. Now, if u tries to reach agreement with v, it has to care about what s says. But since s drops all messages to u, any information propagation from s to u must incur 2 rounds with v acting as the relay.

This example can generalize over multiple hops: although an honest node $u \in [n]$ knows that except for its direct neighbors in its trust graph, everyone else must be corrupt; it must nonetheless wait for information to propagate from nodes multiple hops away in its trust graph. For a node w that is r hops away from u in u's trust graph, information from w may take r rounds to reach u. Summarizing, for our protocol to be round efficient, we would like to maintain the following invariant:

Small diameter invariant: at any point of time, every honest node u's trust graph must have $\Theta(n/h)$ diameter.

Finally, we stress that a difficult challenge we are faced with, is the fact that honest nodes may never be in full agreement w.r.t. their trust graphs at any snapshot of time — in fact, attempting to make honest nodes agree on their trust graph could be as difficult as solving the Byzantine Broadcast problem itself. However, from a technical perspective, what will turn out to be very helpful to us, is the following monotonicity invariant:

Monotonicity invariant: an honest node u's trust graph in round $t > r$ must be a subset of an honest node v's trust graph in round r. Here, we say that an undirected graph $G = (V, E)$ is a subset of another undirected graph $G' = (V', E')$ iff $V \subseteq V'$ and $E \subseteq E'$.

The above trust graph monotonicity invariant can be maintained because of the following intuition: whatever messages an honest node $v \in [n]$ sees in round r, v can relay them such that all other honest nodes must have seen them by round $r + 1$ — in this way the honest node u would perform the same edge/node removal in round $r + 1$ as what v performed in round r.

3.1.1 Conventions and Common Assumptions

Throughout the thesis, we assume that message echoing among honest nodes is implicit (and our protocol will not repeatedly state the echoing):

Implicit echoing assumption: All honest nodes echo every fresh message they have heard from the network, i.e., as soon as an honest node u receives a message m at the beginning of some round r, if this message is well-formed and has not been received before, u relays it to everyone.

Each node has a *consensus module* (see Sections 3.2 and 3.3) and a *trust graph module* which will be described in this section. Messages generated by the trust graph module and the consensus module will have different formats. Henceforth, we may call messages generated by the trust graph module *graph messages*; and we may call all other messages *consensus messages*.

Below, we state some assumptions about the modules and their interfaces. We assume that all messages generated by the consensus module are of the following format:

> **Message format of the consensus module:** All protocol messages generated by the consensus module are of the form $(\mathtt{T}, e, \mathsf{payload})$ along with a signature from the sender, where $\mathtt{T}$ is a string that denotes the type of the message, $e \in \mathbb{N}$ denotes the epoch number (the meaning of this will be clear later in Section 3.3), and $\mathsf{payload}$ is a string denoting an arbitrary payload. Each type of message may additionally require its payload to satisfy some wellformedness requirements.

For example, $(\mathtt{vote}, e, b)$ and $(\mathtt{comm}, e, \mathcal{E})$ represent vote messages and commit messages, respectively in our Byzantine Broadcast protocol (see Section 3.3), where $\mathtt{vote}$ and $\mathtt{comm}$ denote the type of the message, e denotes the epoch number, and the remainder of the message is some payload.

In our consensus module, nodes can misbehave in different ways, and some types of misbehaviors can generate cryptographic evidence to implicate the offending node. We define equivocation evidence below.

> **Equivocation evidence.** In our consensus module, honest nodes are not supposed to double-sign two different messages with the same type and epoch number — if any node does so, it is said to have equivocated. Any node that has equivocated must be malicious. The collection of two messages signed by the same node $u \in [n]$, with the same type and epoch but different payloads, is called an *equivocation evidence* for u.

3.1.2 Warmup: Inefficient Trust Graph Maintenance Mechanism

As a warmup, we first describe an inefficient mechanism for nodes to maintain a trust graph over time such that the aforementioned three invariants are respected. In this warmup mechanism, nodes would need an exponential amount of computation for updating their trust graphs. However, inspired by this inefficient warmup scheme, we can later construct a better approach that achieves polynomial time (see Section 3.1.3).

Note that if the trust graph always remains the complete graph, obviously it would satisfy the aforementioned three invariants. However, keep in mind that the goal for trust graph maintenance is to make sure that corrupt nodes do not hamper liveness. In our protocol, once a node starts to misbehave in certain ways, each honest node would remove them from its trust graph such that they would no longer care about reaching agreement with them.

In our warmup scheme, every node maintains its trust graph in the following manner:

Warmup: an inefficient trust graph maintenance mechanism

- *Node removal upon equivocation evidence.* First, upon receiving an equivocation evidence implicating some node $v \in [n]$, a node removes v from its trust graph as well as all v's incident edges. After the removal, call the post-processing mechanism described below to update the trust graph.
- *Pairwise distrust messages and edge removal.* Sometimes, the consensus module of

node u can observe that a direct neighbor v in its trust graph has not followed the honest protocol (e.g., u is expecting some message from v but v did not send it); however, u may not have a cryptographic evidence to prove v's misbehavior to others. In this case, u's consensus module calls the $\mathsf{Distrust}(v)$ operation:

  – When u's trust graph module receives a $\mathsf{Distrust}(v)$ call, it signs and echoes a distrust message $(\texttt{distrust}, (u, v))$.
  – When a node $w \in [n]$ receives a message of the form $(\texttt{distrust}, (u, v))$ signed by u (w and u might be the same user), w removes the edge (u, v) from its own trust graph[a] and calls the post-processing procedure.

- *Post-processing for maintaining $O(n/h)$ diameter.* The diameter of the trust graph can grow as nodes and edges are being removed. To maintain the property that honest nodes' trust graphs have small diameter, each node performs the following post-processing every time it removes a node or an edge from its trust graph (recall that h denotes the number of honest nodes):

  – **Repeat:** find in its trust graph a node or an edge that is not contained in a clique of size h (henceforth, such a clique is called an h-clique), and remove this node or edge;

    Until no such node or edge exists.
  – u then removes any node that is disconnected from u in u's trust graph.

  Note that the post-processing may be inefficient since it is NP-hard to decide whether there exists an h-clique in a graph.

[a]Since each node will receive its own messages at the beginning of the next round, when a node u calls $\mathsf{Distrust}(v)$, the edge (u, v) will be removed from its own trust graph at the beginning of the next round.

Remark 2. *Note that a* $(\texttt{distrust}, (u, v))$ *message is only valid if it is signed by u, i.e., the first node in the pair of nodes — this makes sure that corrupt nodes cannot misuse distrust messages to cause an edge between two honest nodes to be removed (in any honest node's trust graph).*

Suppose that an honest node never declares $\mathsf{Distrust}$ on another honest node — note that this is a condition that our protocol must respect and it will be proved in Theorem 3.2.2 of Section 3.2. It is not too hard to check that the monotonicity invariant is maintained due to the implicit echoing assumption. We can also check that honest nodes indeed form a clique in all honest nodes' trust graphs. However, proving that all honest nodes' trust graphs have $O(n/h)$ diameter is more technical as it relies on the following graph theoretical observation:

Claim 3.1.1 (Small diameter of h-clique graphs)**.** *Any h-clique graph must have diameter at most $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$ where an h-clique graph is one such that every node or edge is contained within an h-clique.*

Proof. We will prove by contradiction. Assume an h-clique-graph $G = (V, E)$ has diameter $d' > d$. This means that there exists a path $u_0, u_1, \cdots, u_{d'}$ on G which is the shortest path between two nodes u_0 and $u_{d'}$. By definition, there exists an h-clique C_i containing both u_i and u_{i+1} for any $0 \le i \le d' - 1$. Further, any C_i and C_j must be disjoint if $i - j \ge 2$. Otherwise, there would exist a path between u_j and u_{i+1} of length 2, contradicting our assumption that the path is the shortest path. We now discuss different scenarios based on whether n is perfectly divided by h.

- If $n \bmod h \neq 0$, suppose $n = k \cdot h + l$ where k is the quotient of n divided by h and $l \in (0, h)$ is the remainder. By definition, $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1 = 2k$ is even and $d' > 2k$. Thus,

$$\begin{aligned} \left|C_0 \cup C_1 \cup \cdots \cup C_{d'-1}\right| \geq \left|C_0 \cup C_2 \cup \cdots \cup C_{2k}\right| &= \left|C_0\right| + \left|C_2\right| + \cdots + \left|C_{2k}\right| \\ &\geq h \cdot (k+1) > k \cdot h + l = n. \end{aligned} \tag{3.1}$$

  The equation in the first line holds because $C_0, C_2, \cdots, C_{2k}$ are disjoint. We reach a contradiction here since we only have n nodes.

- If n is perfectly divided by h, i.e., $n = k \cdot h$ for some integer k, then $d = 2k - 1$ is an odd number. We then have $d' \geq 2k$ and,

$$\begin{aligned} \left|C_0 \cup C_1 \cup \cdots \cup C_{d'-2}\right| \geq \left|C_0 \cup C_2 \cup \cdots \cup C_{2k-2}\right| &= \left|C_0\right| + \left|C_2\right| + \cdots + \left|C_{2k-2}\right| \\ &\geq h \cdot k = n. \end{aligned} \tag{3.2}$$

  This means that $C_0 \cup C_1 \cup \cdots \cup C_{d'-2}$ already covers all nodes in the graph. So the diameter of the graph should be $d' - 1$, contradicting our assumption that the diameter is d'.

This concludes our proof that the diameter of any h-clique-graph is upper-bounded by $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$. □

This upper bound is tight and can be reached when the graph is a multi-layer graph (see Figure 3.1), where the layer sizes alternate between 1 and $h - 1$. In Figure 3.1, a node is connected with all other nodes in its own layer and the two neighboring layers. Formally, let us denote S_i as the set of nodes in the i^{th} layer ($0 \leq i \leq d$). The graph $G = (V, E)$ satisfies

$$|S_i| = \begin{cases} 1 & \text{if } i \text{ is even} \\ h-1 & \text{if } i \text{ is odd} \end{cases}, \quad V = \bigcup_{i=0}^{d} S_i, \quad E = \Big(\bigcup_{i=0}^{d} (S_i \times S_i)\Big) \cup \Big(\bigcup_{i=1}^{d} (S_{i-1} \times S_i)\Big).$$

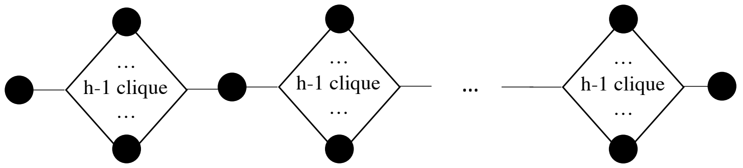

Figure 3.1: A multi-layer graph with the layer size alternating between 1 and $h - 1$. Each layer is completely connected within itself.

We state the following theorem about the warmup scheme.

Theorem 3.1.2 (Inefficient trust graph mechanism)**.** *Suppose that an honest node never declares* Distrust *on another honest node (which is proven to be true in Section 3.2). Then, the above trust graph maintenance mechanism satisfies the honest clique invariant, the monotonicity invariant, and moreover, at any point of time, any honest node's trust graph has diameter at most* $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$.

Proof. To see the honest clique invariant, first observe that no honest node will ever see an equivocation evidence implicating an honest node assuming that the signature scheme is ideal. Therefore, an honest node can never remove another honest node from its trust graph due to having observed an equivocation evidence. Furthermore, since an honest node never declares Distrust on another honest node, no honest node will ever remove an edge between two honest nodes in its trust graph.

The monotonicity invariant follows from the implicit echoing of honest nodes and the fact that if (1) G_u is a subset of G_v and (2) an edge e is not in any h-clique in G_v, then e is also not in any h-clique in G_u. We can prove the monotonicity invariant using induction. In the base case where the round number $r = 0$, the trust graph is a complete graph. Thus, for any two honest nodes u and v, $G_v^1 \subseteq G_u^0$ always holds. We will show that for any round number r, $G_v^{r+1} \subseteq G_u^r$ implies $G_v^{r+2} \subseteq G_u^{r+1}$. Suppose in round r, u receives distrust messages and equivocation proofs on edges $e_1, \cdots, e_m$. u's trust graph in round $r+1$ would then be

$$G_u^{r+1} \leftarrow \text{for } i = 1 \text{ to } m, \Big(\text{apply (remove } e_i) \text{ and post-processing on } G_u^r \Big).$$

The post-processing removes any edge not in any h-clique. Therefore, this is equivalent to

$$G_u^{r+1} \leftarrow \text{ apply post-processing on } G_u^r / \{e_1, \cdots, e_m\}.^{1}$$

Since each honest node echoes all fresh messages it receives, v would receive the distrust messages and equivocation proofs on edges $e_1, \cdots, e_m$ in round $r+1$. Therefore,

$$G_v^{r+2} \subseteq \text{ apply post-processing on } G_v^{r+1} / \{e_1, \cdots, e_m\}.^{2}$$

If $G_v^{r+1} \subseteq G_u^r$, then $G_v^{r+1}/\{e_1, \cdots, e_m\} \subseteq G_u^r/\{e_1, \cdots, e_m\}$. Thus, if an edge is not in any h-clique in $G_u^r/\{e_1, \cdots, e_m\}$, it is also not in any h-clique in $G_v^{r+1}/\{e_1, \cdots, e_m\}$. This means that the post-processing does not change this subset relationship and $G_v^{r+2} \subseteq G_u^{r+1}$ holds. This completes our induction proof on the monotonicity invariant.

Finally, to show the statement about the diameter, observe that the post-processing procedure ensures that the resulting trust graph is an h-clique graph. The statement follows from Claim 3.1.1. □

3.1.3 An Efficient Trust Graph Maintenance Mechanism

Although the warmup mechanism in Section 3.1.2 is inefficient, we can draw some inspiration from it and design an efficient polynomial-time algorithm. In our efficient mechanism, we will maintain every node's trust graph to have diameter at most d, rather than insisting on the more stringent requirement that the graph must be an h-clique graph.

Our idea is to modify the post-processing procedure in the earlier inefficient mechanism to the following efficient approach. Recall that we use $N(v, G)$ to represent the set of v's neighbors in G. If the graph G we are referring to is clear, we just abbreviate it as $N(v)$.

[1] The reason we apply post-processing after each edge removal is to guarantee that the diameter of the trust graph is upper bounded by d at any point of the protocol.

[2] v might have received additional distrust messages or equivocation proofs.

Post-processing for a user u: Iteratively find an edge (v, w) in the trust graph such that $|N(v) \cap N(w)| < h$, and remove the edge; until no such edge can be found. Afterwards, remove all nodes disconnected from u in u's trust graph.

We first show that the new post-processing does not remove edges between honest nodes. Upon termination, it also guarantees that the diameter of the trust graph is upper bounded by $O(n/h)$.

Lemma 3.1.3. *The post-processing (1) only removes edges not in any h-clique and (2) guarantees that the diameter of the trust graph is upper bounded by $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$.*

Proof. Let us consider post-processing on a node u's trust graph G_u. For each edge (v, w) removed during post-processing, $|N(v, G_u) \cap N(w, G_u)| < h$ holds (we only discuss the graph G_u here, so we will abbreviate the neighbor sets as $N(v)$ and $N(w)$). Any clique containing (v, w) can only contain nodes that are in $N(v) \cap N(w)$. So there does not exist an h-clique in G_u that contains (v, w).

We also need to prove that the diameter of the trust graph becomes no larger than d after the post-processing. From this point, we will use G_u just to refer to the trust graph of u when the post-processing terminates. Suppose on the contrary, the diameter of G_u is larger than d. The post-processing guarantees that for any $(v, w) \in G_u$, $|N(v) \cap N(w)| \geq h$. Since the diameter of G_u is larger than d, there must exist two nodes $v, w \in G_u$ such that $d(v, w, G_u) = d + 1$. We define the following notations:

- Suppose the shortest path between v and w is $v_0, \cdots, v_{d+1}$, where v_0 is node v and v_{d+1} is node w.
- We use S_i to denote the set of nodes distance i away from v, i.e., $S_i = \{v' \mid d(v, v', G_u) = i\}$.

By definition, for any $i \neq j$, S_i and S_j should be disjoint. Further, any v_i should belong to the set S_i. Therefore, any $N(v_i)$ should be a subset of $S_{i-1} \cup S_i \cup S_{i+1}$. Since $|N(v_i) \cap N(v_{i+1})| \geq h$ holds for any $0 \leq i \leq d$, we have

$$h \leq |N(v_i) \cap N(v_{i+1})| \leq |(S_{i-1} \cup S_i \cup S_{i+1}) \cap (S_i \cup S_{i+1} \cup S_{i+2})| = |S_i \cup S_{i+1}| = |S_i| + |S_{i+1}|.$$

We construct a graph $G' = (V', E')$ by connecting all nodes between any S_i and S_{i+1}, i.e.,

$$V' = \bigcup_{i=0}^{d+1} S_i, \quad E' = \Big(\bigcup_{i=0}^{d+1} (S_i \times S_i)\Big) \cup \Big(\bigcup_{i=0}^{d} (S_i \times S_{i+1})\Big).$$

For every $0 \leq i \leq d$, the set $S_i \cup S_{i+1}$ forms a clique in G'. And since $|S_i| + |S_{i+1}| \geq h$ holds for any $0 \leq i \leq d$, G' is an h-clique graph. However, G' has diameter $d + 1$. This violates Claim 3.1.1, which proves that the diameter of any h-clique graph is upper bounded by d. We reach a contradiction here. Therefore, after the post-processing terminates, the diameter of the trust graph is no larger than d. This completes our proof. $\square$

In the efficient trust graph maintenance mechanism, the monotonicity invariant is not as apparent. We need to show that if an honest node u removes an edge during post-processing, another honest node v would remove this edge as well in the next round. This can be achieved with the help of the following claim.

Lemma 3.1.4. *If G is a subgraph of H and we use the post-processing algorithm on both G and H to get G' and H', then G' would still be a subgraph of H'.*

Proof. Let us suppose that post-processing removes edges $e_1 = (u_1, v_1), \cdots, e_m = (u_m, v_m)$ from H in order and we denote $H_i = H/\{e_1, \cdots, e_i\}$. By definition of the post-processing algorithm, it must be that for any $1 \le i \le m$,

$$|N(u_i, H_{i-1}) \cap N(v_i, H_{i-1})| < h.$$

We will prove using induction that $e_1, \cdots, e_m$ would be removed from G when we run the post-processing algorithm on G. Firstly, since $G \subseteq H$, we have

$$|N(u_1, G) \cap N(v_1, G)| \le |N(u_1, H) \cap N(v_1, H)| < h.$$

Therefore, if $e_1 \in G$, it would be removed during post-processing. Let us suppose that post-processing has already removed $e_1, \cdots, e_i$ from G, and we denote the graph at this point as G_i. By our assumption,

$$G_i \subseteq G/\{e_1, \cdots, e_i\} \subseteq H/\{e_1, \cdots, e_i\} = H_i.$$

Since $|N(u_{i+1}, H_i) \cap N(v_{i+1}, H_i)| < h$, we have $|N(u_{i+1}, G_i) \cap N(v_{i+1}, G_i)| < h$. This implies that post-processing would remove e_{i+1} as well. This completes our induction proof. $\square$

Using Lemma 3.1.3 and Lemma 3.1.4, we can prove Theorem 3.1.5 as follows.

Theorem 3.1.5 (Efficient trust graph mechanism)**.** *Suppose that an honest node never declares* Distrust *on another honest node (which is proven to be true in Section 3.2). Then, the efficient trust graph maintenance mechanism satisfies the honest clique invariant, the monotonicity invariant, and moreover, at any point of time, any honest node's trust graph has diameter at most $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$.*

Proof. The honest clique invariant is not affected by the changes to the post-processing. As argued in the proof of Theorem 3.1.2, it holds as long as an honest node never declares Distrust on another honest node. By Lemma 3.1.3, the diameter of the trust graph is at most d after calling the post-processing. Since we always call the post-processing algorithm whenever we remove an edge, the diameter of the trust graph is always upper bounded by d.

It remains to show that the monotonicity invariant holds in the efficient trust graph mechanism. The proof idea is the same as in the proof of Theorem 3.1.5. But we state it again for completeness. We will show by induction that for any honest user u, v and any round number r, $G_v^{r+1} \subseteq G_u^r$. In the base case where $r = 0$, $G_v^1 \subseteq G_u^0$ always holds since G_u^0 is a complete graph. We still need to show that for any round number r, $G_v^{r+1} \subseteq G_u^r$ implies $G_v^{r+2} \subseteq G_u^{r+1}$.

Suppose that in round r, u has received distrust messages and equivocation proofs on edges $e_1, \cdots, e_m$. u would then remove $e_1, \cdots, e_m$ from G_u^r and call the post-processing algorithm after each removal. The resultant trust graph would be G_u^{r+1}. It can be shown using Lemma 3.1.4 that this is equivalent to first removing $e_1, \cdots, e_m$ and then calling the post-processing algorithm only once. In other words,

$$G_u^{r+1} = \text{apply post-processing on } G_u^r/\{e_1, \cdots, e_m\}.$$

In round $r+1$, u would echo the distrust messages and equivocation proofs to v. v would remove the edge $e_1, \cdots, e_m$ from G_v^{r+1} and call the post-processing algorithm after each removal. Again, we have

$$G_v^{r+2} \subseteq \text{apply post-processing on } G_v^{r+1}/\{e_1, \cdots, e_m\}.$$

Since $G_v^{r+1} \subseteq G_u^r$, $G_u^r/\{e_1, \cdots, e_m\}$ should also be a subset of $G_v^{r+1}/\{e_1, \cdots, e_m\}$. So by Lemma 3.1.4, G_u^{r+1} should be a subgraph of G_v^{r+2}. This completes our induction proof. Therefore, the monotonicity invariant holds in the efficient trust graph mechanism. $\square$

Finally, observe that the trust graph module's communication (including implicit echoing of graph messages) is upper bounded by $\widetilde{O}(n^4)$ (the $\widetilde{O}$ hides the $\log n$ terms needed to encode a node's identifier). This is because there are at most $O(n^2)$ number of effective `distrust` messages and everyone will echo each such message seen to all nodes.

3.2 New Building Block: the TrustCast Protocol

Starting from this section, we will be describing the consensus module. In this section, we first describe an important building block called Gossip which will play a critical role in our BB protocol. Before describing the consensus module, we first clarify the order in which the trust module and consensus module are invoked within a single round:

1. At the beginning of the round, a node u receives all incoming messages.

2. Next, u's trust graph module processes all the graph-messages and updates its local trust graph:

   - Process all the freshly seen Distrust messages and remove the corresponding edges from its trust graph.
   - Check for new equivocation evidence: if any equivocation evidence is seen implicating any $v \in [n]$, remove v and all edges incident to v from the node's own trust graph.

   Recall also that every time an edge or node is removed from a node's trust graph, a post-processing procedure is called to make sure that the trust graph still has $O(n/h)$ diameter (see Section 3.1.3).

3. Now, u's consensus module processes the incoming consensus messages, and computes a set of messages denoted M to send in this round. The rules for computing the next messages M are specified by our Byzantine Broadcast protocol (Section 3.3) which calls the Gossip protocol (this section) as a building block. The protocol is allowed to query the node's current trust graph (i.e., the state after the update in the previous step).

4. Finally, u sends M to everyone; additionally, for every fresh message first received in this round, u relays it to everyone (recall the "implicit echoing" assumption).

Henceforth, in our consensus module description, whenever we say "at the beginning of round r", we actually mean in round r after Step (2), i.e., after the trust graph module makes updates and yields control to the consensus module.

3.2.1 The Gossip Protocol

Motivation and intuition. We introduce a Gossip protocol that will be used as a building block in our Byzantine Broadcast protocol. In the Gossip protocol, a sender $s \in [n]$ has a message m and wants to share m with other parties. At the end of the Gossip protocol, any honest node either *receives a message from s or removes s from its trust graph*. The Gossip protocol does not guarantee consistency: if the sender is corrupt, different honest parties may output different messages from the sender. However, if the sender is indeed honest, then all honest parties will output the message that the sender sends. Very remotely, the Gossip protocol resembles the notion of a "reliable broadcast" [53] or a "gradecast" [12], [13] which is a weakening of Byzantine Broadcast — many existing works in the consensus literature bootstrap full consensus (or broadcast) from either reliable broadcast or gradecast. Similarly, we will bootstrap Byzantine Broadcast from Gossip; however, we stress that our definition of the Gossip abstraction is novel, especially in the way the abstraction is tied to the trust graph.

Abstraction and notations. A Gossip protocol instance must specify a sender denoted $s \in [n]$; furthermore, it must also specify a verification function Vf for receiving nodes to check the validity of the received message. Therefore, we will use the notation $\mathsf{Gossip}^{\mathsf{Vf},s}$ to specify the verification function and the sender of a Gossip instance. Given a node $u \in [n]$ and a message m, we also use the following convention

$$u.\mathsf{Vf}(m) = \mathsf{true} \text{ in round } r$$

to mean that the message m passes the verification check Vf w.r.t. the node u in round r.

In our Byzantine Broadcast protocol, whenever a sender s calls $\mathsf{Gossip}^{\mathsf{Vf},s}$ to propagate a message m, the verification function Vf and the message m must respect the following two conditions — only if these conditions are satisfied can we guarantee that honest nodes never distrust each other (see Theorem 3.2.2).

- *Validity at origin.* Assuming that the leader s is honest, it must be that $s.\mathsf{Vf}(m) = \mathsf{true}$ in round 0, i.e., at the beginning of the $\mathsf{Gossip}^{\mathsf{Vf},s}$ protocol.

- *Monotonicity condition.* We say that Vf satisfies the monotonicity condition if and only if the following holds. Let $r < t$ and suppose that $u, v \in [n]$ are honest. Then, if $u.\mathsf{Vf}(m) = \mathsf{true}$ in round r, it must hold that $v.\mathsf{Vf}(m) = \mathsf{true}$ in round t as well. Note that in the above, u and v could be the same or different parties.

The first condition guarantees that an honest sender always verifies the message it sends. The second condition, i.e., the Monotonicity condition, guarantees that if an honest node successfully verifies a message, then that message would pass verification of all other honest nodes in future rounds. Together, the two conditions imply that the honest sender's message would pass verification of all honest nodes.

Gossip **protocol.** We describe the $\mathsf{Gossip}^{\mathsf{Vf},s}(m)$ protocol below where a sender $s \in [n]$ wants to propagate a message of the form $m = (\mathtt{T}, e, \mathsf{payload})$ whose validity can be ascertained by the verification function Vf. Recall that by our common assumptions (see Section 3.1.1), honest nodes echo every fresh message seen. Moreover, if an honest node $u \in [n]$ sees the sender's signatures on two messages with the same $(\mathtt{T}, e)$ but different payloads, then u removes the sender s from its trust graph. For brevity, these implicit assumptions will not be repeated in the protocol description below.

Protocol $\mathsf{Gossip}^{\mathsf{Vf},s}(m)$

Input: The sender s receives an input message m and wants to propagate the message m to everyone.

Protocol: In round 0, the sender s sends the message m along with a signature on m to everyone.

Let $d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$, for each round $1 \le r \le d$, every node $u \in [n]$ does the following:

($\star$) If no message m signed by s has been received such that $u.\mathsf{Vf}(m) = \mathsf{true}$ in round r, then for any v that is a direct neighbor of u in u's trust graph: if v is at distance less than r from the sender s, call $\mathsf{Distrust}(v)$.

Outputs: At the beginning of round $d+1$, if (1) the sender s is still in u's trust graph and (2) u has received a message m such that $u.\mathsf{Vf}(m) = \mathsf{true}$, then u outputs m.

To better understand the protocol, consider the example where the sender s is a direct neighbor of an honest node u in u's trust graph. This means that u "trusts" s, i.e., u thinks that s is an honest node. Therefore, u expects to receive s's message in the first round of the Gossip protocol. If u has not received from s in the first round, it knows that s must be corrupted. It would thus remove the edge (u, s) from u's trust graph.

Similarly, if s is at distance r from u in u's trust graph, then u should expect to receive a valid message signed by s in at most r rounds. In case it does not, then u can be convinced that all of its direct neighbors that are at distance $r-1$ or smaller from s in its trust graph must be malicious — therefore u calls Distrust to declare distrust in all such neighbors. Note that the distrust messages generated in round r will be processed at the beginning of round $r+1$. We now utilize the above intuition to prove that the Gossip protocol satisfies the following properties:

- At the end of the Gossip protocol, any honest node either receives a message from s or removes s from its trust graph (Theorem 3.2.1).
- In the Gossip protocol, we never remove edges between two honest nodes in any honest node's trust graph (Theorem 3.2.2).

In the rest of the thesis, we always use the variable d to represent $\lceil n/h \rceil + \lfloor n/h \rfloor - 1$.

Theorem 3.2.1. *Let $u \in [n]$ be an honest node. At the beginning of round $d+1$, either the sender s is removed from u's trust graph or u must have received a message m signed by s such that $u.\mathsf{Vf}(m) = \mathsf{true}$ in some round r.*

Proof. By the definition of the $\mathsf{Gossip}^{\mathsf{Vf},s}$ protocol, if in round r, the node u has not received a message m signed by s such that $u.\mathsf{Vf}(m) = \mathsf{true}$ in round r, then u will call $\mathsf{Distrust}(v)$ for each of its neighbors v that is within distance $r-1$ from s. The $\mathsf{Distrust}(v)$ operation generates a distrust message that will be processed at the beginning of round $r+1$, causing u to remove the edge (u, v) from its trust graph. After removing the edge (u, v), the trust graph module will also perform some post-processing which may further remove additional edges and nodes. After this procedure, s must be at distance at least $r+1$ from u or removed from u's trust graph.

By setting the round number r to d, we can conclude that at the beginning of round $d+1$, if u has not received a message m signed such that $u.\mathsf{Vf}(m) = \mathsf{true}$, then s must be either at distance at least $d+1$ from u or removed from u's trust graph. Yet, u's trust graph must contain a single connected component containing u, with diameter at most d. So s must be removed from u's trust graph. □

Theorem 3.2.2. *If the validity at origin and the monotonicity conditions are respected, then an honest node $u \in [n]$ will never call $\mathsf{Distrust}(v)$ where $v \in [n]$ is also honest.*

Proof. We can prove by contradiction: suppose that in round $r \in [1, d]$, an honest node u calls $\mathsf{Distrust}(v)$ where $v \in [n]$ is also honest. This means that in round r, u has not received a message m signed by s such that $u.\mathsf{Vf}(m) = \mathsf{true}$ in round r. Due to the implicit echoing and the monotonicity condition of Vf, it means that in round $r-1$, v has not received a message m signed by s such that $v.\mathsf{Vf}(m) = \mathsf{true}$ in round $r-1$. We may now consider two cases:

- **Case 1: suppose $r - 1 = 0$.** If the validity at origin condition holds, then v cannot be the sender s. In this case u cannot call $\mathsf{Distrust}(v)$ in round 0 because v is at distance at least 1 from the sender s.

- **Case 2: suppose $r - 1 > 0$.** By definition of the $\mathsf{Gossip}^{\mathsf{Vf},s}$ protocol, in round $r-1$, v would send $\mathsf{Distrust}(w)$ for any w within distance $r-2$ from s in G_v^{r-1}. Suppose v sends distrust messages on $w_1, \cdots, w_l$ and we denote the graph $G' \leftarrow G_v^{r-1}/\{(v, w_1), \cdots, (v, w_l)\}$. Then, in G', the distance between v and s should be at least r. Let us now consider node u and u's trust graph. By trust graph monotonicity, u's trust graph at the beginning of round r, i.e., G_u^r, should be a subset of G_v^{r-1}. Further, u would receive v's distrust messages on $w_1, \cdots, w_l$ in round r. Thus,
$$G_u^r \subseteq G_v^{r-1}/\{(v, w_1), \cdots, (v, w_l)\}.$$
This implies that the distance between v and s in G_u^r should be at least r, contradicting our assumption that the distance between v and s is $r-1$.

In either case, we have reached a contradiction. □

In this section, we provided a Gossip protocol with nice properties (Theorem 3.2.1 and 3.2.2) related to the trust graph. In the next section, we will show how to bootstrap full consensus from the Gossip protocol.

Remark 3. *Later, when Gossip is invoked by a parent protocol, it could be invoked in an arbitrary round r_{init} of the parent protocol; moreover, at invocation, honest nodes' trust graphs need not be complete graphs. In this section (Section 3.2), we denote the initial round as round 0. We say that a sender s trustcasts message m with verification function Vf if s calls $\mathsf{Gossip}^{\mathsf{Vf},s}$ on message m. If the verification function Vf is clear in the context, we just say s trustcasts a message m*

3.3 Byzantine Broadcast under Static Corruptions

We first present a Byzantine Broadcast (BB) protocol assuming an ideal leader election oracle and assuming static corruptions. In subsequent sections, we will remove this idealized leader election oracle through cryptography.

3.3.1 Definitions and Notations

Leader election oracle. We use $\mathcal{F}_{\text{leader}}$ to denote an ideal leader election oracle. The protocol proceeds in *epochs* denoted $e = 1, 2, \ldots$, where each epoch consists of $O(d)$ number of rounds. We assume that:

- The leader of epoch 1, denoted L_1, is the designated sender of the Byzantine Broadcast.
- At the beginning of each epoch $e > 1$, $\mathcal{F}_{\text{leader}}$ chooses a fresh random L_e from $[n]$ and announces L_e to every node. L_e is now deemed the leader of epoch e.

Commit evidence. In our Byzantine Broadcast protocol, each node uses the Gossip protocol to send messages until it becomes confident as to which bit to commit on. Afterwards, it needs to convince other nodes to also commit on this bit using what we call a *commit evidence*. In other words, once a node generates a valid commit evidence, all other nodes that receive it will commit on the corresponding bit. At a high level, we want the commit evidence to satisfy the following properties.

- It is impossible for two nodes to generate valid commit evidences on different bits.
- If the leader in this epoch is honest, at least one honest node should be able to generate a commit evidence on the leader's proposed bit.

The first property guarantees consistency while the second property guarantees liveness. We first define what is a commit evidence in our protocol. After we describe our protocol in Section 3.3.2, we will prove that this definition satisfies the two properties above.

Fix an epoch e and a bit $b \in \{0, 1\}$, we say that a collection $\mathcal{E}$ containing signed messages of the form $(\texttt{vote}, e, b)$ is an epoch-e commit evidence for b w.r.t. G_u^r iff for every $v \in G_u^r$, $\mathcal{E}$ contains a signed message $(\texttt{vote}, e, b)$ from v. Recall that G_u^r is u's trust graph at the beginning of round r (after processing graph-messages). We also call an epoch-e commit evidence for b w.r.t. G_u^r "a commit evidence for (e, b) w.r.t. G_u^r".

Fix $u \in [n]$ and the round r. We say that a commit evidence for (e, b) w.r.t. G_u^r is *fresher* than a commit evidence for (e', b') w.r.t. G_u^r iff $e' > e$. Henceforth, we will assume that $\perp$ is a valid epoch-0 commit evidence for either bit.

Remark 4. *In our protocol description, if we say that "node $u \in [n]$ sees a commit evidence for (e, b) in round r", this means that at the beginning of the round r, after having processed graph-messages, node u has in its view a commit evidence for (e, b) w.r.t. G_u^r. If we say "node $u \in [n]$ sees a commit evidence for (e, b)" without declaring the round r explicitly, then implicitly r is taken to be the present round.*

Lemma 3.3.1 (Commit evidence monotonicity lemma)**.** *Let $u, v \in [n]$ be honest nodes. A commit evidence for (e, b) w.r.t. G_u^r must be a commit evidence for (e, b) w.r.t. G_v^t for any $t > r$. Note that in the above, u and v can be the same or different node(s).*

Proof. Due to the trust graph monotonicity lemma, we have $G_u^t \subseteq G_v^r$ since $t > r$. The fact then follows directly. $\square$

3.3.2 Protocol

Our protocol proceeds in incrementing epochs where each epoch consists of three phases, called **Propose**, **Vote**, and **Commit**, respectively. Each phase has $O(d)$ ($d = \lceil n/h \rceil + \lfloor n/h \rfloor - 1$) rounds. Intuitively, each phase aims to achieve the following objectives:

- **Propose**: the leader uses the Gossip protocol to share the freshest commit evidence it has seen.
- **Vote**: each node uses the Gossip protocol to relay the leader's proposal it receives in the propose phase. At the end of the vote phase, each node checks whether it can construct a commit evidence.
- **Commit**: nodes use the Gossip protocol to share their commit evidence (if any exists).

Besides the three phases, there is also a termination procedure (with the entry point **Terminate**) that runs in the background and constantly checks whether the node should terminate. To apply the Gossip protocol in each phase, we need to define the corresponding verification functions such that the *monotonicity condition* and the *validity at origin condition* (defined in Section 3.2.1) are satisfied. Finally, we need to show that the commit evidence satisfies the properties mentioned in Section 3.3.1.

Throughout the thesis, we use the notation _ to denote a wildcard field that we do not care about.

For each epoch $e = 1, 2, \ldots$:

1. **Propose** ($O(d)$ rounds): The leader of this epoch L_e performs the following:
   - Choose a proposal as follows:
     - If $e = 1$, the sender L_1 chooses $P := (b, \perp)$ where b is its input bit.
     - Else if a non-$\perp$ commit evidence (for some bit) has been seen, let $\mathcal{E}(e, b)$ denote the freshest such commit evidence and let $P := (b, \mathcal{E}(e, b))$.
     - Else, the leader L_e chooses a random bit b and let $P := (b, \perp)$.
   - Trustcast the proposal $(\texttt{prop}, e, P)$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{prop}}, L_e}$ where the verification function $\mathsf{Vf}_{\mathrm{prop}}$ is defined such that $v.\mathsf{Vf}_{\mathrm{prop}}(\texttt{prop}, e, (b, \mathcal{E})) = \mathsf{true}$ in round r iff:
     (a) $\mathcal{E}$ is a valid commit evidence vouching for the bit b proposed; and
     (b) for every $u \in G_v^r$, $\mathcal{E}$ is at least as fresh as any commit evidence trustcast by u in the **Commit** phase of *all* previous epochs — recall that $\perp$ is treated as a commit evidence for epoch 0.

   Notation: at the end of $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{prop}}, L_e}$, for a node $u \in [n]$, if L_e is still in u's trust graph, we say that the unique message $(\texttt{prop}, e, (b, \_))$ output by $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{prop}}, L_e}$ in u's view is L_e's proposal, and the corresponding bit b is L_e's proposed bit (in u's view).

2. **Vote** ($O(d)$ rounds): Every node $u \in [n]$ performs the following:
   - If L_e is still in u's trust graph, then set $b' := b$ where $b \in \{0, 1\}$ is L_e's proposed bit; else set $b' := \perp$.

- Trustcast a vote of the form $(\texttt{vote}, e, b')$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\text{vote}}, u}$, where the verification function $\mathsf{Vf}_{\text{vote}}$ is defined such that $v.\mathsf{Vf}_{\text{vote}}(\texttt{vote}, e, b') = \mathsf{true}$ in round r iff (1) either L_e has been removed from G_v^r, or (2) b' agrees with L_e's proposed bit (in v's view).

3. **Commit** ($O(d)$ rounds): Every node $u \in [n]$ performs the following:

   - If everyone still in u's trust graph voted for the same bit $b \in \{0, 1\}$ (as defined by the outputs of the $\mathsf{Gossip}^{\mathsf{Vf}_{\text{vote}}, u}$ protocols during the **Vote** phase), then **output** the bit b and trustcast a commit message $(\texttt{comm}, e, \mathcal{E})$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\text{comm}}, u}$, where $\mathcal{E}$ contains a signed vote message of the form $(\texttt{vote}, e, \_)$ from everyone in u's trust graph.
   - Else, use $\mathsf{Gossip}^{\mathsf{Vf}_{\text{comm}}, u}$ to trustcast the message $(\texttt{comm}, e, \perp)$.

   We define the verification function $\mathsf{Vf}_{\text{comm}}$ below. $v.\mathsf{Vf}_{\text{comm}}(\texttt{comm}, e, \mathcal{E}) = \mathsf{true}$ in round r iff the following holds: if $L_e \in G_v^r$, $\mathcal{E}$ must be a valid commit evidence for (e, b) where b is L_e's proposed bit.

Terminate: In every round r, every node u checks whether there exists (e, b) such that u has seen, from everyone in G_u^r, a signed message of the form $(\texttt{comm}, e, \mathcal{E})$ where $\mathcal{E}$ a valid commit evidence for (e, b). If so, u terminates (recall that by our implicit assumptions, the node u will echo these messages to everyone before terminating).

Intuition for the verification functions: Recall that in Theorem 3.2.1, we show that at the end of a $\mathsf{Gossip}^{\mathsf{Vf}, s}$ protocol, if the sender s remains in an honest node u's trust graph, then u must have received a message m signed by s such that $u.\mathsf{Vf}(m) = \mathsf{true}$. While the three verifications $\mathsf{Vf}_{\text{prop}}, \mathsf{Vf}_{\text{vote}}, \mathsf{Vf}_{\text{comm}}$ seem complicated, they are more intuitive to understand when we look at what Theorem 3.2.1 implies for each of them.

- $\mathsf{Gossip}^{\mathsf{Vf}_{\text{prop}}, L_e}$ guarantees: at the end of the propose phase in epoch e, if the leader L_e remains in an honest node u's trust graph, then u has received a proposal from L_e containing the freshest commit evidence u has seen.

- For any node v, $\mathsf{Gossip}^{\mathsf{Vf}_{\text{vote}}, v}$ guarantees: at the end of the vote phase in epoch e, if v remains in an honest node u's trust graph, then either (1) the leader L_e is no longer in u's trust graph or (2) u has received v's vote on a bit b which matches L_e's proposed bit (in u's view). In other word, *if the leader L_e remains in u's trust graph at the end of the vote phase, then u must have received votes on L_e's proposed bit from v from every node in u's trust graph.*

- For any node v, $\mathsf{Gossip}^{\mathsf{Vf}_{\text{comm}}, v}$ guarantees: at the end of the commit phase in epoch e, if v remains in an honest node u's trust graph, then either (1) the leader L_e is no longer in u's trust graph or (2) u has received a valid commit evidence on L_e's proposed bit from v. Similarly, this is equivalent to saying that *if the leader L_e remains in u's trust graph at the end of the commit phase, then u must have received a commit evidence on L_e's proposed bit from v.*

Further, in Theorem 3.2.2, we show that if the verification functions respect the *monotonicity condition* and *validity at origin*, then honest nodes always remain connected in any honest node's trust graph. Assume the three verification functions satisfy those properties, then if the leader L_e is honest, for any honest node u:

- In the propose phase, u receives a proposal from L_e containing the freshest commit evidence.
- In the vote phase, u receives consistent votes on L_e's proposed bit from every node in u's trust graph. This allows u to construct a commit evidence on L_e's proposed bit.
- In the commit phase, u receives a commit evidence on L_e's proposed bit from every node in u's trust graph. This allows u to terminate.

In the rest of the section, we will generalize the above intuitions into a formal proof of correctness for our Byzantine Broadcast protocol.

3.3.3 Proof of Correctness for the Verification Functions

To apply the properties of the Gossip protocol, we must show that our verification functions respect the *monotonicity condition* and *validity at origin*. The proof is straightforward. The *monotonicity condition* follows from the trust graph's *monotonicity invariant* and our *implicit echoing* assumption. The *validity at origin* property can be verified by taking the sender's messages into the verification functions and checking if the verification functions output true. For completeness, we list the proof for each verification function and property as follows.

Remark 5. *In Section 3.2, we proved two theorems (Theorems 3.2.1 and 3.2.2) regarding the* Gossip *protocol. Theorem 3.2.2 requires the verification function to respect the monotonicity condition and validity at origin. However, Theorem 3.2.1 does not. It holds for arbitrary verification functions. Therefore, we can apply Theorem 3.2.1 to prove that the verification functions respect the monotonicity condition and validity at origin.*

Lemma 3.3.2. $\mathsf{Vf}_{\text{prop}}$ *satisfies the monotonicity condition.*

Proof. Recall that a propose message $(\texttt{prop}, e, (b, \mathcal{E}))$ passes the verification of $\mathsf{Vf}_{\text{prop}}$ w.r.t. node u in round r iff:

(a) $\mathcal{E}$ is a valid commit evidence vouching for the bit b proposed; and

(b) for every $v \in G_u^r$, $\mathcal{E}$ is at least as fresh as any commit evidence trustcast by v in the **Commit** phase of *all* previous epochs.

Let u, v be two honest nodes and let $r < t$. Suppose that in round r, u verifies the message $(\texttt{prop}, e, (b, \mathcal{E}))$. In round t, v would check the same condition and we want to show that the check will succeed.

If condition (a) holds for u in round r, then it must hold for v in round t by the commit evidence monotonicity lemma. We now focus on condition (b) and assume $e > 1$ without loss of generality. By the trust graph monotonicity lemma, $G_v^t \subseteq G_u^r$. By Theorem 3.2.1, for any node $w \in G_v^t$, v must have received a commit message $(\texttt{comm}, e', \mathcal{E}')$ from w in the **Commit** phase of every epoch $e' < e$. Moreover, $\mathcal{E}'$ must agree with what u has heard. Otherwise, u would have forwarded the equivocating commit message to v (by the implicit echoing assumption) and v would have removed the node w from its trust graph. By the commit evidence monotonicity lemma, if condition (b) passes for u in round r it must pass for v in round $t > r$. We therefore conclude that the verification must succeed w.r.t. v in round t. □

Lemma 3.3.3. $\mathsf{Vf}_{\text{vote}}$ *satisfies the monotonicity condition.*

Proof. Recall that a vote message $(\texttt{vote}, e, b')$ passes the verification of $\mathsf{Vf}_{\text{vote}}$ w.r.t. node u in round r iff: either L_e has been removed from G^r_u, or b' agrees with L_e's proposed bit.

Let u, v be two honest nodes, and let $r < t$. If in round r, the message $(\texttt{vote}, e, b')$ passes the verification of $\mathsf{Vf}_{\text{vote}}$ w.r.t. node u, then it must be that in round r, either $L_e \notin G^r_u$; or $L_e \in G^r_u$ and u heard L_e propose the same bit $b' \in \{0, 1\}$.

If the former happens, then in round t, $L_e \notin G^t_v$ by the trust graph monotonicity lemma and thus in round t, $(\texttt{vote}, e, b')$ must pass the verification function $\mathsf{Vf}_{\text{vote}}$ w.r.t. the node v.

If the latter happens, then if in round t, $L_e \notin G^t_v$ then obviously the verification $\mathsf{Vf}_{\text{vote}}$ would pass w.r.t. v in round t. Henceforth, we focus on the case when $L_e \in G^t_v$. In this case, by Theorem 3.2.1, v must have received a proposal from L_e on some bit b''. Further, by the implicit echoing assumption, u would relay L_e's proposal on b' to v. This implies that $b' = b''$, since otherwise v would have detected equivocation from L_e and removed L_e from its trust graph. Therefore, in round t, $(\texttt{vote}, e, b')$ must pass the verification function $\mathsf{Vf}_{\text{vote}}$ w.r.t. the node v. □

Lemma 3.3.4. $\mathsf{Vf}_{\text{comm}}$ *satisfies the monotonicity condition.*

Proof. Let u, v be honest nodes, and let $r < t$. If $(\texttt{comm}, e, \mathcal{E})$ passes the verification $\mathsf{Vf}_{\text{comm}}$ w.r.t. G^r_u, then either $L_e \notin G^r_u$ or $\mathcal{E}$ is a commit evidence w.r.t. G^r_u. If the former case, by the trust graph monotonicity lemma, $L_e \notin G^t_v$. If the latter case, then $\mathcal{E}$ is a commit evidence w.r.t. G^t_v due to the commit evidence monotonicity lemma.

□

We now prove the validity at origin condition for all invocations of Gossip.

Fact 3.3.5. *If an honest node u uses* Gossip *to send a $(\texttt{prop}, e, P)$ message or a $(\texttt{vote}, e, b)$ message in some round r, the message satisfies the corresponding verification function,* $\mathsf{Vf}_{\text{prop}}$ *or* $\mathsf{Vf}_{\text{vote}}$*, respectively, w.r.t. the node u in round r.*

Proof. For $\mathsf{Vf}_{\text{prop}}$, the proof is straightforward by construction. For $\mathsf{Vf}_{\text{vote}}$, the proof is also straightforward by construction, and additionally observing that if L_e remains in an honest node u's trust graph, it cannot have signed equivocating proposals. □

Fact 3.3.6. *Suppose that in some epoch e by the end of the* **Vote** *phase, L_e remains in an honest node u's trust graph. Then, by the end of the* **Vote** *phase of epoch e, u must have received a vote message of the form $(\texttt{vote}, e, b)$ (where b denotes the bit proposed by L_e) from every node in u's trust graph.*

Proof. By Theorem 3.2.1, if L_e remains in u's trust graph by the end of the **Vote** phase, u must have received a proposal $(\texttt{prop}, e, \_)$ from L_e in the propose phase. Moreover, for any v that remains in u's trust graph by the end of the **Vote** phase, u must have received a vote $(\texttt{vote}, e, b)$ from v.

Now, the fact follows because u would check $\mathsf{Vf}_{\text{vote}}$ on every vote it receives, and $\mathsf{Vf}_{\text{vote}}$ makes sure that the vote is only accepted if the vote agrees with L_e's proposal. □

Fact 3.3.7. *If an honest node u trustcasts a $(\texttt{comm}, e, \mathcal{E})$ message in some round r, the message satisfies the verification function* $\mathsf{Vf}_{\text{comm}}$ *w.r.t. the node u in round r.*

Proof. Follows directly from Fact 3.3.6. □

We have shown that the three verification functions all respect the *monotonicity condition* and *validity at origin*. Therefore, by Theorem 3.2.2, the Gossip protocol never removes edges between honest nodes in any honest node's trust graph.

Lemma 3.3.8. *For any two honest nodes u and v, throughout the entire Byzantine Broadcast protocol, v remains one of u's neighbors in u's trust graph.*

3.3.4 Consistency and Validity Proof

We first prove that our Byzantine Broadcast protocol achieves consistency, i.e., honest nodes always output the same bit. We divide the proof into two parts. First, we show that within the same epoch, two honest nodes cannot commit on different bits. Secondly, we show that even across different epochs, consistency is still guaranteed.

Lemma 3.3.9 (Consistency within the same epoch)**.** *If an honest node $u \in [n]$ sees an epoch-e commit evidence for the bit $b \in \{0, 1\}$ in some round r, and an honest node $v \in [n]$ sees an epoch-e commit evidence for the bit $b' \in \{0, 1\}$ in some round t, it must be that $b = b'$.*

Proof. Let $\mathcal{E}$ be the epoch-e commit evidence seen by u in round r and let $\mathcal{E}'$ be the epoch-e commit evidence seen by v in round t. Due to the honest clique invariant of the trust graph, $\mathcal{E}$ must contain signatures on $(\texttt{vote}, e, b)$ from every honest node, and $\mathcal{E}'$ must contain signatures on $(\texttt{vote}, e, \widetilde{b})$ from every honest node. However, each honest node will only vote for a single bit in any given epoch e. It holds that $b = b'$. □

Lemma 3.3.10 (Consistency across epochs)**.** *If an honest node $u \in [n]$ outputs the bit b in some epoch e, then in every epoch $e' > e$, no honest node $v \in [n]$ can ever see a commit evidence for $(e', 1 - b)$.*

Proof. We will use induction to show that honest nodes will never receive commit evidence for the bit $1 - b$ in any epoch after e. By the protocol definition, for u to output b in epoch e, it must have seen a commit evidence for (e, b) at the beginning of the **Commit** phase in epoch e. We have already shown in Lemma 3.3.9 that any two nodes cannot commit on different bits within the same epoch. Therefore, there cannot exist any commit evidence for $1 - b$ in epoch e. Thus, we have shown the base case of our induction.

Suppose no honest node has seen any commit evidence for $1-b$ between epoch e and e' ($e' \geq e$), we will show that no commit evidence will be seen for $1 - b$ in epoch $e' + 1$ as well. Note that in epoch e, u will use $\mathsf{Gossip}^{\mathsf{Vf}_{\text{comm}}, u}$ to trustcast its commit evidence for (e, b), and all honest nodes will receive it by the end of epoch e. Since no commit evidence for $1 - b$ has been seen afterwards, for any honest node, the freshest commit evidence it has seen is on b. Now, during epoch $e' + 1$, every honest node will reject $L_{e'+1}$'s proposal (where reject means not passing the $\mathsf{Vf}_{\text{prop}}$ function) unless it is for the same bit b; and if they do reject $L_{e'+1}$'s proposal, they will vote on $\perp$. Therefore, in epoch $e' + 1$, no honest node will vote for $1 - b$, and no honest node will ever see a commit evidence for $(e' + 1, 1 - b)$. This completes our induction proof. □

Theorem 3.3.11 (Consistency)**.** *If honest nodes u and v output b and b', respectively, it must be that $b = b'$.*

Proof. For an honest node to output b in epoch e, it must observe a commit evidence for (e, b) in epoch e. Consider the earliest epoch e in which an honest node, say, u', outputs a bit b. By definition, every other honest node will output in epoch e or greater. By Lemma 3.3.9, no honest node will output $1 - b$ in epoch e. By Lemma 3.3.10, no honest node will output $1 - b$ in epoch $e' > e$. $\square$

Next, we show that our protocol achieves validity.

Theorem 3.3.12 (Validity). *If the designated sender L_1 is honest, then everyone will output the sender's input bit.*

Proof. By Fact 3.3.6 and the honest clique invariant, for any honest node u, any node that remains in its trust graph by the end of the **Vote** phase of epoch 1 must have trustcast to u a vote of the form $(\texttt{vote}, e = 1, b)$ where b must agree with L_1's proposed bit. Thus, u will output b in epoch 1. $\square$

Theorems 5.3.13 and 5.3.14 together imply that our protocol achieves Byzantine Broadcast. It remains to show that our protocol terminates and has expected constant round complexity.

Remark 6. *Throughout Section 3.3, we assumed that the signature scheme is ideal and there are no signature forgeries. When we replace the ideal signature with a real-world instantiation, it will introduce only negligible failure probability.*

3.3.5 Round Complexity Analysis

Finally, we show that our protocol achieves liveness, i.e., all honest nodes eventually terminate. Moreover, we analyze the round complexity and communication complexity of the protocol, showing that the protocol terminates in expected $O((n/h)^2)$ rounds and has $\widetilde{O}(n^4)$ communication complexity.

Fact 3.3.13. *If some honest node terminates in round r, then all honest nodes will have terminated by the end of round $r + 1$.*

Proof. If an honest node terminates in round r, it must have received consistent commit evidence from every node in its trust graph. By the implicit echoing assumption, it would forward those commit evidences to all other honest nodes before round $r + 1$. By the trust graph monotonicity invariant and the commit evidence monotonicity lemma (Lemma 3.3.1), all other honest nodes would gather enough commit evidence in round $r + 1$ and terminate as well. $\square$

The following theorem says that liveness will ensue as soon as there is an honest leader in some epoch (if not earlier). Now if the leader election is random, this will happen in expected $O(n/h)$ number of epochs. Since each epoch is $O(d) = O(n/h)$ rounds, every honest node outputs some bit in expected $O((n/h)^2)$ rounds.

Theorem 3.3.14 (Liveness). *If in some epoch e, the leader L_e is honest, then one round after this epoch, every honest node would have terminated.*

Proof. Without loss of generality, we may assume that no node has yet terminated by the end of epoch e, since otherwise by Fact 3.3.13, the theorem immediately holds. If no honest node has terminated by the end of epoch e, then we may assume that every honest node will participate in all the Gossip protocols till the end of epoch e and thus we can rely on the properties of Gossip in our reasoning.

Let u be an honest node. By Fact 3.3.6 and the honest clique invariant, at the end of the **Vote** phase of epoch e, u must have received a vote message on L_e's proposed bit from every node in u's trust graph. Further, by applying Theorem 3.2.1 to $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{comm}}, -}$, we know that by the end of the **Commit** phase, u must have received a commit evidence on L_e's proposed bit from every node in u's trust graph. Thus, all honest nodes will have terminated by the end of epoch e. □

In Theorem 3.3.14, we proved that as soon as some epoch has an honest leader, all honest nodes will terminate at most 1 round after the epoch's end. Each epoch has $O(d) = O(n/h)$ number of rounds, and with random leader election, in expectation we need $O(n/h)$ number of rounds till we encounter an honest leader. Thus, the expected round complexity is $O((n/h)^2)$. We can also show that with probability $1 - \delta$, the round complexity is bounded by $\log(\frac{1}{\delta}) \cdot \frac{n}{h} / \log(\frac{1}{1-h/n})$.

The total number of consensus messages generated by honest nodes in each epoch (not counting implicit echoing) is at most $O(n)$. Each message is at most $\widetilde{O}(n)$ in size (the $\widetilde{O}$ hides the $\log n$ terms needed to encode a node's identifier). Each such consensus message will be delivered to $O(n)$ nodes and each node will echo every fresh message to everyone. Therefore, the total amount of communication pertaining to the consensus module (including implicit echoing of consensus messages) is $\widetilde{O}(n^4)$ if everyone behaved honestly. On top of this, honest nodes also need to echo messages sent by corrupt nodes and there can be (unbounded) polynomially many such messages. However, we can easily make the following optimization: for consensus messages with the same type and same epoch, every honest node echoes at most two messages originating from the same node (note that this is sufficient to form an equivocation evidence to implicate the sender). With this optimization, the per-epoch total communication for sending consensus messages is upper bounded by $\widetilde{O}(n^4)$. As mentioned earlier in Section 3.1, the total amount of communication for the trust graph module is also upper bounded by $\widetilde{O}(n^4)$. Thus, the total communication is upper bounded by $\widetilde{O}(n^4 \cdot E)$ where E denotes the number of epochs till termination. Note that in expectation $E = n/h$; moreover, with probability $1 - \delta$, E is upper bounded by $\log(\frac{1}{\delta}) / \log(\frac{1}{1-h/n})$.

Theorem 3.3.15. *The protocol described in this section (with an idealized leader election oracle) achieves Byzantine Broadcast in expected $O((n/h)^2)$ number of rounds.*

Proof. Follows directly from Theorems 5.3.13, 5.3.14 and 3.3.14. □

3.3.6 Instantiating Leader Election under a Static Adversary

So far we assumed an ideal leader election oracle. We can instantiate this leader election oracle using known cryptographic tools and obtain a protocol secure under the static corruption model.

We first explain a simple approach for instantiating the leader election oracle assuming that corruption decisions are made statically, i.e., before the protocol starts.

The approach is the following. First, the adversary decides who to corrupt. Next, a common random string $\mathsf{crs} \in \{0,1\}^\lambda$ is chosen where λ denotes a security parameter. Then, the protocol execution begins. In each epoch $e > 1$, the leader L_e is computed as follows where PRF denotes a pseudo-random function:

$$\text{For } e > 1: \quad L_e := (\mathsf{PRF}_{\mathsf{crs}}(e) \mod n) + 1.$$

Note that in the above, it may seem counter-intuitive that the PRF's secret key crs is publicly known. This is because a static adversary selects the corrupted set before crs is generated. Therefore, the adversary cannot adaptively corrupt the elected leader even if crs is publicly known. The random variable we care about bounding is the number of rounds till we encounter an honest leader. We want to show that the random variables in the ideal and real protocols are computationally indistinguishable.

Henceforth, we use Π_{ideal} to denote the protocol in Section 3.3, and we use Π_{real} to denote the same protocol, but instantiating the leader election as above. Let R_{ideal} be the random variable denoting the number of rounds till we have an honest leader in an execution of Π_{ideal}, and let R_{real} be the random variable denoting the number of rounds till we have an honest leader in an execution of Π_{real}.

Lemma 3.3.16. *R_{ideal} and R_{real} are computationally indistinguishable.*

Proof. Π_{ideal} is essentially the same as Π_{real} but where the PRF is replaced with a random function — notice also that under the static corruption model, publicly announcing the leader schedule after the adversary determines which nodes to corrupt does not affect the random variable R_{ideal}.

Suppose that R_{ideal} and R_{real} are computationally distinguishable. This means that there is an efficient distinguisher $\mathcal{D}$ which, knowing the identities of the corrupt nodes and the sequence of leaders chosen, can tell whether Π_{ideal} or Π_{real} is executing. Now, we can construct an efficient reduction $\mathcal{R}$ that can distinguish with non-negligible probability whether the oracle it is interacting with is a PRF or a random function. To do so, the reduction can interact with the adversary to learn which nodes it wants to corrupt. Then, it queries the oracle to obtain the sequence of leaders. It gives the identities of corrupt nodes and the sequence of leaders to $\mathcal{D}$ and outputs the same bit as $\mathcal{D}$. This contradicts the PRF's definition that a PRF is indistinguishable from a random function. Therefore, R_{ideal} and R_{real} are computationally indistinguishable. □

Therefore, we have the following theorem for the above real-world protocol Π_{real}.

Theorem 3.3.17. *Assume the static corruption model and that the* PRF *adopted is secure. Then, the aforementioned Π_{real} protocol achieves Byzantine Broadcast in expected $O((n/h)^2)$ number of rounds.*

Proof. Follows directly from Theorem 3.3.15 and Lemma 3.3.16. □

3.4 Achieving Security under a Weakly Adaptive Adversary

In this section, we show how to change the protocol in Section 3.3 such that it achieves security even under a weakly adaptive adversary. The weakly adaptive adversary can corrupt arbitrary nodes during any round of the protocol, as long as the total number of nodes it corrupts does not exceed a given upper bound f. However, when the adversary corrupts a node u in round r, it cannot erase the message u has already sent in round r.

Let us first see why the protocol in Section 3.3 fails to work under such a weakly adaptive adversary. Suppose an honest leader proposes a bit $b \in \{0, 1\}$ to all other nodes in the propose phase. Upon receiving the proposal, the adaptive adversary will learn the leader's identity. It can then corrupt the leader and generate an equivocating proposal, i.e., a proposal on $1 - b$. By sending equivocating proposals to all other nodes, it forces all nodes to remove the leader from their trust graphs. Thus, no one will commit / terminate during this epoch. By performing the above action repeatedly in each epoch, the adversary can make the protocol last for at least f epochs. To defend against a weakly adaptive adversary, we make two fundamental changes to the protocol:

- *Postpone leader election*: In the propose phase, instead of selecting a leader and let the leader propose, every node pretends to be the leader and sends its own proposal using Gossip. After all nodes have trustcast their proposals, we elect a leader using a verifiable random function (definition provided in Section 2.1.5). Nodes will then focus on the leader's proposal and ignore proposals from all other nodes. Note that it is possible for honest nodes to have inconsistent views on who the leader is. However, we will show that this does not affect the correctness of our protocol.

- *Make proposals unforgeable even after corrupting the leader*: If an node u proposes a bit $b \in \{0, 1\}$ and u remains uncorrupted throughout the propose phase, then the adversary cannot forge a proposal on $1 - b$ from u even if the adversary immediately corrupts u after the propose phase.

At the high level, we want an honest leader to fulfill its duty before its identity is known to the adversary. By postponing leader election, the adversary cannot learn which node would be elected until all nodes have trustcast their proposals. We also need to guarantee that as long as the leader performs honestly before revealing its identity (the leader election), then all honest nodes will commit on the leader's proposal – despite potential malicious behavior from the leader after its identity is revealed. To do this, we need to make sure that adversary cannot forge equivocating proposals from the leader after the propose phase. This is achieved by introducing a new building block called GossipAck, which is a simple extension of the previous Gossip. In GossipAck, a sender $s \in [n]$ first trustcasts a message to everyone. Upon receiving the message from the sender, a node trustcasts an acknowledgement claiming to receive the message, which we call an *ACK* message. At the end of the GossipAck protocol, we guarantee that for any honest node u, either

- s is no longer in u's trust graph, or

- u has received a unique and valid message from s; moreover, u has heard an ACK for the same message from every node in u's trust graph.

During the propose phase, we require that each node propose using GossipAck instead of Gossip. This forces a valid proposal to contain an ACK message from every node. In this way, although the adversary may immediately corrupt the leader after the leader election, it is too late for the now-corrupt leader to insert a new equivocating proposal, since there is no time for this new equivocating proposal to acquire ACKs from everyone.

We will apply a VRF as a cryptographic primitive in the leader election (defined in Section 2.1.5). In Section 3.4.1, we introduce our GossipAck protocol. Finally, in Section 3.4.2, we introduce our Byzantine Broadcast protocol under a weakly adaptive adversary. It is followed by proof of correctness that is structured similar to Section 3.3.

3.4.1 New Building Block: GossipAck Protocol

We describe an additional helpful building block called GossipAck that is a simple extension of the previous Gossip. In GossipAck, a sender $s \in [n]$ trustcasts a message and then every node acknowledges (ACK) the message (also using Gossip). If a node receives ACKs on different messages, which must contain equivocating signatures from the sender, then the sender must be corrupted and can be removed from the trust graph. At the end of the GossipAck protocol, we guarantee that for any honest node u, either

- s is no longer in u's trust graph, or
- u has received a unique and valid message from s; moreover, u has heard an ACK for the same message from every node in u's trust graph.

$\mathsf{GossipAck}^{\mathsf{Vf},s}$:

1. The sender $s \in [n]$ trustcasts the message m with $\mathsf{Gossip}^{\mathsf{Vf},s}$.
2. For every honest node $u \in [n]$,
   - if the previous $\mathsf{Gossip}^{\mathsf{Vf},s}$ outputs a message m signed by s such that $u.\mathsf{Vf}(m) = \mathsf{true}$, then set $m' \leftarrow (\mathtt{ack}, s, m)$.
   - else, set $m' \leftarrow (\mathtt{ack}, s, \perp)$.

   u trustcasts the message m' with $\mathsf{Gossip}^{\mathsf{Vf}',u}$, where $v.\mathsf{Vf}'(\mathtt{ack}, s, m) = \mathsf{true}$ in round r iff

   (a) either s is no longer in G_v^r or m must agree with what s has trustcast (in v's view); and

   (b) either $m = \perp$ or $v.\mathsf{Vf}(m) = \mathsf{true}$ in round r.

Fact 3.4.1. *If* Vf *satisfies the monotonicity condition, then* Vf' *also satisfies the monotonicity condition.*

Proof. Recall that for any node u, $u.\mathsf{Vf}'(\mathtt{ack}, s, m) = \mathsf{true}$ in round r iff

(a) either s is no longer in G_u^r or m must agree with what s has trustcast (in u's view); and

(b) either $m = \perp$ or $u.\mathsf{Vf}(m) = \mathsf{true}$ in round r.

Let u, v be honest nodes, and let $r < t$. Suppose $u.\mathsf{Vf}'(\mathtt{ack}, s, m) = \mathsf{true}$ in round r, we want to show that $v.\mathsf{Vf}'(\mathtt{ack}, s, m) = \mathsf{true}$ in round t. Let us consider consider conditions (a) and (b) separately. Condition (b) holds for v because of the monotonicity of Vf. We now focus on condition (a).

Since $u.\mathsf{Vf}'(\mathtt{ack}, s, m) = \mathsf{true}$ in round r, it must be that in round r, either $s \notin G_u^r$, or $s \in G_u^r$ and the message m is what u heard s trustcast. If the former holds, then in round t, $s \notin G_v^t$ by the trust graph's monotonicity invariant, and thus condition (a) holds for v in round t. If the latter holds, then if in round t, $s \notin G_v^t$, condition (a) holds for v in round t. We thus focus on the case when

$s \in G_v^t$. In this case, it must be that v heard s trustcast the same message, since otherwise u would have sent to v the equivocating message trustcast by s, and v would have removed s from its trust graph. This implies that condition (a) holds for v in round t. Thus, we conclude that in round t, $(\texttt{ack}, s, m)$ must pass the verification check of Vf' w.r.t. the node v. □

Fact 3.4.2. Vf' *satisfies the validity at origin property, i.e., when an honest node* u *trustcasts* $(\texttt{ack}, s, m)$ *with* $\mathsf{Gossip}^{\mathsf{Vf}',u}$ *in some round* r *during* $\mathsf{GossipAck}^{\mathsf{Vf},s}$*, it must be that* $u.\mathsf{Vf}'(\texttt{ack}, s, m) = \mathsf{true}$ *in round* r*.*

Proof. By construction and by property of the first $\mathsf{Gossip}^{\mathsf{Vf},s}$. □

Lemma 3.4.3. *Assume that* Vf *satisfies the monotonicity condition and that the message* m *input to* s *satisfies the validity at origin condition. Then, at the end of the* $\mathsf{GossipAck}$ *protocol, for any honest node* u*, either*

1. *s is no longer in u's trust graph, or*

2. *u has heard s trustcast a unique message m such that $u.\mathsf{Vf}(m) = \mathsf{true}$ at the end of the protocol; moreover, u must have received an ACK of the form $(\texttt{ack}, s, m)$ from every node in its trust graph.*

Proof. Since Vf satisfies the monotonicity condition and the validity at origin condition, by Fact 3.4.1 and Fact 3.4.2, Vf' also satisfies the monotonicity condition and the validity at origin condition. Thus, by Theorem 3.2.1, for any two honest nodes u and v, u must have received a valid ACK from v that passes Vf' in $\mathsf{Gossip}^{\mathsf{Vf}',v}$. In other words, by the end of the $\mathsf{GossipAck}$ protocol, any honest node u must have received valid ACKs that each passes Vf' from all nodes in its trust graph.

Now, we want to show that the ACKs u receives match what s trustcasts (in u's view). If s is still in u's trust graph by the end of $\mathsf{GossipAck}$, again by Theorem 3.2.1, u must have received a valid message from s. By the definition of Vf', each ACK message received by u must agree with what s has trustcast. This completes our proof. □

3.4.2 Protocol

Strawman attempt. We first discuss a strawman attempt using a Verifiable Random Function (VRF) that achieves security against a weakly adaptive adversary. In every epoch, every node u computes $(y_u, \pi_u) := \mathsf{VRF}(\mathsf{crs}, \mathsf{sk}_u, e)$. Now, y_u is said to be u's charisma and we define the leader to be the node with the maximum charisma. The issue with this approach is that the adversary, upon observing that u has the maximum charisma and is the leader, can immediately corrupt u, and make u send an equivocating proposal. Such an attack will not affect consistency, however, it will hamper liveness due to the following: every honest node, upon seeing u's equivocating proposal, removes u from its trust graph; and now they would vote for $\perp$ in the Vote phase. An adversary with a corruption budget of f can continue this attack for f epochs in which an honest node becomes the leader, and thus liveness can take as long as $\Omega(f)$ epochs to ensue.

To defeat the aforementioned attack, we are inspired by techniques from the standard Byzantine Broadcast literature [10], [13] but it is not so trivial to adapt these techniques to our setting. At a high level, during the Propose phase of each epoch, everyone multicasts a proposal using $\mathsf{GossipAck}$ pretending that it might be the elected leader. Because $\mathsf{GossipAck}$ rather than Gossip is used, effectively everyone would also trustcast an ACK for everyone's proposal. Note that at this time, the VRF outcomes have not been revealed and the adversary cannot effectively single out and target the leader.

Our key idea is to require that a valid proposal must contain everyone's ACK message. In this way, when nodes reveal their VRF outcomes (i.e., their charisma), the adversary may immediately corrupt the leader, but it is already too late for the now-corrupt leader to insert a new equivocating proposal, because there is no time for this new equivocating proposal to acquire ACKs from everyone. To integrate this idea into our protocol involves further technicalities and subtleties, but with this intuition in mind, we can now present our protocol formally.

Commit evidence. Commit evidence is defined in the same way as in Section 3.3 except that our new vote message has a few more terms (which will become clear shortly) — but we simply ignore these additional terms when defining a commit evidence. Concretely, fix an epoch e and a bit $b \in \{0, 1\}$. We say that a collection $\mathcal{E}$ containing signed messages of the form $(\mathtt{vote}, e, b, \_, \_, \_)$ is an epoch-e commit evidence for b w.r.t. G_u^r iff for every $v \in G_u^r$, $\mathcal{E}$ contains a signed message $(\mathtt{vote}, e, b, \_, \_, \_)$ from v. We also call an epoch-e commit evidence for b w.r.t. G_u^r "a commit evidence for (e, b) w.r.t. G_u^r".

Protocol. Our protocol is described below. Note that although the protocol seems complicated, most of it is the same as the protocol in Section 3.3. The main difference lays in how nodes propose messages and elect the leader.

Setup. Run $(\mathsf{crs}, \{\mathsf{pk}_u, \mathsf{sk}_u\}_{u\in[n]}) \leftarrow \mathsf{VRF.Gen}(1^\lambda)$. Publish $(\mathsf{crs}, \mathsf{pk}_1, \dots, \mathsf{pk}_n)$ and give sk_u to each $u \in [n]$.

Assumption. For the initial sender $s \in [n]$ in epoch 1, we redefine the outcome of $\mathsf{VRF.Eval}(\mathsf{crs}, \mathsf{sk}_s, 1)$ to be $(\infty, \perp)$, and we assume that $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_s, 1, \infty, \perp) = 1$. This makes sure that the initial sender s has the maximum charisma in epoch $e = 1$. We shall also assume that by construction, the function VRF will append to the outcome y the unique identifier of the node u. In this way, the evaluation outcomes for two different nodes must be *distinct*.

Main Protocol. For each epoch $e = 1, 2, \dots,$

1. **Propose**: ($O(d)$ rounds) Every node $u \in [n]$ performs the following:
   - Choose a bit to propose and an evidence as follows:
     - If $e = 1$ and u is the initial sender, u chooses $P := (b, \perp)$ where b is its input bit.
     - Else if a non-$\perp$ commit evidence (for some bit) has been seen, let $\mathcal{E}(e, b)$ denote the freshest such commit evidence and let $P := (b, \mathcal{E}(e, b))$.
     - Else, u chooses a random bit b and let $P := (b, \perp)$.
   - u ackcasts the proposal $(\mathtt{prop}, e, P)$ by calling $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, u}$ where

$$v.\mathsf{Vf}_{\mathrm{prop}}(\mathtt{prop}, e, (b, \mathcal{E})) = \mathsf{true} \text{ in round } r \text{ iff}$$

(a) $\mathcal{E}$ is a valid commit evidence vouching for the bit b proposed; and

(b) for every $w \in G_v^r$, $\mathcal{E}$ is at least as fresh as any commit evidence trustcast by w in the **Commit** phase of *all* previous epochs $e' < e$ — recall that $\perp$ may be treated as a commit evidence for epoch 0.

2. **Elect**: (1 round) Every node $u \in [n]$ computes $(y, \pi) := \mathsf{VRF.Eval}(\mathsf{crs}, \mathsf{sk}_u, e)$, and sends the signed tuple $(\mathtt{elect}, e, y, \pi)$ to everyone.

3. **Prepare**: $(O(d)$ rounds) Every node $u \in [n]$ does the following:

   - Let $S \subseteq [n]$ be the set of nodes v satisfying the following: [a]

     (a) u has received from v a signed tuple of the form $(\mathtt{elect}, e, y_v, \pi_v)$ where $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_v, e, y_v, \pi_v) = 1$ — henceforth, y_v is said to be v's charisma;

     (b) u has received from v a signed proposal of the form $(\mathtt{prop}, e, (b, \_))$. Moreover, everyone that remains in u's trust graph has ACKed this proposal in$\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, v}$ earlier.

   - Find the node $L \in S$ whose charisma y_L is maximized based on lexicographical ordering.

   - Trustcast the tuple $(\mathtt{prep}, e, b, L, y_L, \pi_L)$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{prep}}, u}$ where

$$v.\mathsf{Vf}_{\mathrm{prep}}(\mathtt{prep}, e, b, L, y, \pi) = \mathsf{true\ in\ round\ } r \mathsf{\ iff}$$

     (a) Everyone in G_v^r has ACKed a proposal of the form $(\mathtt{prop}, e, (b, \_))$ by the end of the $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, L}$ instance; and

     (b) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_L, e, y, \pi) = 1$.

   Henceforth, given a prepare message of the form $(\mathtt{prep}, e, b, L, y, \pi)$, y is said to be the charisma of the prepare message.

4. **Vote**: $(O(d)$ rounds) Every node $u \in [n]$ performs the following:

   - Compare the $(\mathtt{prep}, e, \_, \_, \_, \_)$ messages that have been trustcast by all nodes v that still remain in u's trust graph, and pick the one $(\mathtt{prep}, e, b^*, L^*, y^*, \pi^*)$ whose charisma value y^* is the maximum.

   - Trustcast a vote of the form $(\mathtt{vote}, e, (b^*, L^*, y^*, \pi^*))$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{vote}}, u}$ where

$$v.\mathsf{Vf}_{\mathrm{vote}}(\mathtt{vote}, e, (b, L, y, \pi)) = \mathsf{true\ in\ round\ } r \mathsf{\ iff}$$

     (a) Everyone in G_v^r has ACKed a proposal for b signed by L by the end of the $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, L}$;

     (b) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_L, e, y, \pi) = 1$;

     (c) For everyone $w \in G_v^r$, y must be at least as large as the charisma of the prepare message trustcast to v by w.

5. **Commit**: ($O(d)$ rounds) Every node $u \in [n]$ performs the following:

   - If everyone still in u's trust graph voted for the same bit $b \in \{0, 1\}$ (as defined by the outputs of the $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{vote}}, u}$ protocols during the **Vote** phase), then **output** the bit b, and trustcast a commit message $(\texttt{comm}, e, \mathcal{E})$ by calling $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{comm}}, u}$, where $\mathcal{E}$ contains a signed vote message of the form $(\texttt{vote}, e, (b, \_))$ from everyone in u's trust graph.
   - Else, use $\mathsf{Gossip}^{\mathsf{Vf}_{\mathrm{comm}}, u}$ to trustcast the message $(\texttt{comm}, e, \bot)$.

   We define the verification function $\mathsf{Vf}_{\mathrm{comm}}$ below. $v.\mathsf{Vf}_{\mathrm{comm}}(\texttt{comm}, e, \mathcal{E}) = \mathsf{true}$ in round r iff:

   (a) either v has seen a tuple $(\texttt{elect}, e, y, \pi)$ signed by some $w \notin G_v^r$ such that (1) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_w, e, y, \pi) = 1$, and (2) for everyone $w' \in G_v^r \cup S$, y is greater than the charisma of the prepare message trustcast by w'.

   (b) or $\mathcal{E}$ must be a valid epoch-e commit evidence.

Terminate: Same as in Section 3.3.

[a]We want to make sure that as long as a node remains honest in the propose phase, it will be in the set S of any honest node u.

3.4.3 Proof of Correctness for the Verification Functions

Lemma 3.3.2 and Fact 3.3.5 still hold in our new protocol and the proofs are the same as before. This means that Vf_{prop} satisfies the monotonicity condition and the validity at origin condition. We now check that the other verification functions satisfy the two conditions as well.

Lemma 3.4.4. $\mathsf{Vf}_{\mathrm{prep}}$ *satisfies the monotonicity condition.*

Proof. Recall that a prepare message $(\texttt{prep}, e, b, L, y_L, \pi_L)$ passes $\mathsf{Vf}_{\mathrm{prep}}$ w.r.t. node u in round r iff

(a) Everyone in G_u^r has ACKed a proposal of the form $(\texttt{prop}, e, (b, \_))$ by the end of the $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, L}$ instance; and

(b) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_L, e, y, \pi) = 1$.

Condition (b) clearly satisfies the monotonicity condition. We now focus on condition (a). Let u, v be honest and let $t > r$. Recall that $G_v^t \subseteq G_u^r$ by trust graph monotonicity. Suppose that condition (a) is satisfied for some message w.r.t. G_u^r, it suffices to show that v has heard every $w \in G_v^t$ ACK the same proposal in the $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, L}$ instance as what u heard from w. If this is not true, then u would have relayed the equivocating message to v and v would have already removed w from its trust graph by the beginning of round t. □

Lemma 3.4.5. $\mathsf{Vf}_{\mathrm{vote}}$ *satisfies the monotonicity condition.*

Proof. Recall that a vote message $(\texttt{vote}, e, (b, L, y, \pi))$ passes $\mathsf{Vf}_{\mathrm{vote}}$ w.r.t. node u in round r iff

(a) Everyone in G_u^r has ACKed a proposal for b signed by L by the end of the $\mathsf{GossipAck}^{\mathsf{Vf}_{\mathrm{prop}}, L}$; and

(b) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_L, e, y, \pi) = 1$; and

(c) For everyone $w \in G^r_u$, y must be at least as large as the charisma of the prepare message trustcast to u by w.

In Lemma 3.4.4, we have already shown that conditions (a) and (b) satisfy the monotonicity condition. We now focus on condition (c). Suppose that u and v are honest and $t > r$. Recall that $G^t_v \subseteq G^r_u$ by the trust graph monotonicity. Suppose that condition (c) is satisfied for some message w.r.t. G^r_u, it suffices to show that the same y contained in the message is at least as large as the charisma of the prepare message trustcast to v by any $w \in G^t_v$. This follows since v must have heard any $w \in G^t_v$ trustcast the same prepare message as what u heard w trustcast; otherwise, u would have relayed the equivocating message to v and v would have removed w from its trust graph by the beginning of round t. ☐

Lemma 3.4.6. $\mathsf{Vf}_{\mathrm{comm}}$ *satisfies the monotonicity condition.*

Proof. A commit message $(\mathtt{comm}, e, \mathcal{E}) = \mathsf{true}$ passes $\mathsf{Vf}_{\mathrm{comm}}$ w.r.t. node u in round r iff

1. either v has seen a tuple $(\mathtt{elect}, e, y, \pi)$ signed by some $w \notin G^r_v$ such that (1) $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_w, e, y, \pi) = 1$, and (2) for everyone $w' \in G^r_v \cup S$, y is greater than the charisma of the prepare message trustcast by w'.
2. or $\mathcal{E}$ must be a valid epoch-e commit evidence.

The monotonicity of condition (a) follows from trust graph monotonicity as well as the implicit echoing assumption. The monotonicity of condition (b) follows from the monotonicity of commit evidences. ☐

Next, we show that the verification functions satisfy the validity at origin condition.

Fact 3.4.7. *If an honest node u trustcasts a $(\mathtt{prop}, e, P)$, $(\mathtt{prep}, e, P)$ or a $(\mathtt{vote}, e, b)$ message in some round r, the trustcast message satisfies the corresponding verification function,* $\mathsf{Vf}_{\mathrm{prop}}$, $\mathsf{Vf}_{\mathrm{prep}}$, $\mathsf{Vf}_{\mathrm{vote}}$, *respectively, w.r.t. the node u in round r.*

It can be verified that $\mathsf{Vf}_{\mathrm{prop}}$, $\mathsf{Vf}_{\mathrm{prep}}$ and $\mathsf{Vf}_{\mathrm{vote}}$ satisfy the validity at origin condition by construction. The verification functions are designed such that they will always accept the trustcast messages. The validity at origin condition for $\mathsf{Vf}_{\mathrm{comm}}$, on the other hand, is not as simple.

Lemma 3.4.8. *At the beginning of the* **Commit** *phase of some epoch e, for an honest node u, if the largest charisma u has seen from any elect message, i.e.,*

$$\max_y \{\textit{elect messages } (\mathtt{elect}, e, y, \pi) \textit{ that } u \textit{ has seen such that } \mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_w, e, y, \pi) = 1\},$$

comes from the prepare message trustcast by some w in u's trust graph. Then, every w' who still remains u's trust graph must have trustcast a vote for the same bit b that u has voted for in this epoch.

Proof. We can prove by contradiction. Suppose that the premise holds but some v in u's trust graph has successfully trustcast a vote of the form $(\mathtt{vote}, e, (1 - b, L, y', \pi'))$to u. Since $\mathsf{Vf}_{\mathrm{vote}}$ accepts v's vote message w.r.t. u, by the definition of $\mathsf{Vf}_{\mathrm{vote}}$, we have

- $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_L, e, y', \pi') = 1$, and,
- for every w in u's trust graph, y' must be at least as large as the charisma of the prepare message trustcast to u by w.

Since the premise holds, y' must be the largest charisma u has seen for any node. By definition of the protocol, u must have voted for the bit $1 - b$ too. Thus, we have reached a contradiction. □

Fact 3.4.9. *If an honest node u trustcasts a $(\texttt{comm}, e, \mathcal{E})$ message in some round r, the trustcast message satisfies the verification function, $\mathsf{Vf}_{\text{comm}}$ w.r.t. the node u in round r.*

Proof. Follows from Lemma 3.4.8. □

Therefore, all the verification functions used in the protocol satisfy the monotonicity condition and the validity at origin condition.

3.4.4 Consistency, Liveness and Validity Proof

We first show that our protocol achieves consistency. Compared to the protocol in Section 3.3.2, the changes we made to the protocol in Section 3.4.2 does not affect its consistency. Thus, the consistency proof is exactly the same as before (see Section 3.3.4). We briefly provide a proof sketch below.

Theorem 3.4.10 (Consistency)**.** *The protocol described in Section 3.4.2 satisfies consistency.*

Proof. In Section 3.3, we proved consistency using Lemma 3.3.9 and Lemma 3.3.10.

- Lemma 3.3.9: If an honest node $u \in [n]$ sees an epoch-e commit evidence for the bit $b \in \{0, 1\}$ in some round r, and an honest node $v \in [n]$ sees an epoch-e commit evidence for the bit $b' \in \{0, 1\}$ in some round t, it must be that $b = b'$.
- Lemma 3.3.10: If an honest node $u \in [n]$ outputs the bit b in some epoch e, then in every epoch $e' > e$, no honest node $v \in [n]$ can ever see a commit evidence for $(e', 1 - b)$.

The two lemmas hold for the protocol in this Section 3.4.2 as well. The proofs for the two lemmas remain unchanged. Therefore, any two honest nodes cannot output different bits. □

Next, we show that our protocol achieves liveness and terminates in expected constant rounds. Observe that Fact 3.3.13 of Section 3.3.5 still holds due to the same argument as before. We restate it as follows.

Fact 3.3.13. *If some honest node terminates in round r, then all honest nodes will have terminated by the end of round $r + 1$.*

During the execution, even before nodes reveal their charisma for some epoch e, we can already define a node u's epoch-e charisma as the honestly computed VRF outcome $\mathsf{VRF.Eval}(\mathsf{crs}, \mathsf{sk}_u, e)$. This definition is well-formed no matter whether the node is honest or corrupt.

Definition 3.4.11 (Lucky epoch)**.** Henceforth, we say that epoch e is *lucky* iff the node with the maximum epoch-e charisma has not been corrupted until it has sent a signed $(\texttt{elect}, e, \_, \_)$ message.

Lemma 3.4.12. *Suppose that the VRF satisfies unforgeability. Except with negligible probability, the following holds: if e is a lucky epoch, then one round after the end of epoch e, all honest nodes will terminate.*

Proof. By Fact 3.3.13, if any honest node terminates during epoch e, all honest nodes will terminate in the next round. Therefore, it suffices to prove the lemma assuming that no honest node has terminated by the end of epoch e, i.e., we may assume that all honest nodes will participate in all the Gossip protocols till the end of epoch e. Thus, we can safely use the properties of Gossip.

Suppose epoch e is a lucky epoch. This means that the node L with the maximum epoch-e charisma has not been corrupted until it has sent the signed elect message $(\texttt{elect}, e, y, \pi)$. Since the weakly adaptive adversary cannot remove messages already sent, all honest nodes will receive the elect message. Every honest node will then trustcast $(\texttt{prep}, e, b, L, y, \pi)$ where b is L's proposed bit in the **Propose** phase of epoch e. Since L remains honest throughout the propose phase, all nodes have a consistent view on L's proposed bit and they can only Ack b. After the propose phase, honest nodes no longer Ack any new proposal. Thus, even if L becomes corrupt immediately after sending the elect message, it cannot gather Ack messages from honest nodes on the bit $1 - b$.

By Theorem 3.2.1, after the **Vote** phase, for any two honest nodes u and v, u must have received a vote from v that passes $\mathsf{Vf}_{\text{vote}}$. Due to condition (c) of the $\mathsf{Vf}_{\text{vote}}$ check, the vote from v must vote for the same bit b that L proposes. Thus, at the end of the **Vote** phase, any honest node would receive votes on b from every node in its trust graph, which forms a commit evidence for b.

Again, by Theorem 3.2.1, after the **Commit** phase, for any two honest nodes u and v, u must have received a commit message from v that passes $\mathsf{Vf}_{\text{comm}}$. Recall that a $\perp$ commit message passes $\mathsf{Vf}_{\text{comm}}$ w.r.t. node u iff there exists a node not in u's trust graph, whose charisma is greater than all the prepare messages u has received. However, since epoch e is a lucky epoch, all honest nodes generate their prepare messages from L, which has the largest charisma in epoch e. Therefore, all the commit messages received must be non-$\perp$ commit messages. And since all honest nodes vote on b in the **Vote** phase, the commit message must be on b. In conclusion, unless the adversary can successfully forge a VRF result which happens with negligible probability, any honest node will receive commit messages from every node in its trust graph on b. This satisfies the termination condition and all honest nodes will terminate.

□

Lemma 3.4.13. *Suppose that the VRF satisfies pseudorandomness under selective opening. Then, let $\mathbf{R}_{\text{lucky}}$ be a random variable denoting the first lucky epoch. It must be that there is a negligible function $\mathsf{negl}(\cdot)$ such that for every R,*

$$\Pr[\mathbf{R}_{\text{lucky}} \geq R] \leq \Pr[\mathsf{Geom}(h/n) \geq R] + \mathsf{negl}(\lambda)$$

where $\mathsf{Geom}(h/n)$ denotes a geometric random variable with probability h/n.

Proof. We can consider an ideal-world protocol which is defined just like the real-world protocol except for the following: whenever a node needs to compute $\mathsf{VRF.Eval}(\mathsf{crs}, \mathsf{pk}_u, e)$, it will instead call an ideal functionality $\mathcal{F}.\mathsf{Eval}(u, e)$. Upon receiving this call, $\mathcal{F}$ picks y, at random if this is the first time $\mathsf{Eval}(u, e)$ is queried, and records the tuple (u, e, y). Now $\mathcal{F}$ returns the answer y that has been recorded for the query (u, e), and the tuple $(y, \perp)$ will be used in place of the outcome of the VRF evaluation. Similarly, whenever a node needs to call $\mathsf{VRF.Vf}(\mathsf{crs}, \mathsf{pk}_u, e, y, \pi)$, the call is replaced with a call to $\mathcal{F}.\mathsf{Vf}(u, e, y)$, which simply checks if the tuple (u, e, y) has been recorded — if so, return 1; else return 0.

In this ideal-world protocol, since leaders are elected completely randomly, it is not hard to see that $\Pr[\mathbf{R}_{\text{lucky}} \geq R] = \Pr[\mathsf{Geom}(h/n) \geq R]$.

By the definition of VRF, it is impossible to distinguish between the results of VRFs (the real-world protocol) and uniformly random distribution (the ideal-world protocol) for any polynomially bounded adversary. It thus follows that in the real-world protocol,

$$\Pr[\mathbf{R}_{\text{lucky}} \geq R] \leq \Pr[\mathsf{Geom}(h/n) \geq R] + \mathsf{negl}(\lambda)$$

as long as the adversary is polynomially bounded.

□

Theorem 3.4.14 (Liveness)**.** *Assume that the VRF adopted satisfies pseudorandomness under selective opening and unforgeability. Then, the protocol described in Section 3.4.2 achieves liveness in $O((n/h)^2)$ number of rounds.*

Proof. Follows directly from Lemma 3.4.12 and Lemma 3.4.13. □

Finally, we show that the protocol also achieves validity. Note that the definition of validity needs to be slightly adjusted under the weakly adaptive adversary model. Since a weakly adaptive adversary can corrupt arbitrary nodes at any time, the validity requirement only makes sense when the initial sender remains honest throughout the entire protocol.

Theorem 3.4.15 (Validity)**.** *Assume that the VRF adopted satisfies unforgeability. For the protocol described in Section 3.4.2, the following holds except with negligible probability: if the designated sender s is (forever) honest, then everyone will output the sender's input bit.*

Proof. Recall that by our construction, s is guaranteed to have the maximum charisma in epoch 1. The proof of Lemma 3.4.12 implies that if s is (forever) honest, at most one round after epoch $e = 1$, all honest nodes will have terminated with an output that agrees with s's proposal. □

3.5 Conclusion

In this chapter, we presented a Byzantine Broadcast protocol with amortized $O(1)$ round complexity that works even under dishonest majority. The round complexity is constant and the communication complexity is $\widetilde{O}(n^4)$ (for the entire system). We believe this is the first protocol that gives constant round complexity for Byzantine Broadcast under dishonest majority.

It has been shown by Garay et al. [18] that no randomized protocol can achieve BB in less than $O(n/(n-f))$ number of rounds, even assuming static corruption and allowing standard setup assumptions. Therefore, for the (narrow) regime $n - f = o(n)$, there is still an asymptotical gap between our upper bound and their lower bound. Bridging this gap is an exciting direction for future work.

Chapter 4

Communication Complexity of BB protocol

We have presented the trust graph structure (Section 3.1) and utilized it to achieve expected constant round complexity under a weakly adaptive adversary and $f = (1 - \varepsilon)n$ corruption for any constant $\varepsilon > 0$. However, our BB protocol in Section 3.1 require $\tilde{O}(n^4)$ communication complexity. Even under honest majority, the state-of-the-art communication complexity is quadratic, which is not ideal for large distributed systems. In this chapter, we show how to utilize the trust graph idea to improve the communication complexity. Specifically, we show how to achieve (1) amortized quadratic communication complexity for any $f = (1 - \varepsilon)n$ and (2) amortized linear communication complexity for any $f < n$. Both results work under a strongly adaptive adversary.

Remark 7. *In this thesis, we consider both expected round complexity and amortized communication complexity. While the two words "expected" and "amortized" have similar meaning, the contexts in which these words are used are different. Our protocols are all randomized (it is well known that deterministic BB protocol requires at least $f + 1$ rounds). Therefore, expected round complexity is measured as the expectation against the randomness we used, in a single execution of the BB protocol. Amortized communication complexity, on the other hand, represents the average communication cost incurred in a significant long sequence of BB executions.*

4.1 Amortized Quadratic Communication under Dishonest Majority

Let us start with a simple BB protocol based on Trustcast (denoted as Gossip) and the Dolev-Strong protocol that achieves amortized $O(\kappa n^2)$ communication complexity for the dishonest majority case, i.e., $f < n$.

Overview. The state-of-the-art protocol for communication complexity under dishonest majority is the Dolev-Strong protocol, which costs cubic communication ($O(\kappa n^2 + n^3)$ with multi-signature or $O(\kappa n^3)$ with signature). Our first natural idea is, instead of directly agreeing on the sender's value, we use the Dolev-Strong protocol to agree on the sender's dishonesty when the sender misbehaves. Each time we call the Dolev-Strong protocol, at least one node will be proven corrupt and removed from the protocol. This way, the Dolev-Strong protocol is called at most $f + 1$ times across all instances.

Algorithm 4.1.1: Quadratic communication multi-shot BB

Let G_u be an undirected complete graph of n vertices. Each slot $k \geq 1$ takes $T = n + f + 3$ rounds. Each node u runs the following steps.

TrustCast. In round 0, start invoking $\mathsf{Gossip}(G_u, n, k)$ where S_k works as the sender S in Gossip. Let m be a value received from S_k (through $\langle \mathsf{prop}, m, k \rangle_{S_k}$) by the beginning of round n.

Dolev-Strong. In each round $n + 1 \leq t \leq n + f + 2$ run the following. Let $\tau = t - (n + 1)$, i.e., rounds after starting this phase.

- $\tau = 0$: if S_k is not in G_u, then multicast $\langle \mathsf{corrupt}, S_k \rangle_u$ (if not sent before).
- $1 \leq \tau \leq f + 1$: If u has received $\langle \mathsf{corrupt}, S_k \rangle_*$ signed by at least τ distinct nodes and S_k is not in G_u, then multicast them (those not sent before) and $\langle \mathsf{corrupt}, S_k \rangle_u$ (if not sent before).

Finally, in round $t = n + f + 2$, if u has not sent $\langle \mathsf{corrupt}, S_k \rangle_u$, then commit m for slot k. Otherwise, commit $\perp$. Note that each $\langle \mathsf{corrupt}, v \rangle_w$ message is shared among all slots and sent/forwarded only once.

Now, to agree on a dishonest sender, we have to provably detect the sender's misbehavior. For the honest majority case, more than f accusations from honest nodes work as a corrupt-proof. But this does not work directly for the dishonest majority case. We instead utilize the Trustcast protocol. Recall that for L instances of Trustcast with each node maintaining the trust graph across instances, the total communication cost is $O(\kappa n^2 L + \kappa n^4)$. In each instance, an honest node multicasts the sender's messages at most twice which costs $O(\kappa n^2)$. The cost of maintaining the trust graph is bounded by $O(\kappa n^4)$ across all instances, since for each edge in the trust graph, honest nodes multicast the accuse message at most once.

4.1.1 Our protocol

Our protocol is described in Algorithm 4.1.1. It consists of two phases: 1) the Trustcast protocol to receive the sender's message or create a corrupt-proof when the sender is silent, and 2) the Dolev-Strong-style protocol to agree on whether the sender is dishonest, which helps decide whether the sender's message (if received) is committed. Note that each node uses the same trust graph across all slots. So communication to maintain the trust graph and detect malicious senders is bounded and amortized over all slots. Likewise, each $\langle \mathsf{corrupt}, v \rangle_w$ message in the Dolev-Strong phase is shared among all slots and is sent/forwarded only once. So communication in the Dolev-Strong phase is also amortized over all slots.

4.1.2 Proof of correctness

We prove the correctness of Algorithm 4.1.1. Termination and sequentiality are obvious. Below, we say a node u *votes for the corruption of* S_k if u sends $\langle \mathsf{corrupt}, S_k \rangle_u$.

Lemma 4.1.1 (Validity). *If the sender S_k is honest, then all honest nodes commit the sender's message at slot k.*

Proof. Due to the honest clique invariant of Gossip, an honest sender never gets removed from any honest node's trust graph. So honest nodes never vote for the corruption of an honest sender. Thus, all honest nodes always commit the honest sender's message. □

Lemma 4.1.2 (Consistency). *If two honest users u and v commit m_u and m_v respectively, at slot k, then $m_u = m_v$.*

Proof. We first show that if an honest node u votes for the corruption of S_k, then any honest node v also votes for it.

Suppose u sends $\langle \mathsf{corrupt}, S_k \rangle_u$ before the last round (i.e., $\tau < f + 1$), then v receives it by the beginning of round $\tau + 1$. The node u must have removed S_k from G_u in round τ. Due to the monotonicity invariant of Gossip, v removes S_k from G_v in round $\tau + 1$. Also, u must have forwarded the corrupt messages from at least τ nodes, so v receives the corrupt messages from $\tau + 1$ distinct nodes by round $\tau + 1$. Thus, u also votes for the corruption of S_k.

Suppose u sends $\langle \mathsf{corrupt}, S_k \rangle_u$ at the last round (i.e., $\tau = f + 1$). Then, u must have received the corrupt messages from $f + 1$ distinct nodes, at least one of them must be from an honest node, which must have sent it before the last round. The analysis above implies v votes for the corruption of S_k.

Therefore, if an honest node commits $\perp$, then all honest nodes also commit $\perp$.

Suppose an honest node u commits $m \neq \perp$, then honest nodes do not commit $m' \neq m$; otherwise, these two different prop messages are forwarded to all honest nodes by the beginning of round $n + 1$ (when the Dolev-Strong phase starts), which would lead to all honest nodes voting for the sender's corruption and u would not commit m. Since none of the honest nodes could have voted for the sender's corruption, they must have received m by the termination property of Gossip, hence they all commit m. Therefore, all honest nodes commit the same value at the same slot. □

4.1.3 Communication complexity.

We analyze the communication complexity of Algorithm 4.1.1.

- In the Gossip protocol, an honest node multicasts the sender's messages at most twice. This adds up to $O(\kappa n^2)$ communication complexity per instance. Since we use the same trust graph for all slots, the cost of maintaining the trust graph is $O(\kappa n^4)$ across all instances.

- In the proof of Lemma 4.1.2, we showed that if any honest node sent a corrupt message during Dolev-Strong, then all honest nodes would remove the sender from their trust graphs. Therefore, there can be at most f instances where any honest node sends messages in the Dolev-Strong protocol. So the communication complexity for the Dolev-Strong protocol is upper bounded by $O(\kappa n^3 \cdot f)$ across all instances.

To sum up, the total cost is $O(\kappa n^2 L + \kappa n^4)$ for L slots. Note that the round complexity of Algorithm 4.1.1 is $\Theta(f)$, which is the same as the Dolev-Strong protocol. Since the dishonest majority setup is more on the theory side, we are interested in presenting the simplest protocol that can achieve quadratic communication complexity under dishonest majority. It remains an interesting problem whether we can combine Algorithm 4.1.1 with the protocol in Section 3.3.2 to achieve expected linear round complexity and amortized quadratic communication complexity at the same time.

4.2 Amortized Linear Communication under Honest Majority

In this section, we present a BB protocol that achieves amortized linear communication complexity under $f < (0.5 - \varepsilon)n$ corruption for any constant $\varepsilon > 0$. Our protocol is described in Algorithm 4.2. At a high level, our multi-shot BB protocol consists of multiple slots, where each slot implements a single-shot BB. Each slot progresses through many *epochs*, with each epoch having a unique *leader*. Similar to Section 3, a leader proposes a value, other nodes vote for it, and nodes commit it after collecting enough votes.

Note that while the protocol still follows the high-level idea of trust graphs and Trustcast, it is easier to explain the protocol from first principles rather than using Trustcast as a black box. This is because Trustcast is designed for the dishonest majority case. Using Trustcast as a black box under the honest majority scenario will add unnecessary steps and complicate the protocol. Before explaining in detail, we first define some notions and notations used in our protocol.

Definition and notations. For each slot $k \geq 1$, we have at most $f + 2$ epochs, each taking 11 rounds. Epoch i of slot k starts in round $t = 11((k - 1)(f + 2) + i)$. The leader L_0 of the first epoch $i = 0$ is the sender S_k, and the leader L_i of epoch $1 \leq i \leq f + 1$ is node i.

To reduce message size, we use a threshold signature scheme to combine a set of votes into a *certificate*. A certificate for a value m in epoch i of slot k, denoted $\mathcal{C}_{k,i}(m)$, is $\mathsf{thsig}(\mathsf{vote}, k, i, m)$ of an $(n - f, n)$-threshold signature scheme, i.e., aggregated votes from a quorum of $n - f$ nodes. For a technical reason, we also consider $\perp$ as a certificate for any slot k and value m. We define *freshness* of certificates of the *same slot* by epoch: the higher the epoch, the fresher the certificate (e.g., $\mathcal{C}_{k,1}(m)$ is fresher than $\mathcal{C}_{k,0}$). Also, any certificate is fresher than $\perp$.

We say a leader L_i *equivocates*, if there are two different proposals in the same epoch and slot, i.e., $\langle \mathsf{prop}, k, i, m, \mathcal{C} \rangle_{L_i}$ and $\langle \mathsf{prop}, k, i, m', \mathcal{C} \rangle_{L_i}$ for $m \neq m'$.

Common path. We now explain the protocol in more detail. The first 7 rounds of each epoch perform the propose-then-vote operation. The leader proposes a value m (round Propose), and other nodes vote for it twice. They first vote for the value m (round Vote), then vote for the certificate $\mathcal{C}_{k,i}(m)$ (round Propagate-2). Nodes commit m after receiving a *commit-proof* $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$. To make the cost linear, we use two known techniques.

First, we use the leader as a "message hub" [32]. Namely, nodes send signed votes (threshold signature shares) only to the leader, and the leader, after collecting a quorum of $n - f$ votes, aggregates them into a κ-size certificate (or a commit-proof) and sends it to nodes. This allows nodes to make progress with linear costs under an honest leader.

Second, we use an expander graph to prevent the formation of certificates on two different values (which would lead to disagreement) [25]. More specifically, we use an $(n, 2\varepsilon, 1 - 2\varepsilon)$-expander with each vertex representing each node. Each node, after receiving a proposal from the leader, sends it to its neighbors in the expander (round Propagate-1) before voting. If a certificate $\mathcal{C}_{i,k}(m)$ exists, $n - f \geq f + 2\varepsilon n$ nodes, out of which at least $2\varepsilon n$ nodes are honest, must have sent the proposal of m to their neighbors. The expansion property implies more than $(1-2\varepsilon)n \geq 2f$ nodes, out of which at least $f+1$ are honest, would receive the proposal. They would never vote for $m' \neq m$, so $\mathcal{C}_{i,k}(m')$ cannot exist.

Achieving liveness with dishonest leaders. So far, we have explained how to commit safely with linear communication when the leader is honest. But if the leader is dishonest, a commit-proof may not be formed. In that case, to ensure liveness, nodes accuse the leader by sending accuse messages to all nodes (round Query-1). If the leader is completely silent, at least $n-f$ nodes accuse the leader which forms a *corrupt-proof* of the leader. The corrupt proof will be forwarded to everybody (after aggregation), and honest nodes will simply ignore this leader ever after. But the situation will be more complex if the dishonest leader sends messages selectively. Some nodes may receive the commit-proof, but some may not. Since not everybody accuses the leader, we do not have a corrupt-proof; but not everybody receives a commit-proof, either. The last four rounds (rounds 8–11) resolve this issue. Roughly, we try to disseminate the commit-proof in two steps. First, the node u missing the commit-proof queries one node v selected deterministically (round Query-1), who responds with the commit-proof (round Repond-1). If v does not help, then node u accuses v and queries all nodes, some of whom have a commit-proof (round Query-2, Respond-2). The high-level idea is, though quadratic communication may be incurred in this latter query-all step, the number of such occurrences is bounded. An honest node u selects its helper from nodes that it has not accused (Query-1). So each honest node will eventually find an honest helper, after which they can receive a commit-proof from the helper alone. A dishonest node u may try to keep invoking the query-all step. But honest nodes respond only when u accuses a new node (Respond-2). Thus, the dishonest node u will eventually run out of new nodes to accuse, after which honest nodes will no longer respond to u.

Algorithm 4.2: Linear communication multi-shot BB

Each epoch $0 \leq i \leq f+1$ of slot $k \geq 1$ takes 11 rounds. Let L_i be the leader of epoch i, and G_ε be an $(n, 2\varepsilon, 1-2\varepsilon)$-expander that is known to all nodes. Each node u runs the following steps if it has neither 1) committed in slot k, nor 2) received the corrupt-proof $\mathsf{thsig}(\mathsf{accuse}, L_i)$.

// Leader proposes a value

1. **Collect**: Send the freshest slot-k certificate to L_i.
2. **Propose**: If $u = L_i$, multicast $\langle \mathsf{prop}, k, i, m, \mathcal{C}\rangle_{L_i}$ where:
   (a) If u has received a slot-k certificate ($\neq \perp$), then $\mathcal{C}$ is the freshest $\mathcal{C}_{k,j}(m)$ among those ever received.
   (b) Otherwise, $\mathcal{C} = \perp$, and $m \leftarrow \mathsf{bc}_k$ (if $i=0$) or an arbitrary value (if $i>0$).

// Vote for a leader's value

3. **Propagate-1**: If u receives a proposal $\langle \mathsf{prop}, k, i, m, \mathcal{C}_{k,j}(m)\rangle_{L_i}$ s.t. $\mathcal{C}_{k,j}(m)$ is a certificate as fresh as what u sent to L_i during **Collect**, then send the proposal to its neighbors in the expander G_ε.

4. **Vote**: If u has detected equivocation of L_i, u multicasts $\langle \mathsf{accuse}, L_i \rangle_u$ (if not yet sent). Else, if u has sent L_i's proposal on m in **Propagate-1**, send $\langle \mathsf{vote}, k, i, m \rangle_u$ to L_i.

// Vote for a certificate

5. **Certificate**: If $u = L_i$ and it receives $n - f$ $\langle \mathsf{vote}, k, i, m \rangle_*$ messages, aggregate them to generate $\mathcal{C}_{k,i}(m) \leftarrow \mathsf{thsig}(\mathsf{vote}, k, i, m)$ and multicast it.

6. **Propagate-2**: If u receives $\mathcal{C}_{k,i}(m)$, send it to its neighbors in G_ε, and send $\langle \mathcal{C}_{k,i}(m) \rangle_u$ to the leader L_i.

7. **Commit**: If $u = L_i$ and it receives $n-f$ $\langle \mathcal{C}_{k,i}(m) \rangle_*$ messages, aggregate them to generate a *commit-proof* $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$ and multicast it.

// Disseminate a commit-proof

8. **Query-1**: If u has not received any commit-proof of epoch i, multicast $\langle \mathsf{accuse}, L_i \rangle_u$ (if not yet sent), and send $\langle \mathsf{query}_1, k, i \rangle_u$ to the smallest node v s.t. 1) u has not accused v and 2) v has not accused L_i.

9. **Respond-1**: If u has received $\langle \mathsf{query}_1, k, i \rangle_v$ and it has a commit-proof $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$, send $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$ to v if 1) v has accused L_i and 2) u is the smallest node v has not accused.

10. **Query-2**: If u sent a query_1 message to node v in **Query-1** and has not received a commit-proof from v, then multicast $\langle \mathsf{accuse}, v \rangle_u$ and $\langle \mathsf{query}_2, k, i \rangle_u$.

11. **Respond-2**: If u has received $\langle \mathsf{accuse}, w \rangle_v$ and $\langle \mathsf{query}_2, k, i \rangle_v$ and it has a commit-proof $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$, send $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$ to v, if this is the first time u receives $\langle \mathsf{accuse}, w \rangle_v$.

At any point in the protocol:

($\star$) Upon receiving a commit-proof $\mathsf{thsig}(\mathcal{C}_{k,j}(m))$, commit m at slot k.

($\star$) Upon receiving $\langle \mathsf{accuse}, v \rangle_w$, forward it to the accused node v.

($\star$) Upon receiving $n-f$ $\langle \mathsf{accuse}, v \rangle_*$ messages for any v, aggregate them into a corrupt-proof $\mathsf{thsig}(\mathsf{accuse}, v)$ and multicast it.

($\star$) Upon receiving a commit-proof $\mathsf{thsig}(\mathcal{C}_{k,j}(m))$ for any j, if u has received $n - f$ $\langle \mathsf{accuse}, L_j \rangle_*$ messages, then multicast the commit-proof.

4.2.1 Proof of correctness

We first show that the protocol satisfies *consistency* using the following two lemmas. Lemma 4.2.1 implies that nodes cannot commit different messages within the same epoch. Lemma 4.2.2 implies that nodes cannot commit differently even across different epochs.

Lemma 4.2.1. *If certificates $\mathcal{C}_{k,i}(m)$ and $\mathcal{C}_{k,i}(m')$ both exist, then $m = m'$.*

Proof. Suppose $\mathcal{C}_{k,i}(m)$ exists, then at least $n - f \geq f + 2\varepsilon n$ nodes must have sent $\langle \mathsf{vote}, k, i, m\rangle_*$ vote message, out of which at least $2\varepsilon n$ honest nodes must have forwarded the leader's proposal to the neighbors in the expander graph (in round Propagate-1). Due to the expansion property, more than $(1 - 2\varepsilon)n \geq 2f$ nodes, out of which at least $f + 1$ honest nodes would receive the proposal, who would never vote for $m' \neq m$. Thus, $\mathcal{C}_{k,i}(m')$ cannot exist. □

Lemma 4.2.2. *If there exists a commit-proof* $\mathsf{thsig}(\mathcal{C}_{k,i}(m))$*, then for all epochs* $j > i$*, there cannot exist a certificate* $\mathcal{C}_{k,j}(m')$ *on* $m' \neq m$*.*

Proof. The commit-proof requires a quorum of $n - f$ nodes' signatures on $\mathcal{C}_{k,i}(m)$. Therefore, at least $n - 2f \geq 2\varepsilon n$ honest nodes must have received $\mathcal{C}_{k,i}(m)$ in epoch i of slot k. After they propagate it in round Propagate-2, more than $(1 - 2\varepsilon)n \geq 2f$ nodes (out of which at least $f + 1$ are honest) must have received $\mathcal{C}_{k,i}(m)$. The $f + 1$ honest nodes will send $\mathcal{C}_{k,i}(m)$ to the next leader L_{i+1} during Leader setup, so they will never vote for a proposal in the epoch $i + 1$ unless it contains an epoch i certificate. As a certificate $\mathcal{C}_{k,i}(m')$ cannot exist for $m' \neq m$ (by Lemma 4.2.1), they will never vote for m' in epoch $i + 1$ of slot k, which implies $\mathcal{C}_{k,i+1}(m')$ cannot exist. Inductively, for any $j > i$, a certificate $\mathcal{C}_{k,j}(m')$ cannot exist. □

With the above lemmas, we can now show that our protocol satisfies *consistency*.

Theorem 4.2.3. *The protocol satisfies consistency. If any two honest nodes* u *and* v *commit* m_u *and* m_v *respectively, at the slot* k*, it must be that* $m_u = m_v$*.*

Proof. An honest node outputs if it has observed a commit-proof. Suppose u observes a commit-proof on m_u of epoch i and v observes a commit-proof on m_v of epoch j. We can assume w.l.o.g. that $i \leq j$.

- If $i = j$, by Lemma 4.2.1, there cannot exist a commit-proof on different messages in epoch i. Therefore, $m_u = m_v$.
- If $i < j$, by Lemma 4.2.2, if a commit-proof on m_u exists in epoch i, then any future commit-proof must also be on m_u. Therefore, $m_u = m_v$.

This completes our consistency proof. □

Finally, to prove the remaining properties, we prove the following lemma that shows an honest leader's epoch is always successful.

Lemma 4.2.4. *If the leader* L_i *is honest, all honest nodes commit by the end of epoch* i *in each slot.*

Proof. Consider slot $k = 1$. In epochs before i, since L_i is not the leader, the only case where L_i gets accused by an honest node u is when u has sent query_1 to L_i but it has not sent back a commit-proof to u during round Respond-1. However, if L_i does not have a commit-proof to send back to u, then L_i must have also accused the epoch's leader, and u would not have sent query_1 to L_i. So, honest nodes do not accuse L_i before epoch i. Now, in epoch i, since $n - f$ accusations cannot exist for L_i, all honest nodes (of at least $n - f$) vote for the leader's proposal (the leader's proposal is always accepted by honest nodes since it contains the freshest certificate among those honest nodes sent during the round Collect, forming a certificate and then a commit-proof. So honest nodes commit and do not accuse L_i. Therefore, in the next slot (and inductively in all later slots), all honest nodes commit by the end of epoch i without being accused by honest nodes. □

The lemma above trivially implies *validity* as the first leader L_0 is the sender. Also, since we have at most f dishonest leaders, we will have an honest leader by epoch $f+1$. So, by the end of each slot, all honest nodes commit a value for the slot. This implies our protocol satisfies *termination* and *sequentiality*.

4.2.2 Communication and round complexity

Let us first consider the cost of *expensive slots*: a slot with more than one epoch with non-zero communication. In an expensive slot, the leaders of all epochs except the last one must be accused by all honest nodes, forming the corrupt-proofs (i.e., $n-f$ accusations) for these leaders; Otherwise, at least an honest node who does not accuse the leader must have received a commit-proof before round Query-1 which is forwarded to all honest nodes (in either round Respond-1 or 2) and all honest nodes would stop sending messages in all later epochs. So, the number of epochs across all expensive slots is $O(n)$, and the total cost across all expensive slots is $O(\kappa n^3)$ as each epoch costs at most $O(\kappa n^2)$ communication.

Next, for non-expensive slots (i.e., with only one epoch), we consider *expensive epochs*: an epoch with super-linear communication. Obviously, the first 7 rounds cost $O(\kappa n)$ per epoch. The cost of forwarding a commit-proof will be super-linear communication only when a corrupt-proof is formed; such epochs exist at most f times, hence the total cost is $O(\kappa n^3)$ across all expensive epochs. Finally, the cost of rounds 8–11 is analyzed below:

1. Query-1: Each honest node sends only one message; per-epoch cost is linear.
2. Respond-1: Each honest node can receive a response from one node, which costs linear (in the total number of nodes n) per epoch. Suppose in an epoch, multiple honest nodes R respond to a malicious node v. In this case, v has accused all nodes in R except the highest one, which are forwarded to the accused nodes, who will stop responding in all later epochs/slots. Therefore, for each malicious node v, there are at most n epochs s.t. multiple honest nodes respond to v; the total cost is $O(\kappa n^3)$ across all expensive epochs.
3. Query-2: Each honest node sends query_2 when it accuses a new node (i.e., the helper), which can happen at most f times; the total cost is $O(\kappa n^3)$ across all expensive epochs.
4. Respond-2: An honest node u responds to a node v only if v accuses a new node, which can happen at most n times; the total cost is $O(\kappa n^3)$ across all expensive epochs.

To sum up, the total cost is $O(\kappa nL + \kappa n^3)$ for L slots. As for the round complexity, all honest nodes would commit once an honest leader is elected. Following the analysis in Section 3, while the protocol requires linear rounds in the worst case, it still enjoys an expected constant round complexity.

Chapter 5

Sublinear Round BB under Strongly Adaptive Adversary and Dishonest Majority

In Section 3, we achieved expected round complexity under dishonest majority and a weakly adaptive adversary. However, the trust graph and Trustcast solution fails when the adversary is strongly adaptive. A strongly adaptive adversary can discover who the leader is in each epoch and adaptively corrupt the leader, causing the protocol to run for $f + 1$ epochs. Therefore, an efficient BB protocol under a strongly adaptive adversary requires completely new techniques. We first give a high-level overview of our main techniques.

Delayed-exposure message distribution. One major new technique we introduce is a delayed-exposure message distribution mechanism. Specifically, we devise a novel polylogarithmic round, randomized protocol that allows all n honest nodes to each distribute a time-lock puzzle that embeds a message they want to send; moreover, by the end of polylogarithmically many rounds, all honest nodes can obtain the solutions of all other honest nodes' puzzles. On the other hand, even if the adversary has unbounded parallelism, it cannot learn any information about honest nodes' messages encoded in the puzzles within one round of time; and thus the adversary cannot make informed adaptive corruptions based on the message contents.

To solve this problem, we need to overcome several technical challenges. First, although we allow the adversary to have access to unbounded parallelism, it is unrealistic to expect that honest machines are also equipped with up to n amount of parallelism. Like in the standard distributed protocol literature, we assume that the honest nodes are *sequential* RAM machines and thus they cannot solve puzzles in parallel. However, if they solved all the puzzles sequentially it would take them a linear number of rounds which is what we want to avoid in the first place.

Second, the honest nodes do not even have a consistent view of the puzzles being distributed which makes it difficult to coordinate who solves which puzzles. To overcome these challenges, we devise a novel *age-based sampling* technique where nodes sample puzzles to solve and the probability of sampling grows exponentially w.r.t. how long ago the puzzle was first seen. The detailed construction and its analysis are in Sections 5.1.3 and 5.1.4.

We stress that the delayed-exposure primitive can be of independent interest in other protocol design contexts — in this sense, besides our new construction, we also make a conceptual contribution in defining this new primitive and formulating its security properties (see Section 5.1).

Applying the delayed-exposure distribution mechanism.

Once we have the delayed-exposure distribution mechanism, we can combine it with techniques proposed in the recent work by Chan et al. [16], and upgrade their weakly adaptive protocol to a strongly adaptive one while preserving a polylogarithmic round complexity. For this upgrade, the most challenging aspect is how to prove security. The most natural approach towards proving security is to first prove that the real-world protocol securely emulates a natural ideal-world counterpart, and then argue the security of the ideal-world protocol (which does not involve cryptography) using an information-theoretic argument. Unfortunately, this approach fails partly because the time-lock puzzles only provide transient secrecy of the messages they encode. Instead, we work around this issue by devising a sequence of hybrids from the real-world execution to an ideal-world execution without cryptography, and we show that for every adjacent pair of hybrids, the probability of certain relevant bad events can only increase. Eventually, we upper bound the probability of bad events in the ideal world using an information-theoretic argument.

Soundness of cryptography w.r.t. an adaptive adversary. Last but not the least, in our construction and when arguing about the sequence of hybrids, one technicality that arises is that the adversary is strongly adaptive, and therefore some of the cryptographic building blocks we use must be commensurate and secure against selective opening attacks. This technicality will show up throughout the section in our definitions, constructions, and proofs.

5.1 Delayed-Exposure Message Distribution

5.1.1 Definitions

Syntax

We first introduce the syntax of the $\mathsf{Distribute}(1^\lambda, m_1, \ldots, m_n)$ protocol. At the beginning of the protocol, every node $u \in [n]$ is given a message $m_u \in \{0,1\}^\ell$ of length $\ell(\lambda, n)$ which is upper bounded by a fixed polynomial in λ and n. In the $\mathsf{Distribute}$ protocol, every node makes an attempt to multicast its message m_u to everyone else. At the end of R_{distr} number of rounds, everyone outputs $(m'_1, \ldots, m'_n)$ where $m'_u \in \{0,1\}^\ell \cup \{\perp\}$ denotes the message received from node $u \in [n]$, and $\perp$ indicates that nothing has been received from u.

In the following, we will allow setup assumptions for constructing our $\mathsf{Distribute}$ protocol, specifically, we assume that the setup algorithm $\mathsf{Gen}(1^\lambda)$ outputs a common reference string denoted crs. Moreover, there is a public-key infrastructure (PKI) later used for digital signatures. We assume that during the setup phase, we run the key generation algorithm of a digital signature scheme which outputs a public-key and secret-key pair for every node $u \in [n]$, henceforth denoted vk_u and ssk_u, respectively. We assume that the crs and the PKI consisting of $\{\mathsf{vk}_u, \mathsf{ssk}_u\}_{u\in[n]}$ can be reused across multiple instances of the $\mathsf{Distribute}$ protocol.

Security

At the beginning of the $\mathsf{Distribute}$ protocol, either everyone is honest, or a subset of nodes have already been corrupted by the adversary $\mathcal{A}$. During the $\mathsf{Distribute}$ protocol, the adversary $\mathcal{A}$ can adaptively corrupt more nodes, and upon newly corrupting a node $u \in [n]$, the adversary $\mathcal{A}$ receives u's internal state.

Liveness. Liveness requires the following: let $\widetilde{H}$ denote the set of nodes that remain honest till the beginning of the second round of the Distribute protocol; except with negligible in λ probability, it holds that for every node $u \in \widetilde{H}$, all forever-honest nodes[1] output m_u as the message received from u.

Momentary secrecy. Roughly speaking, we want that even when the adversary $\mathcal{A}$ may have unbounded polynomial parallelism, the honest nodes' messages remain secret to $\mathcal{A}$ till the beginning of the second round of Distribute. Formally, we define the following $\mathsf{Expt}(1^\lambda, \{m_u^*\}_{u\in[n]})$ experiment.

Experiment $\mathsf{Expt}(1^\lambda, \{m_u^*\}_{u\in[n]})$**:** The experiment $\mathsf{Expt}(1^\lambda, \{m_u^*\}_{u\in[n]})$ is defined as follows:

- **Setup.** Run the honest setup algorithm which outputs a common reference string denoted crs and a key pair $(\mathsf{vk}_u, \mathsf{ssk}_u)$ for every $u \in [n]$. The crs and the public verification keys $\{\mathsf{vk}_u\}_{u\in[n]}$ are given to $\mathcal{A}$;
- **Query.** The query phase runs for an arbitrary polynomial amount of time. During this time, $\mathcal{A}$ may adaptively make the following queries where multiple sessions of the Distribute protocol are allowed to be initiated concurrently:
  - *Session.* $\mathcal{A}$ specifies a session identifier sid, as well as a set of input messages $\{m_u\}_{u\in H}$ where H denotes the so-far honest nodes. Now, the so-far honest nodes execute the honest Distribute protocol using the inputs $\{m_u\}_{u\in H}$ and session identifier sid, and interact with $\mathcal{A}$.
  - *Corrupt.* At any time, $\mathcal{A}$ specifies a new node $u \in [n]$ to corrupt, and at this moment ssk_u and all random coins used by node u so far in the protocol are given to $\mathcal{A}$;
- **Challenge.** Finally, $\mathcal{A}$ outputs `challenge` with a challenge session identifier sid^*: it is required that sid^* be a fresh one that has never been queried before. Let H^* denote the honest nodes at the moment. Now, compute the first-round messages denoted M^* that H^* would send in the real-world Distribute protocol with the session identifier sid^*, and using the inputs $\{m_u^*\}_{u\in H^*}$. Output $\mathcal{A}$'s view[2] in the experiment as well as M^*.

We say that the Distribute protocol satisfies momentary secrecy iff for any non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ and any non-uniform $p.p.t.$ $2\mathcal{T}_\emptyset$-bounded parallel distinguisher $\mathcal{D}$, there is a negligible function $\mathsf{negl}(\cdot)$ for the following to be true for any choice of λ, and any $\{m_u^*\}_{u\in[n]}$ and $\{\widetilde{m}_u^*\}_{u\in[n]}$,

$$\left|\Pr\left[\mathcal{D}(1^\lambda, \mathsf{Expt}(1^\lambda, \{m_u^*\}_{u\in[n]})) = 1\right] - \Pr\left[\mathcal{D}(1^\lambda, \mathsf{Expt}(1^\lambda, \{\widetilde{m}_u^*\}_{u\in[n]}))) = 1\right]\right| \\ \leq \mathsf{negl}(\lambda).$$

5.1.2 NIZK preliminaries

We will make use of a non-interactive zero-knowledge proof (NIZK) system that is secure against strongly adaptive corruptions. We present the formal definition of such a NIZK system in this

[1] Forever honest w.r.t. the Distribute protocol means that the node remains honest till the end of the protocol.

[2] Here, $\mathcal{A}$'s view may contain any output $\mathcal{A}$ has produced so far which might have taken arbitrary polynomial time to compute prior to the start of the challenge phase.

section. We use $f(\lambda) \approx g(\lambda)$ to mean that there exists a negligible function $\nu(\lambda)$ such that $|f(\lambda) - g(\lambda)| < \nu(\lambda)$.

A non-interactive proof system henceforth denoted NIZK for an NP language $\mathcal{L}$ consists of the following algorithms.

- $\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{L})$: Takes in a security parameter λ, a description of the language $\mathcal{L}$, and generates a common reference string crs.
- $\pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathsf{stmt}, w)$: Takes in crs, a statement stmt, a witness w such that $(\mathsf{stmt}, w) \in \mathcal{L}$, and produces a proof π.
- $b \leftarrow \mathsf{V}(\mathsf{crs}, \mathsf{stmt}, \pi)$: Takes in a crs, a statement stmt, and a proof π, and outputs 0 (reject) or 1 (accept).

Perfect completeness. A non-interactive proof system is said to be perfectly complete, if an honest prover with a valid witness can always convince an honest verifier. More formally, for any $(\mathsf{stmt}, w) \in \mathcal{L}$, we have that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda, \mathcal{L}),\ \pi \leftarrow \mathsf{P}(\mathsf{crs}, \mathsf{stmt}, w) : \mathsf{V}(\mathsf{crs}, \mathsf{stmt}, \pi) = 1\right] = 1.$$

Non-erasure computational zero-knowledge. Non-erasure zero-knowledge requires that under a simulated CRS, there is a simulated prover that can produce proofs without needing the witness. Further, upon obtaining a valid witness to a statement a-posteriori, the simulated prover can explain the simulated NIZK with the correct witness.

We say that a proof system $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ satisfies non-erasure computational zero-knowledge iff there exists a probabilistic polynomial-time algorithm $(\mathsf{Gen}_0, \mathsf{P}_0, \mathsf{Explain})$ such that

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda), \mathcal{A}^{\mathsf{Real}(\mathsf{crs},\cdot,\cdot)}(\mathsf{crs}) = 1\right] \approx \Pr\left[(\mathsf{crs}_0, \tau_0) \leftarrow \mathsf{Gen}_0(1^\lambda), \mathcal{A}^{\mathsf{Ideal}(\mathsf{crs}_0,\tau_0,\cdot,\cdot)}(\mathsf{crs}_0) = 1\right],$$

where $\mathsf{Real}(\mathsf{crs}, \mathsf{stmt}, w)$ runs the honest prover $\mathsf{P}(\mathsf{crs}, \mathsf{stmt}, w)$ with randomness r and obtains the proof π, and then outputs (π, r); $\mathsf{Ideal}(\mathsf{crs}_0, \tau_0, \mathsf{stmt}, w)$ runs the simulated prover $\pi \leftarrow \mathsf{P}_0(\mathsf{crs}_0, \tau_0, \mathsf{stmt}, \rho)$ with randomness ρ and without a witness, and then runs $r \leftarrow \mathsf{Explain}(\mathsf{crs}_0, \tau_0, \mathsf{stmt}, w, \rho)$ and outputs (π, r).

Perfect knowledge extraction. We say that a proof system $(\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ satisfies perfect knowledge extraction, if there exists probabilistic polynomial-time algorithms $(\mathsf{Gen}_1, \mathsf{Extr})$, such that for all (even unbounded) adversary $\mathcal{A}$,

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{Gen}(1^\lambda) : \mathcal{A}(\mathsf{crs}) = 1\right] = \Pr\left[(\mathsf{crs}_1, \tau_1) \leftarrow \mathsf{Gen}_1(1^\lambda) : \mathcal{A}(\mathsf{crs}_1) = 1\right],$$

and moreover,

$$\Pr\left[\begin{array}{c} (\mathsf{crs}_1, \tau_1) \leftarrow \mathsf{Gen}_1(1^\lambda) \\ (\mathsf{stmt}, \pi) \leftarrow \mathcal{A}(\mathsf{crs}_1) \\ w \leftarrow \mathsf{Extr}(\mathsf{crs}_1, \tau_1, \mathsf{stmt}, \pi) \end{array} : \begin{array}{c} \mathsf{V}(\mathsf{crs}_1, \mathsf{stmt}, \pi) = 1 \\ \text{but } (\mathsf{stmt}, w) \notin \mathcal{L} \end{array}\right] = 0.$$

Theorem 5.1.1 (Instantiation of NIZK [20]). *Assume that the decisional linear assumption holds in suitable bilinear groups. Then, there exists a proof system that satisfies perfect completeness, non-erasure computational zero-knowledge, and perfect knowledge extraction.*

5.1.3 Construction

Assumptions. In the construction below, we assume that there is a public-key infrastructure (PKI) available, and *nodes sign all messages* that they want to send. Only messages attached with valid signatures from the purported senders are considered valid, and all invalid messages are discarded. We also make an *implicit echoing* assumption: we assume that every honest node will echo every fresh message received to everyone, such that if a forever-honest node $u \in [n]$ observes some message at the beginning of round r, then every so-far honest node will have received it by the beginning of round $r + 1$.

NP language. We make use of a non-interactive zero-knowledge proof (NIZK) defined in Section 5.1.2. Let us briefly review the NP language used in our NIZK proofs. A statement is of the form $\mathsf{stmt} := (u, Z)$, and a witness is of the form $w := (m, \sigma, \rho)$. We assume that $(\lambda, \mathcal{T}_{\text{solve}}, \mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ are global parameters and do not repeat them in the statement. A statement $\mathsf{stmt} := (u, Z)$ is in the language vouched for by a valid witness $w := (m, \sigma, \rho)$ iff there exists (m, σ, ρ) such that $Z = \mathsf{TLP.Gen}(1^\lambda, \mathcal{T}_{\text{solve}}, (m, \sigma); \rho)$ and moreover $\Sigma.\mathsf{Vf}(\mathsf{vk}_u, m, \sigma) = 1$.

Protocol. Let $\mathsf{TLP} := (\mathsf{Gen}, \mathsf{Solve})$ denote a time-lock puzzle with hardness parameter ξ as defined in Section 2.1.4, and let $\mathcal{T}_\emptyset$ denote the duration of one synchronous round. Let $\mathsf{NIZK} := (\mathsf{Gen}, \mathsf{P}, \mathsf{V})$ denote a non-interactive zero-knowledge proof system as defined in Section 5.1.2. Let $\Sigma := (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Vf})$ denote a digital signature scheme. The $\mathsf{Distribute}$ protocol is described below.

Input: Each node $u \in [n]$ receives the input $m_u \in \{0, 1\}^\ell$. Without loss of generality, henceforth, we shall assume that the message m_u itself is tagged with the sender's identifier $u \in [n]$. Below, we may assume that we always prefix the message m_u with the string `inp`.

Setup: Run $\mathsf{crs}_{\text{nizk}} \leftarrow \mathsf{NIZK.Gen}(1^\lambda)$ and publish $\mathsf{crs}_{\text{nizk}}$. Recall that there is a PKI and nodes sign all messages they send, henceforth, we use vk_u and ssk_u to denote the public- and secret-key of node u, respectively.

Protocol:

1. **Initial round**: every node $u \in [n]$ does the following:

   - let $\sigma := \Sigma.\mathsf{Sign}(\mathsf{ssk}_u, m_u)$; call $Z_u \leftarrow \mathsf{TLP.Gen}(1^\lambda, \mathcal{T}_{\text{solve}}, (m_u, \sigma); \rho)$ where $\mathcal{T}_{\text{solve}} := 2\mathcal{T}_\emptyset/\xi$ and ρ explicitly denotes the randomness consumed by the $\mathsf{TLP.Gen}$ algorithm;
   - call $\pi_u \leftarrow \mathsf{NIZK.P}(\mathsf{crs}_{\text{nizk}}, (u, Z_u), (m_u, \sigma, \rho))$;
   - sign and multicast the tuple $(\mathsf{puz}, Z_u, \pi_u)$ to everyone.

   Henceforth, we assume that whenever an honest node receives a message of the form $(\mathsf{puz}, Z_u, \pi_u)$ signed by u, it calls $\mathsf{NIZK.V}(\mathsf{crs}_{\text{nizk}}, (u, Z_u), \pi_u)$ and if the verification fails, the message is discarded immediately without being processed. If the verification succeeds, the puzzle Z_u is considered as received and we say that it belongs to u.

2. **Solve phase**: Henceforth, every $\mathcal{T}_{\text{epoch}} := \mathcal{T}_{\text{solve}} \cdot \lceil \frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3) \rceil + 1$ rounds is called an *epoch*: round 1 is the beginning of the first epoch; round $\mathcal{T}_{\text{epoch}} + 1$ is the beginning of the second epoch, and so on.

Repeat the following for a total of $E = \lceil \log_2 n \rceil + 1$ epochs.

- At the beginning of each epoch, let S denote the set of all puzzles received so far and belonging to *active* nodes. For $Z \in S$, define the puzzle's *age* $\alpha(Z)$ as follows: $\alpha(Z) := \lceil \frac{r-r'}{\mathcal{T}_{\text{epoch}}} \rceil$ where r denotes the beginning round of the epoch and $r' \leq r$ denotes the first round in which Z was observed.
- For each $Z \in S$ sequentially, perform the following: flip a random coin that comes up heads with probability $p := \min\left(\frac{2^{\alpha(Z)} \cdot \ln(16n/h)}{h}, 1\right)$; if the coin comes up heads, then solve the puzzle Z by calling $(m, \sigma) \leftarrow \mathsf{TLP.Solve}(Z)$. Once solved, multicast the solution (m, σ) to everyone[a].

Output: at any time, upon observing a tuple (m, σ) where σ is a valid signature on m from the purported sender (henceforth, denoted $v \in [n]$), if no message from v has been output yet, output m as the message received from v and mark v as *inactive*. At the end of the protocol, if no message from some $v \in [n]$ has been output, then we output a canonical message $\bot$ as the message from v.

Detect equivocation: at any time, if multiple puzzles have been received from the same node $v \in [n]$, mark v as *inactive*[b].

[a]We may assume that if more than $\frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3)$ number of puzzles are chosen to be solved in some epoch, the node simply aborts outputting $\mathsf{failure}$ — we will show in the proof of Lemma 5.1.4 later that this does not happen except with negligible probability.

[b]Both the **Output** and **Equivocation** entry points are processed at the beginning of every round before all other actions of the round.

Clearly, the total round complexity of the $\mathsf{Distribute}$ protocol is

$$(\lceil \log_2 n \rceil + 1) \cdot \frac{\mathcal{T}_{\text{solve}}}{\mathcal{T}_{\emptyset}} \cdot \left(\frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3) \rceil + 1 \right) + 1.$$

Assume that n is polynomially bounded in λ, then the round complexity is upper bounded by $O(\frac{n}{\xi \cdot h}) \cdot \mathsf{poly}\log \lambda$.

5.1.4 Proofs: Liveness

Henceforth, we use the term "m is *in honest view*" to mean that some forever-honest node has seen m.

Fact 5.1.2. *If some forever-honest node observes a puzzle Z at the beginning of epoch e, then all so-far honest nodes will have observed Z by the beginning of epoch $e + 1$.*

Proof. Follows directly from the implicit echoing assumption, i.e., honest nodes echo every fresh message they see. □

Fact 5.1.3. *Assume that the* NIZK *scheme satisfies perfect knowledge extraction, and the* TLP *scheme satisfies correctness. Then, except with negligible probability, the following must hold: if any forever-honest node solves a puzzle Z belonging to $v \in [n]$ by the beginning of the last round of epoch e, then no puzzle from v will still be active in any so-far honest node's view at the beginning of epoch $e + 1$.*

Proof. If the NIZK satisfies perfect knowledge extraction, then it must be that the solved solution is a (m, σ) pair such that σ is a valid signature from v on m. Since an honest node has solved the puzzle and found the solution by the beginning of the last round of epoch e, it will multicast (m, σ) to everyone in the last round of epoch e, and by the beginning of epoch $e + 1$, every so-far honest node will have observed (m, σ) and will have marked v as inactive. $\square$

Given Fact 5.1.2, we know that honest nodes' perception of a puzzle's age can differ by at most 1. We say that a puzzle Z's *minimum age* is $\alpha \geq 0$ in epoch e, iff all forever-honest nodes have observed it by the beginning of epoch $e - \alpha$, but at least one forever-honest node has not observed it by the beginning of epoch $e - \alpha - 1$.

Henceforth, if at the beginning of some epoch e, a so-far honest has seen a puzzle belonging to an active node, then the puzzle is said to be *active*. Due to the equivocation detection rule, it must be that from each active node, at most one active puzzle has been seen.

Lemma 5.1.4. *Let $\alpha \geq 0$ and $n \geq \log^2 \lambda$. Except with negligible in λ probability, the following holds: at the beginning of every epoch, there can be at most $n/2^{\alpha-1}$ active puzzles belonging to distinct nodes, and with minimum age α, in honest view.*

Proof. We first state some simplifying assumptions that can be made without loss of generality. We may assume that honest nodes use a puzzle's minimum age to determine the probability p with which a puzzle is selected to be solved. Note that in the real-world protocol, a node does not necessarily know the minimum age of the puzzle, but we may assume it for proving this lemma since making p smaller will only increase the probability of the bad event stated in the lemma that we care about bounding. Furthermore, let us first assume that any forever-honest node has enough time to solve all puzzles it chooses to solve during any epoch e, and not only so, they can be solved by the beginning of the last round of the epoch e — later we will show that indeed this is the case except with negligible probability.

For $\alpha = 0$, the lemma trivially holds. Henceforth, we may assume that $\alpha \geq 1$. Fix any epoch e, and let $S_{\alpha-1}$ denote all active puzzles whose minimum age is $\alpha - 1 \geq 0$ at the beginning of epoch e. This means that all so-far honest nodes will choose to solve any puzzle in $S_{\alpha-1}$ with probability $p = \min(\frac{2^{\alpha-1} \cdot \ln(16n/h)}{h}, 1)$ in epoch e. We would like to upper bound the probability that at least $n/2^{\alpha}$ puzzles in $S_{\alpha-1}$ are not selected by any forever-honest node in epoch e. Due to Fact 5.1.3, if $\frac{2^{\alpha-1} \cdot \ln(16n/h)}{h} \geq 1$, then there cannot be any active puzzles of minimum age α left in any honest node's view at the beginning of the next epoch. Henceforth, we may also assume that $p = \frac{2^{\alpha-1} \cdot \ln(16n/h)}{h} < 1$.

Consider a fixed honest node u and an active puzzle from a fixed node v: the probability that u does not select an active puzzle from v is at most $1 - p$. The probability that any fixed set of h forever-honest nodes (denoted W) all do not select an active puzzle from v is $(1 - p)^h$. For any fixed set of $n/2^{\alpha-1}$ nodes denoted Γ that has a puzzle of minimum age $\alpha - 1$ in honest view in epoch e, the probability that a fixed set of h forever-honest nodes' puzzle choices do not intersect with Γ is at most $(1 - p)^{h \cdot n/2^{\alpha-1}}$.

The probability that there exists a choice for the set W (consisting of h forever-honest nodes), and a set $\Gamma \subset [n]$ of size $n/2^{\alpha-1}$ (who have a puzzle of age $\alpha - 1$ in honest view in epoch e), such that W's puzzle choices do not intersect with Γ is upper bounded by the following expression:

$$
\begin{aligned}
& (1-p)^{h \cdot n/2^{\alpha-1}} \cdot \binom{n}{h} \cdot \binom{n}{n/2^{\alpha-1}} \\
\leq & \left(1 - \frac{2^{\alpha-1} \cdot \ln(16n/h)}{h}\right)^{hn/2^{\alpha-1}} \cdot \binom{n}{h} \cdot \binom{n}{n/2^{\alpha-1}} \\
= & \left(1 - \frac{2^{\alpha-1} \cdot \ln(16n/h)}{h}\right)^{\frac{h}{2^{\alpha-1} \cdot \ln(16n/h)} \cdot \ln(\frac{16n}{h}) \cdot n} \cdot \binom{n}{h} \cdot \binom{n}{n/2^{\alpha-1}} \\
\leq & \exp\left(-\ln\left(\frac{16n}{h}\right) \cdot n\right) \cdot \binom{n}{h} \cdot \binom{n}{n/2^{\alpha-1}} \\
= & \left(\frac{h}{16n}\right)^{n} \cdot \binom{n}{h} \cdot \binom{n}{n/2^{\alpha-1}} \qquad (*) \\
= & \left(\frac{h}{16n}\right)^{n} \cdot \binom{n}{h} \cdot \binom{n}{n - n/2^{\alpha-1}} \\
\leq & \left(\frac{h}{16n}\right)^{n} \cdot \left(\frac{en}{h}\right)^{h} \cdot \left(\frac{en}{n(1-1/2^{\alpha-1})}\right)^{n(1-1/2^{\alpha-1})} \\
\leq & \left(\frac{h}{16n}\right)^{n} \cdot \left(\frac{en}{h}\right)^{n} \cdot \left(\frac{en}{n(1-1/2^{\alpha-1})}\right)^{n} \\
= & \left(\frac{h}{16n} \cdot \frac{en}{h} \cdot \frac{en}{n(1-1/2^{\alpha-1})}\right)^{n} \leq \exp(-\Theta(n)).
\end{aligned}
$$

In the above derivation, if $\alpha = 1$, then the last term $\binom{n}{n/2^{\alpha-1}}$ in the expression $(*)$ is equal to 1. Therefore, in the derivation steps after the expression $(*)$, we can simply assume that $\alpha > 1$ which only makes the expression $\binom{n}{n/2^{\alpha-1}}$ larger.

So far, we have assumed that if a forever-honest node selects some puzzle to solve in some epoch e, it will actually have enough time to solve the puzzle by the beginning of the last round of epoch e. We now show that the allotted epoch duration $\mathcal{T}_{\text{epoch}} := \mathcal{T}_{\text{solve}} \cdot \lceil \frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3) \rceil + 1$ is indeed long enough to meet this requirement except with negligible probability. Basically, if in epoch e, the number of puzzles of minimum age α left in honest view is at most $n/2^{\alpha-1}$, and an honest node selects each puzzle with probability $p = \min(\frac{2^{\alpha-1} \cdot \ln(16n/h)}{h}, 1)$, we can bound the total number of puzzles of minimum age α an honest node selects to solve with the following two cases:

- Case 1: if $\frac{2^{\alpha-1} \cdot \ln(16n/h)}{h} \geq 1$, then $2^{\alpha-1} \geq h/\ln(16n/h)$. The total number of puzzles of minimum age α selected to solve is upper bounded by $n/2^{\alpha-1} \leq \frac{n}{h} \cdot \ln(16n/h)$.

- Case 2: if $\frac{2^{\alpha-1} \cdot \ln(16n/h)}{h} < 1$, then the expected number of puzzles of minimum age α selected to solve is upper bounded by

$$\frac{n}{2^{\alpha-1}} \cdot \frac{2^{\alpha-1} \cdot \ln(16n/h)}{h} = \frac{n}{h} \cdot \ln(16n/h).$$

By the Chernoff bound, the probability that the number of puzzles of minimum age α selected to solve is more than

$$\frac{n}{h} \cdot \ln(16n/h) + \sqrt{\frac{n}{h} \cdot \ln(16n/h)} \cdot \log^2 \lambda \leq \frac{n}{h} \cdot \ln(16n/h) \cdot \log^2 \lambda \qquad (**)$$

is upper bounded by $\exp(-\Omega(\log^4 \lambda))$.

Recall that the number of ages is upper bounded by the number of epochs, that is $\lceil \log_2 n \rceil + 1$. Now, taking a union bound over all possible ages, except with $\exp(-\Omega(\log^4 \lambda))$ probability, the total number of puzzles an honest node chooses to solve in an epoch is upper bounded by

$$\frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot \lceil \log_2 n \rceil \leq \frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3).$$

Finally, taking a union bound over the number of epochs which is polynomially bounded in λ, we have that except with $\exp(-\Omega(\log^4 \lambda))$ probability, the above bad event will never happen throughout all epochs.

Thus, the allotted epoch duration $\mathcal{T}_{\text{epoch}} := \mathcal{T}_{\text{solve}} \cdot \lceil \frac{2n}{h} \cdot \ln(\frac{16n}{h}) \cdot \log^2 \lambda \cdot (\log n + 3) \rceil + 1$ is sufficiently long such that except with negligible in λ probability, an honest node has time to solve all puzzles it chooses in every epoch. □

Remark 8. *The expression $(**)$ is by no means the tightest possible bound, but it is outside the scope of this thesis to understand what the best possible constant c is in the $O(\log^c(\lambda, n))$ round complexity bound. With our current techniques, we only know how to achieve polylogarithmic round complexity in the strongly adaptive setting under corrupt majority. It is an exciting open question whether we can improve the result to, say, expected constant rounds, or prove impossibility.*

Theorem 5.1.5 (Liveness). *Assume that $n \geq \log^2 \lambda$, that the* NIZK *scheme satisfies soundness, the* TLP *scheme satisfies correctness, and the signature scheme Σ satisfies existential unforgeability under chosen-message attacks. Then, the above* Distribute *protocol satisfies liveness.*

Proof. Let $\widetilde{H}$ denote the set of nodes that remain honest till the beginning of the second round of the Distribute protocol. For every $u \in \widetilde{H}$, every so-far honest node will have received an honestly generated puzzle Z_u from u at the beginning of the second round, i.e., the beginning of the first epoch of the **Solve** phase. At the beginning of the next round after the final epoch E, Z_u, if still active, would have age E in every honest node's view, i.e., its minimum age is E. Due to Lemma 5.1.4, except with negligible probability, the number of active puzzles from nodes in $\widetilde{H}$ is then upper bounded by $n/2^{E-1} < 1$. Now, since honest nodes do not double sign puzzles, except with negligible probability, it must be that every forever-honest node has output a message for everyone in $\widetilde{H}$. □

5.1.5 Proofs: Momentary Secrecy

Experiment $\mathsf{Hyb}(1^\lambda, \{m_u^*\}_{u\in[n]})$**.** The hybrid experiment $\mathsf{Hyb}(1^\lambda, \{m_u^*\}_{u\in[n]})$ is defined almost identically as $\mathsf{Expt}^{\mathcal{A}}(1^\lambda, \{m_u^*\}_{u\in[n]})$, except with the following modifications:

- Run the simulated NIZK setup algorithm $\mathsf{Gen}_0(1^\lambda)$ which outputs a crs and a trapdoor τ;

- Whenever an honest node u is supposed to compute a NIZK proof by calling

$$\pi_u \leftarrow \mathsf{NIZK.P}(\mathsf{crs}_{\text{nizk}}, \mathsf{stmt} = (u, Z_u), w = (m_u, \sigma, \rho); \mathsf{coins}),$$

  now we instead call the simulated prover

$$\pi_u \leftarrow \mathsf{NIZK.P}_0(\mathsf{crs}_{\text{nizk}}, \tau, \mathsf{stmt} = (u, Z_u); \mathsf{coins}).$$

  Note that P_0 uses the trapdoor τ but does not use the witness to output a simulated proof;

- Whenever a node u newly becomes corrupt and the experiment needs to explain the random coins used earlier by u, it calls the NIZK's Explain algorithm, that is,

$$\mathsf{NIZKcoins} \leftarrow \mathsf{NIZK.Explain}\,(\mathsf{crs}_{\text{nizk}}, \tau, \mathsf{stmt} = (u, Z_u), w = (m_u, \sigma, \rho); \mathsf{coins})$$

  to output an explanation of the coins used in generating the earlier NIZK proofs. These coins are returned to $\mathcal{A}$ along with all other random coins consumed by the newly corrupted node u earlier.

Claim 5.1.6. *Assume that the* NIZK *scheme satisfies non-erasure computational zero-knowledge. Then, the outputs of the experiments* $\mathsf{Expt}(1^\lambda, \{m_u^*\}_{u\in[n]})$ *and* $\mathsf{Hyb}(1^\lambda, \{m_u^*\}_{u\in[n]})$ *are computationally indistinguishable.*

Proof. Follows directly from the computational zero-knowledge of the NIZK system. □

Experiment $\mathsf{Ideal}(1^\lambda)$. The ideal experiment $\mathsf{Ideal}(1^\lambda)$ is almost identical to $\mathsf{Hyb}(1^\lambda, \_)$, except with the following modification:

- At the beginning of the challenge phase, let H^* denote the so-far honest nodes. We compute the first-round message $(\mathsf{puz}, Z_u, \pi_u)$ for every $u \in H^*$ as below: call $Z_u \leftarrow \mathsf{TLP.Gen}(1^\lambda, \mathcal{T}_{\text{solve}}, (0, \sigma))$ where $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_u, 0)$. Further, call the NIZK's simulated prover $\widetilde{\mathsf{P}}$ which uses τ but not the witness to generate a simulated proof π_u.

Claim 5.1.7. *Assume that the* TLP *scheme satisfies ξ-hardness. Then, for any $\{m_u^*\}_{u\in[n]}$, no non-uniform parallel $2\mathcal{T}_\emptyset$-bounded distinguisher $\mathcal{D}$ can distinguish the outputs of the experiments* $\mathsf{Ideal}(1^\lambda)$ *and* $\mathsf{Hyb}(1^\lambda, \{m_u^*\}_{u\in[n]})$, *except with negligible probability.*

Proof. We can consider a sequence of hybrids for $i \in [0, h]$, such that in the i-th hybrid, during the challenge session, the first $\min(i, |H^*|)$ nodes in H^* (by lexicographical ordering) will use the input $(0, \sigma)$ where $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_u, 0)$ to compute a puzzle. If there exists a non-uniform parallel $2\mathcal{T}_\emptyset$-bounded distinguisher $\mathcal{D}$ that can distinguish the outputs of the experiments $\mathsf{Ideal}(1^\lambda)$ and $\mathsf{Hyb}(1^\lambda, \{m_u^*\}_{u\in[n]})$ with more than negligible probability, by the hybrid argument, there must exist a pair of adjacent hybrids indexed j and $j + 1$ that $\mathcal{D}$ can distinguish with non-negligible probability.

We can construct a non-uniform $p.p.t.$ parallel machine $\mathcal{B}$ which breaks the ξ-hardness of the TLP scheme. $\mathcal{B}$ simulates the experiment for $\mathcal{A}$ and let u be the $(j + 1)$-th node in H^* at the beginning of the challenge session. At the beginning of the challenge session, for every v that is among the first j nodes in H^*, $\mathcal{B}$ computes their puzzles using the input $(0, \sigma)$ where

$\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_v, 0)$; and for everyone else in H^* that is not among the first $j+1$ nodes, $\mathcal{B}$ will compute their puzzles using the input (m_v^*, σ') where $\sigma' \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_v, m_v^*)$.

After this computation is done, $\mathcal{B}$ now computes the first-round message for the $(j+1)$-th node in H^*. To do so, $\mathcal{B}$ interacts with a TLP challenger which either returns a puzzle either for the string $(0, \sigma)$ where $\sigma \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_u, 0)$, or for the string (m_u^*, σ') where $\sigma' \leftarrow \Sigma.\mathsf{Sign}(\mathsf{ssk}_u, m_u^*)$. This answer will be used as the puzzle for the $(j+1)$-th node in H^*.

At this moment, $\mathcal{B}$ gives the view of $\mathcal{A}$, including all random coins consumed by $\mathcal{A}$ and all outputs of $\mathcal{A}$ so far, as well as the first-round messages of H^* in the challenge session to the distinguisher $\mathcal{D}$, and in at most $2\mathcal{T}_\emptyset$ time, it outputs the same answer as $\mathcal{D}$. $\square$

At this moment, by the hybrid argument, we have that no non-uniform $p.p.t.$ $2\mathcal{T}_\emptyset$-bounded parallel machine $\mathcal{D}$ can distinguish the outputs of $\mathsf{Expt}^{\mathcal{A}}(1^\lambda, \{m_u^*\}_{u\in[n]})$ and $\mathsf{Ideal}(1^\lambda)$ with more than negligible probability. By a symmetric argument, the same holds for $\mathsf{Expt}^{\mathcal{A}}(1^\lambda, \{\widetilde{m}_u^*\}_{u\in[n]})$ and $\mathsf{Ideal}(1^\lambda)$. Thus, we can conclude that no non-uniform $p.p.t.$ $2\mathcal{T}_\emptyset$-bounded parallel machine $\mathcal{D}$ can distinguish the outputs of $\mathsf{Expt}^{\mathcal{A}}(1^\lambda, \{m_u^*\}_{u\in[n]})$ and $\mathsf{Expt}^{\mathcal{A}}(1^\lambda, \{\widetilde{m}_u^*\}_{u\in[n]})$ with more than negligible probability.

5.2 Byzantine Broadcast Protocol

5.2.1 Protocol

Without loss of generality, we may assume that $u = 1$ is the designated sender for the Byzantine Broadcast. Our protocol will make use of a VRF scheme which is defined in Section 2.1.5, and will rely on the Distribute protocol that is defined and constructed in Section 5.1.

Vote. A vote from a node $u \in [n]$ for the bit $b \in \{0,1\}$ is a tuple of the form $(\texttt{vote}, b, u, D, \sigma)$ such that $\mathsf{VRF.Vf}(\mathsf{crs}_{\mathrm{vrf}}, \mathsf{pk}_u, b, D, \sigma) = 1$, and moreover, it must be that either $D < D_p$ or $u = 1$. Here, D_p denotes a *difficulty parameter* whose choice will be specified shortly.

Batch of votes. An r-batch of votes for a bit $b \in \{0,1\}$ is a collection of valid votes from r distinct nodes, and moreover, it must be that one of these votes comes from the designated sender.

Protocol. Our Byzantine Broadcast protocol is described below. Recall that $h = n - f$ denotes the number of honest nodes.

Initially, every node u's $\mathsf{Extracted}_u$ set is set to $\emptyset$. The designated sender $u = 1$ computes and records a vote for its input bit b by computing $(D, \sigma) \leftarrow \mathsf{VRF.Eval}(\mathsf{crs}_{\mathrm{vrf}}, \mathsf{sk}_1, b)$.

Parameters. Let ℓ be the length of the first term of the VRF's evaluation outcome. The difficulty parameter D_p is set such that the probability that a random string of length ℓ is less than D_p with probability $p \in (\frac{\log^2\lambda}{h}, \frac{3\log^2\lambda}{h}) \cap (0,1)$. The number of phases $R := 6\log^2\lambda \cdot \frac{n}{h}$.

Setup. We use two instances of the Distribute protocol, denoted $\mathsf{Distribute}^0$ and $\mathsf{Distribute}^1$ respectively, and each instance is used by nodes to distribute batches of votes for the bit 0 and 1, respectively. For $b \in \{0,1\}$, call the setup of Distribute which outputs $(\mathsf{crs}^b_{\mathrm{distr}}, \{\mathsf{vk}^b_u, \mathsf{ssk}^b_u\}_{u\in[n]})$. Call $(\mathsf{crs}_{\mathrm{vrf}}, \{\mathsf{pk}_u, \mathsf{sk}_u\}_{u\in[n]}) \leftarrow \mathsf{VRF.Gen}(1^\lambda)$. Now, publish the public parameters $(\mathsf{crs}^0_{\mathrm{distr}}, \mathsf{crs}^1_{\mathrm{distr}}, \mathsf{crs}_{\mathrm{vrf}})$ and give each node u the secret keys ssk^0_u, ssk^1_u, and sk_u.

Phase $r \in [1 \ldots R]$**.** Each phase consists of $R_{\text{distr}} + 1$ rounds where R_{distr} denotes the round complexity of the $\mathsf{Distribute}$ protocol.

1. In the first round, every node u performs the following: for each bit $b \in \{0, 1\}$, if node u has seen a valid r-batch of votes for b and $b \notin \mathsf{Extracted}_u$, then it multicasts any such r-batch for b to everyone, and sets $\mathsf{Extracted}_u \leftarrow \mathsf{Extracted}_u \cup \{b\}$.

2. The next step lasts for R_{distr} rounds. Each node $u \neq 1$ does the following: for each bit $b \in \{0, 1\}$, it invokes a new session of the $\mathsf{Distribute}^b$ protocol with a new session identifier r to distribute either an $(r + 1)$-batch of votes or a dummy message:

   - If it has recorded a valid r-batch of votes for b and node u has never computed a vote for b before, then it attempts to vote for b by computing $(D, \sigma) \leftarrow \mathsf{VRF.Eval}(\mathsf{crs}_{\text{vrf}}, \mathsf{sk}_u, b)$. If $D < D_p$, then execute the following:
     - let $\mathsf{Extracted}_u \leftarrow \mathsf{Extracted}_u \cup \{b\}$;
     - invoke $\mathsf{Distribute}^b$ to distribute a valid $(r + 1)$-batch of votes for b, possibly by adding its own vote $(\mathtt{vote}, b, u, D, \sigma)$.
   - Else, if the node did not decide to distribute an $(r + 1)$-batch of votes for b in the above, then invoke $\mathsf{Distribute}^b$ to distribute a dummy message $\perp$ which is encoded as a string of the same length as an $(r + 1)$-batch of votes for b.

At any time, if any valid vote is received over the network or output by the $\mathsf{Distribute}^0$ or $\mathsf{Distribute}^1$ protocols, the vote is then recorded by the node.

Output. At the end of the R phases, if $|\mathsf{Extracted}_u| = 1$ node u outputs the unique bit in $\mathsf{Extracted}_u$; else output the default bit 0.

Round complexity. The total round complexity of the above protocol is upper bounded by $R \cdot R_{\text{distr}} = \left(\frac{n}{h}\right)^2 \cdot \frac{1}{\xi} \cdot \mathsf{poly} \log \lambda$. As a special case, in the case 99% or any arbitrarily large constant fraction of nodes are corrupt, and assuming that the hardness parameter ξ is a constant, the round complexity of the protocol is $\mathsf{poly} \log \lambda$.

5.3 Proofs for our Byzantine Broadcast Protocol

5.3.1 Additional Terminology

For convenience, we will use the following terminology.

- We say that an execution satisfies consistency for the bit $b \in \{0, 1\}$, iff the following holds: if some forever-honest node u has b in its $\mathsf{Extracted}_u$ set at the end, then every forever-honest node v has b in its $\mathsf{Extracted}_v$ set at the end, too. To show consistency, we just have to prove that except with negligible probability over the choice of the randomized execution, consistency holds for $b = 0$ as well as $b = 1$.

- For convenience, we say that a node u *mines* a vote for $b \in \{0, 1\}$ if it calls $(D, \sigma) \leftarrow \mathsf{VRF.Eval}(\mathsf{crs}_{\text{vrf}}, \mathsf{sk}_u, b)$ to attempt to compute a vote for b, and recall that whether a mining attempt is successful depends on whether the outcome D is less than the difficulty parameter D_p. All honest mining attempts are made in the second round of some phase, i.e., the first round of the Distribute protocol of that phase.

- We say that a node $u \in [n]$ is in the 0-committee iff the first term in the output of $\mathsf{VRF.Eval}(\mathsf{crs}_{\mathrm{vrf}}, \mathsf{sk}_u, 0)$ is smaller than D_p. Members of the 1-committee is similarly defined.

We will consider two types of bad events. We will later prove that if, except with negligible probability, neither type of bad event happens for both bits, then consistency is respected except with negligible probability.

- *Type A bad event for the bit* b: In the second round of some phase r, all so-far honest nodes have made attempts to mine a vote for b (in the second round of some phase $r' \leq r$); and yet, the adversary $\mathcal{A}$ manages to corrupt every member of the b-committee either before it even made a mining attempt for b, or by the beginning of the third round of the phase in which it makes a mining attempt for b.
- *Type B bad event for the bit* b: Either at least R nodes are members of the b-committee, or no node is a member of the b-committee.

5.3.2 Proof Overview: Challenges and Intuition

We would like to use probablistic reasoning to argue that the above two types of bad events do not happen except with negligible probability. The probabilistic reasoning could be accomplished using standard measure concentration bounds if all the cryptography we employ were "ideal". Unfortunately, the cryptography we employ is far from ideal. One problem we encounter is that our delayed-exposure distribution primitive Distribute guarantees secrecy against only an adversary who is restricted to run in a small number of parallel steps. An adversary who can take more parallel steps can completely break the secrecy of Distribute by solving the time-lock puzzles. For our final protocol, of course, we want to prove security against any *parallel* $p.p.t.$ *adversary* who is allowed to take an *unbounded* number of parallel steps.

Partly due to this reason, the most natural proof strategy completely fails: we are not able describe an ideal protocol (without cryptography), and show that the real-world protocol *securely emulates* the ideal protocol by a standard, *simulation-based* notion. What comes to the rescue is that we only need to prove that certain security properties hold in the real-world protocol; and proving these properties eventually boils down to showing that certain bad events (as defined above) happen with negligible probability. Therefore, instead of proving that the real-world protocol *securely emulates* some ideal protocol, our strategy is to prove that the probability of bad events is not higher in the real-world protocol than in the ideal protocol (barring negligible differences). To do this, we will define a polynomially long sequence of hybrids, starting from the real-world protocol, all the way to the ideal protocol which does not have any cryptography: we will prove that for every pair of adjacent hybrids, the probability of bad events in the former is not higher than the probability of bad events in the latter (barring negligible differences).

We now elaborate on our blueprint — below in our proof overview, we mainly focus on how we bound the probability of Type-A bad events since this is the most technical part of the proof. First, we make several modifications to the real-world protocol and obtain a hybrid called $\mathsf{Hyb}'_{\star}$. $\mathsf{Hyb}'_{\star}$

is no longer a consensus protocol, it should simply be viewed as a game in which the adversary $\mathcal{A}$ is trying to cause Type-A bad events to happen. The modifications we made ensure that the probability of Type-A bad events can only increase in $\mathsf{Hyb}'_\star$ in comparison with the real-world protocol. Importantly, in $\mathsf{Hyb}'_\star$, we introduce a *final guessing phase*: if at the end of the protocol, $\mathcal{A}$ has corrupted $f' < f$ number of nodes, i.e., it has more corruption budget left, we give $\mathcal{A}$ an extra opportunity to guess who, among the remaining honest nodes that have not made a mining attempt for b, are members of the b-committee. If $\mathcal{A}$ can correctly guess all remaining honest b-committee members in at most $f - f'$ tries, we also declare that $\mathcal{A}$ wins, i.e., a Type-A bad event has happened.

At this moment, it is not clear why we introduce the final guessing phase yet. This will become clear in the next hybrid $\mathsf{Hyb}_\star$. In $\mathsf{Hyb}_\star$, we modify the final guessing phase, such that the remaining honest nodes who have not made a mining attempt for b yet would use true random coins rather than VRFs to determine if they are a member of the b-committee. In this way, the game becomes ideal (i.e., without cryptography) after the final guessing phase starts. Partly relying on the security of VRF, one can show that any parallel $p.p.t.$ $\mathcal{A}$ cannot cause Type-A bad events to happen more often in $\mathsf{Hyb}'_\star$ than in $\mathsf{Hyb}_\star$ (barring negligible differences).

Now, in the remainder of the proof, our idea is to start from the end of the experiment, and make each phase "ideal" one by one. In other words, in each hybrid, we will make the final guessing phase start one phase earlier, until at the very end, the final guessing phase starts upfront and therefore the whole game becomes ideal (i.e., no cryptography). In the process, we make sure that $\mathcal{A}$'s probability of causing bad events does not decrease (barring negligible differences).

At this moment, it is a good time to revisit how we can overcome the aforementioned problem where the overall adversary is a parallel machine running in unbounded parallel steps but our $\mathsf{Distribute}$ primitive only gives secrecy against an adversary who is restricted to run in a small number of parallel steps. With the above proof strategy, informally speaking, at some point, we need to compare the probability of Type-A bad event in the following two adjacent hybrids — henceforth let r^* be the phase immediately preceeding the final guessing phase:

1. in the first hybrid, in phase r^*, honest nodes run the real $\mathsf{Distribute}$ protocol using real inputs;

2. in the second hybrid, in phase r^*, honest nodes run the $\mathsf{Distribute}$ protocol using input $\vec{0}$.

In both of these hybrids, the adversary $\mathcal{A}$ can win the game if it either wins in the final guessing phase, or if $\mathcal{A}$ can guess, by the beginning of the third round of phase r^*, which honest nodes successfully made mining attempts for b in phase r^*. To succeed in the latter, $\mathcal{A}$ might try to gain some leverage by attacking the $\mathsf{Distribute}$ protocol of phase r^*, but because of the short-fuse deadline $\mathcal{A}$ must make the guess by, the $\mathsf{Distribute}$ protocol in phase r^* is unhelpful to $\mathcal{A}$ due to the momentary secrecy property. Even though after phase r^*, $\mathcal{A}$ may completely break the secrecy of the $\mathsf{Distribute}$ protocol of phase r^*, recall that the game becomes ideal immediately after phase r^*. Therefore, breaking the secrecy of the phase-r^* $\mathsf{Distribute}$ protocol no longer helps $\mathcal{A}$ after phase r^*.

Last but not the least, besides the aforementioned technicalities, yet another is that $\mathcal{A}$ is adaptive, and we need to handle the adaptivity with particular care in our proofs. Now, without further ado, we present our formal proofs.

5.3.3 Bounding the Probability of Type-A Bad Events

Let Real denote an execution of the real-world protocol in which the adversary $\mathcal{A}$'s goal is to cause a Type-A bad event to happen. In the remainder of this section, we will consider a sequence of hybrid experiments starting with Real such that for each pair of adjacent experiments, the probability of a Type-A bad event in the latter is an upper bound of the probability of a Type-A bad event in the former (ignoring negligible differences). In the end, we will upper bound the probability of a Type-A bad event in the final experiment called Hyb_1. Hyb_1 essentially gets rid of the cryptography and therefore we can upper bound the probability of a Type-A bad event in Hyb_1 with a simple information-theoretic, probabilistic argument.

In the following we fix an arbitrary $b \in \{0, 1\}$, and we care about bounding Type-A bad events for the bit b. Henceforth whenever we say Type-A bad event, it means a Type-A bad event for the bit b. Also, recall that we use the notation $f = n - h$ to denote the maximum number of corruptions allowed.

Experiment $\mathsf{Hyb}'_\star$

Since we only care about bounding Type-A bad events for the bit b, we make some simplifications to the protocol without decreasing the probability of a Type-A bad event. We therefore define a hybrid experiment $\mathsf{Hyb}'_\star$:

1. At the beginning of the protocol, for each $u \in [n]$, we compute $(D_u, \sigma_u) \leftarrow \mathsf{VRF.Eval}(\mathsf{crs}_{\mathrm{vrf}}, \mathsf{sk}_u, 1 - b)$ and disclose to $\mathcal{A}$ the pair (D_u, σ_u) which is the evaluation outcome and proof for the bit $1 - b$. Of course, upon the new corruption of some node v, we now need to explain to $\mathcal{A}$ the coins in the above evaluation for v too.

2. During the protocol, in each phase, we only run the $\mathsf{Distribute}^b$ protocol but not the $\mathsf{Distribute}^{1-b}$ protocol; similarly, we need not run the setup for $\mathsf{Distribute}^{1-b}$ either.

3. If some honest node u tried to call $\mathsf{Distribute}^b$ to send a valid $(r + 1)$-batch of votes for b in the second round of phase r and u remains honest till the beginning of the third round of phase r, the experiment declares that $\mathcal{A}$ has failed to cause a Type-A bad event, and simply aborts, outputting $\mathsf{adv\text{-}fail}$.

4. Immediately after the second round of phase R, for every honest node u who has already made a mining attempt for b, we disclose its VRF secret key sk_b to $\mathcal{A}$ even if u has not been corrupted by $\mathcal{A}$ (and note that these nodes do not count towards the corruption budget).

   Now, we allow $\mathcal{A}$ an extra **final guessing phase**, in which $\mathcal{A}$ can adaptively specify nodes to corrupt one by one; all nodes specified must not have made a mining attempt for b yet. Every time $\mathcal{A}$ specifies a new node u to corrupt, it learns its VRF secret key sk_u. The experiment stops when $\mathcal{A}$ has made f corruption queries in total. At this moment, if A has corrupted all members of the b-committee who have not made a mining attempt for b at the beginning of the final guessing phase, then declare that a Type-A bad event has happened.

Claim 5.3.1. *If for some non-uniform $p.p.t.$ parallel machine $\mathcal{A}$, a Type-A bad event happens with probability μ in the real-world experiment* Real*, then there exists a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ such that a Type-A bad event happens with probability at least μ in* $\mathsf{Hyb}'_\star$.

Proof. We can add these modifications one by one and in each step argue that if there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ in the previous experiment that can cause a Type-A bad event to occur with probability μ, then there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ in the modified experiment that can cause a Type-A bad event to occur with probability at least $\mu - \mathsf{negl}(\lambda)$.

First, we can add the modification 4. It is not hard to see that this phase only gives the adversary more opportunities in causing a Type-A bad event. In some sense, the final guessing phase is saying, at the end of the protocol, if not so-far honest nodes have made mining attempts for b (in this case a Type-A bad event cannot happen), we will pretend as if all of them made mining attempts for b and give $\mathcal{A}$ another opportunity to guess who the b-committee members are.

With modification 4, we can essentially imagine that all so-far honest nodes will have made mining attempts for b by the end of the protocol. Therefore, modification 3 is cosmetic, it basically checks for Type-A bad events constantly in the background and does not change the probability of Type-A bad event.

Next, we can add modifications 1 and 2. It is not hard to see that if there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ in the previous experiment that can cause a Type-A bad event to occur with probability μ, then we can construct a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ in the modified experiment that can cause a Type-A bad event to occur with probability at least μ. Basically $\mathcal{A}'$ simply simulates the $\mathsf{Distribute}^{1-b}$ instances for $\mathcal{A}$ using its knowledge of all the VRF evaluations and proofs for $1-b$. Whenever $\mathcal{A}$ corrupts some v, $\mathcal{A}'$ learns the explanation of the coins that contributed towards v's VRF proofs for $1-b$, in this way, $\mathcal{A}'$ can provide the necessary explanation to $\mathcal{A}$ too.

□

Experiment $\mathsf{Hyb}_\star$

Experiment $\mathsf{Hyb}_\star$ is almost the same as $\mathsf{Hyb}'_\star$, except that when the final guessing phase starts, every so-far honest node who has not made a mining attempt for b yet chooses a random D from an appropriate domain instead of using the honest VRF outcome to determine whether it is a member of the b-committee. Furthermore, during the final guessing phase, when $\mathcal{A}$ corrupts any node, we no longer disclose the node's VRF key to $\mathcal{A}$.

Lemma 5.3.2. *Assume that the VRF scheme satisfies pseudorandomness under selective opening (see Definition 2.1.5). Then, suppose that there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ that can cause a Type-A bad event to happen in $\mathsf{Hyb}'_\star$ with probability μ, then there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ that can cause a Type-A bad event to happen in $\mathsf{Hyb}_\star$ with probability at least $\mu - \mathsf{negl}(\lambda)$ for some appropriate negligible function $\mathsf{negl}(\cdot)$.*

Proof. We prove this lemma through a sequence of intermediate hybrids described below.

Hybrid $\widetilde{\mathsf{H}}_\star$. The experiment $\widetilde{\mathsf{H}}_\star$ is defined in almost the same way as $\mathsf{Hyb}'_\star$ except with the following modifications:

- During the setup, we will replace the VRF's setup with the simulated setup algorithms $\mathcal{S}_0$ which generates the $(s_1, \ldots, s_n)$ part of the secret keys, and $\mathcal{S}_1$ which generates $(\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \tau)$. The adversary $\mathcal{A}$ is given the public components $\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n$.

- Whenever an honest node u needs to evaluate the VRF on input b', we compute the VRF outcome y honestly using sk_u, but call $\mathcal{S}_2(\tau, \mathsf{pk}_u, b', y)$ instead to compute a simulated proof using the trapdoor τ.
- Whenever an honest node u gets corrupted, we call the $\mathcal{S}_3$ algorithm which returns ρ_u and all the randomness u used in earlier VRF evaluations. Now return $\mathsf{sk}_u := (s_u, \rho_u)$ to $\mathcal{A}$. If this is before the final guessing round, also return the random coins output by $\mathcal{S}_3$ to $\mathcal{A}$, as well as randomness the newly corrupted node used in the Distribute protocol instances so far. Immediately after the second round of phase R, call $\mathcal{S}_3$ for every honest node v who has already made a mining attempt for b, and return s_u and the term ρ_v output by $\mathcal{S}_3$ to $\mathcal{A}$.

Claim 5.3.3. *Assume that the VRF scheme satisfies pseudorandomness under selective opening (see Definition 2.1.5). Then, $\mathcal{A}$'s views in $\widetilde{\mathsf{H}}_\star$ and $\mathsf{Hyb}'_\star$ are computationally indistinguishable.*

Proof. Follows directly from the second part of the definition of pseudorandomness under selective opening. □

Hybrid $\widetilde{\mathsf{H}}_f$. The experiment $\widetilde{\mathsf{H}}_f$ is almost the same as $\widetilde{\mathsf{H}}_\star$ except with the following modification: when the last node u becomes corrupt during the final guessing phase, for u and all remaining honest nodes who have not made a mining attempt for b, we choose a random number of appropriate length to determine whether the node is in the b-committee. Moreover, for the last corruption u in the final guessing stage, we do not disclose u's secret key or coins to $\mathcal{A}$.

Claim 5.3.4. *Assume that the VRF scheme satisfies pseudorandomness under selective opening. Then, if there exists a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_\star$ with probability μ, then there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_f$ with probability at least $\mu - \mathsf{negl}(\lambda)$ for some negligible function* $\mathsf{negl}(\cdot)$.

Proof. Whenever $\mathcal{A}$ makes the last corruption query during the final guessing phase, whether a Type-A bad event has occurred is already determined no matter whether we disclose the secret key of the newly corrupt node to $\mathcal{A}$. Therefore, henceforth, we simply assume that nothing is disclosed to $\mathcal{A}$ upon the last corruption during the final guessing phase.

Basically, we define $\mathcal{A}'$ to be the same as $\mathcal{A}$ and it runs $\mathcal{A}$ till it makes the last corruption query during the final guessing phase. We show that if $\mathcal{A}$ can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_\star$ with more than negligibly higher probability than $\mathcal{A}'$ in experiment $\widetilde{\mathsf{H}}_f$, we can construct a reduction $\mathcal{B}$ to break the first game in the definition of pseudorandomness under selective opening.

Essentially, $\mathcal{B}$ interacts with its challenger $\mathcal{C}$ as defined in the first game in the definition of pseudorandomness under selective opening. Moreover, $\mathcal{B}$ simulates the experiment $\widetilde{\mathsf{H}}_\star$ to $\mathcal{A}$ right till the moment $\mathcal{A}$ makes the last corruption query in the final guessing phase.

- During setup, $\mathcal{B}$ asks its challenger $\mathcal{C}$ to run the setup who generates $(s_1, \ldots, s_n)$. $\mathcal{B}$ now runs $\mathcal{S}_1$ to generate $\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n, \tau$ and it gives the terms $\mathsf{crs}, \mathsf{pk}_1, \ldots, \mathsf{pk}_n$ to $\mathcal{A}$.
- Whenever the experiment needs to evaluate the first term of the VRF outcome, $\mathcal{B}$ instead forwards the query to its challenger $\mathcal{C}$, and then it simulates the proof part by calling $\mathcal{S}_2$ just like in $\widetilde{\mathsf{H}}_\star$.

- Whenever $\mathcal{A}$ corrupts an honest node u (except for the last corruption query in the final guessing phase), $\mathcal{B}$ issues a corruption query to $\mathcal{C}$, learns s_u, and then calls the $\mathcal{S}_3$ algorithm which returns ρ_u and all the randomness u used in earlier VRF evaluations. Now, return $\mathsf{sk}_u := (s_u, \rho_u)$ to $\mathcal{A}$, and if this is before the final guessing round, also return the random coins output by $\mathcal{S}_3$ to $\mathcal{A}$, as well as the coins u used in Distribute protocol instances so far.

- Immediately after the second round of phase R, for every honest node v who has already made a mining attempt for b, $\mathcal{B}$ issues a corruption query to $\mathcal{C}$ for v, learns the s_v, and then calls $\mathcal{S}_3$ to obtain ρ_v. It returns the pair (s_v, ρ_v) to $\mathcal{A}$.

- When $\mathcal{A}$ makes the last corruption query during the final guessing stage, $\mathcal{B}$ sends multiple challenge queries on the input b to $\mathcal{C}$ to obtain the evaluation outcomes for the last corrupted node, as well as all remaining honest nodes who have not made a mining attempt for b.

Besides the above, $\mathcal{B}$ simulates the rest of the $\widetilde{\mathsf{H}}_\star$ faithfully.

These evaluation outcomes are used to determine whether a Type-A event has happened at this moment. Note that if $\mathcal{C}$ returned random answers, the experiment is the same as $\mathcal{A}'$ interacting with $\widetilde{\mathsf{H}}_f$; otherwise it is the same as $\mathcal{A}$ interacting with $\widetilde{\mathsf{H}}_\star$.

We stress that in the above, we are using a multi-challenge version of the first-game of the pseudorandomness under selective opening notion, where in the challenge phase, the adversary can specify multiple challenge queries rather than a single one. As argued in Chan et al. [54], the multi-challenge version is equivalent to the single challenge version by a standard hybrid argument. □

Hybrid $\widetilde{\mathsf{H}}_{f-1}$. The experiment $\widetilde{\mathsf{H}}_{f-1}$ is almost the same as $\widetilde{\mathsf{H}}_f$ except with the following modification: when the second to last corruption query u is made in the final guessing phase (or if fewer than 2 corruption queries are made in the final guessing phase, then for all of them):

- we do not return anything to $\mathcal{A}$ upon the corruption query; and
- we use a random number for the node u as well as all remaining honest nodes who have not made a mining attempt for b, to determine if the corresponding node is a member of the b-committee.

Claim 5.3.5. *Assume that the VRF scheme satisfies pseudorandomness under selective opening. If there exists a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_f$ with probability μ, then there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_{f-1}$ with probability at least $\mu - \mathsf{negl}(\lambda)$ for some negligible function $\mathsf{negl}(\cdot)$.*

Proof. Basically, $\mathcal{A}'$ runs $\mathcal{A}$ till it makes the second to last corruption query u in the final guessing phase. $\mathcal{A}'$ also makes the same corruption query u as the second to last query, but for the last corruption query, it just chooses to corrupt an arbitrary honest node that has not made a mining attempt for b.

We can construct a reduction $\mathcal{B}$ in a similar way as in the proof of Claim 5.3.4, except that now $\mathcal{B}$ stops at the second to last corruption query (for the node u) in the final guessing stage, and then $\mathcal{B}$ asks $\mathcal{C}$ for the evaluation outcome on the input b for u as well as any remaining honest node who has not made a mining attempt for b. The returned evaluation outcomes are used to decide whether the corresponding node is a member of the b-committee. It is not hard to see that if $\mathcal{C}$ returns random answers, the experiment above would have the same probability of causing a Type-A bad event as in $\widetilde{\mathsf{H}}_{f-1}$; else it has the same probability of causing a Type-A bad event as in $\widetilde{\mathsf{H}}_f$. □

Hybrid $\widetilde{\mathsf{H}}_1$. We can define a sequence of hybrids $\widetilde{\mathsf{H}}_f$, $\widetilde{\mathsf{H}}_{f-1}$, ..., $\widetilde{\mathsf{H}}_1$, until eventually we arrive at $\widetilde{\mathsf{H}}_1$, which is almost the same as $\mathsf{Hyb}_\star$ except that we are using the simulated setup and simulated VRF proofs in $\widetilde{\mathsf{H}}_1$. Specifically, in experiment $\widetilde{\mathsf{H}}_i$, when $\mathcal{A}$ is making the last but $(f-i+1)$-th corruption query in the final guessing phase, we switch to using random outcomes for all remaining honest nodes who have not made a mining attempt for b (including the one being corrupted right now), to determine if the corresponding node is in the b-th committee; and moreover at this moment we do not disclose anything more to $\mathcal{A}$.

Claim 5.3.6. *Assume that the VRF scheme satisfies pseudorandomness under selective opening. If there exists a non-uniform $p.p.t.$ parallel machine $\mathcal{A}$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_i$ with probability μ, then there is a non-uniform $p.p.t.$ parallel machine $\mathcal{A}'$ that can cause a Type-A bad event to happen in $\widetilde{\mathsf{H}}_{i-1}$ with probability at least $\mu - \mathsf{negl}(\lambda)$ for some negligible function* $\mathsf{negl}(\cdot)$.

Proof. The proof is essentially identical to that of Claim 5.3.5. □

Claim 5.3.7. *Assume that the VRF scheme satisfies pseudorandomness under selective opening. Then, $\mathcal{A}$'s views in $\widetilde{\mathsf{H}}_1$ and $\mathsf{Hyb}_\star$ are computationally indistinguishable.*

Proof. Directly follows from the second part of the pseudorandomness under selective opening notion. □

With the above sequence of hybrid experiments, we have concluded the proof of Lemma 5.3.2. □

Experiments Hyb'_r and Hyb_r

We define a sequence of hybrids below $\{\mathsf{Hyb}'_r, \mathsf{Hyb}_r\}_{r\in[1,R]}$.

Experiment Hyb'_r. Experiment Hyb'_r is almost identical as $\mathsf{Hyb}_\star$ except the following modifications:

- In phase r of the protocol, we pretend instead that so-far honest nodes use the inputs $\vec{0}$ for the phase-r $\mathsf{Distribute}^b$ protocol, and compute their first-round messages (denoted M) of the $\mathsf{Distribute}^b$ protocol. We give M to $\mathcal{A}$, and let it run till the beginning of the third round of phase r (i.e., the second round of the phase-r $\mathsf{Distribute}$ protocol). $\mathcal{A}$ outputs, among other terms, a set of new nodes to corrupt by the beginning of the third round of phase R.
- At this moment, every remaining honest node who has not yet made a mining attempt for b will use a random string of appropriate length to determine if it is a member of the b-committee. We now let $\mathcal{A}$ engage in a final guessing phase as defined before.

The definition of a Type-A bad event in Hyb_r is the same as in $\mathsf{Hyb}_\star$.

Experiment Hyb_r**.** Experiment Hyb_r is almost the same as $\mathsf{Hyb}_\star$ except that at the beginning of the second round of phase r, the experiment discloses all honest nodes' secret keys for the $\mathsf{Distribute}^b$ instance to $\mathcal{A}$. $\mathcal{A}$ now enters the final guessing phase, in which all remaining honest nodes who have not yet made mining attempts for b switch to using random coins to determine if they are a member of the b-committee.

Claim 5.3.8. *Assume that the* $\mathsf{Distribute}$ *protocol satisfies momentary secrecy. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: if for some non-uniform* $p.p.t.$ $\mathcal{A}$*, Type-A bad events happen with probability* μ *in* $\mathsf{Hyb}_\star$*, then there must exist a non-uniform* $p.p.t.$ $\mathcal{A}'$ *such that Type-A bad events happen with probability at least* $\mu - \mathsf{negl}(\lambda)$ *in* Hyb'_R.

Proof. Consider the following experiment Expt^β in which a reduction $\mathcal{B}$ interacts with a challenger $\mathcal{C}$ as well as the adversary $\mathcal{A}$.

- For the $\mathsf{Distribute}^b$ instance, it will embed the public parameters passed to it by $\mathcal{C}$.
- Whenever $\mathcal{B}$ needs to play on behalf of honest nodes in $\mathsf{Distribute}^b$ protocols (not including in phase R), it forwards the query to $\mathcal{C}$ instead providing the inputs of the so-far honest nodes, and acts as a relay between $\mathcal{C}$ and $\mathcal{A}$ for messages of the $\mathsf{Distribute}^b$ protocol.
- Whenever some honest node is corrupted by $\mathcal{A}$, it forwards the corruption query to $\mathcal{C}$, and forwards the internal states returned by $\mathcal{C}$ to $\mathcal{A}$; besides this, $\mathcal{B}$ also gives $\mathcal{A}$ the secret keys and random coins pertaining to the VRF of the newly corrupt node.
- Finally, during phase R, $\mathcal{B}$ invokes a challenge session with $\mathcal{C}$. Depending on the bit β, $\mathcal{C}$ will either use the honest nodes' real inputs in the challenge $\mathsf{Distribute}^b$ protocol if $\beta = 0$; else if $\beta = 1$, it will use the vector $\vec{0}$ as honest inputs to the challenge $\mathsf{Distribute}^b$ protocol. Let M^β be the first-round messages of the challenge $\mathsf{Distribute}^b$ protocol computed by $\mathcal{C}$. Let view^β be the joint view of $\mathcal{A}$ and $\mathcal{B}$ at this point.
- ($\diamond$): Now, give M^β to $\mathcal{A}$, run it till the beginning of the next round, and $\mathcal{A}$ outputs, among other terms, a set K^β of nodes to corrupt.
- At this moment, any remaining honest node who has not made a mining attempt for b uses random coins to decide if it is a member of the b-committee, and we let $\mathcal{A}$ engage in the final guessing phase.

Besides the above, $\mathcal{B}$ simply runs the experiment $\mathsf{Hyb}_\star$ faithfully. Observe that if $\beta = 0$, the experiment is the same as $\mathsf{Hyb}_\star$ till the beginning of the third round of phase R (i.e., second round of the phase-R $\mathsf{Distribute}$ protocol); else if $\beta = 1$, the experiment is the same Hyb'_R till the beginning of the third round of phase R.

Let X denote the total number of honest nodes who have not made a mining attempt for b by the beginning of the third round of phase R, let Y denote the total number of nodes corrupted by the beginning of the third round of phase R, and let Z be a bit indicating whether at the beginning of the third round of phase R, $\mathsf{adv\text{-}fail}$ has occurred — recall that if the set K does not contain all honest b-committee members who made a mining attempt for b in phase R, then $\mathsf{adv\text{-}fail}$ would occur.

Recall that after the beginning of the third round of phase R, the experiment enters a final guessing phase in which all honest nodes who have not made a mining attempt for b yet use random coins to decide if they are members of the b committee. To prove that the probability of Type-A bad events in Hyb'_R can must be at least as high as in $\mathsf{Hyb}_\star$ barring negligible differences, it suffices to show that for any $x \in [n]$, $0 \leq y \leq x$, and $z \in \{0,1\}$,

$$\left| \Pr_{\mathsf{Expt}^0}[X = x, Y = y, Z = z] - \Pr_{\mathsf{Expt}^1}[X = x, Y = y, Z = z] \right| \leq \mathsf{negl}(\lambda)$$

Suppose not. Then, there must exist $x \in [n]$, $0 \leq y \leq x$, and $z \in \{0,1\}$, such that $\Pr_{\mathsf{Expt}^0}[X = x, Y = y, Z = z]$ and $\Pr_{\mathsf{Expt}^1}[X = x, Y = y, Z = z]$ differ by a non-negligible amount. Now, we can construct a non-uniform $p.p.t.$ parallel $2\mathcal{T}_\emptyset$-bounded distinguisher $\mathcal{D}$ that can distinguish (M^0, view^0) and (M^1, view^1) with more than negligible probability. Basically, $\mathcal{D}$ takes M^β and view^β, and runs whatever $\mathcal{A}$ runs in the $(\diamond)$ step for exactly one round which is $\mathcal{T}_\emptyset$ amount of time. At this moment, among $\mathcal{A}$'s output, there is an additional set K of nodes to corrupt. Finally, $\mathcal{D}$ tallies the counts X, Y, and the bit Z; here the tallying includes the set K. $\mathcal{D}$ outputs 1 if $(X, Y, Z) = (x, y, z)$; else it outputs 0. The tallying can be computed in logarithmic in n parallel time. Due to our assumption on the duration of an round, an honest node must be able to process all n received messages within a round, and the tallying can be computed in a single round, that is, at most $\mathcal{T}_\emptyset$ time. Therefore, $\mathcal{D}$ runs in at most $2\mathcal{T}_\emptyset$ time in total.

Remark 9. *We stress that $\mathcal{A}$'s views are NOT computationally indistinguishable in the two hybrids since $\mathcal{A}$ can run in unbounded parallel time. We are merely arguing that the probability of Type-A bad events are not decreased by switching the phase-R* Distribute *messages. It is NOT true that $(M^0, \mathsf{view}^0, K^0)$ and $(M^1, \mathsf{view}^1, K^1)$ are computationally indistinguishable, because (M^0, K^0) and (M^1, K^1) can potentially be distinguished by an adversary running in sufficiently long time. However, any property on $(M^0, \mathsf{view}^0, K^0)$ or $(M^1, \mathsf{view}^1, K^1)$ that can be checked in small parallel runtime should happen with almost the same probability regardless of the choice of β.*

□

Claim 5.3.9. *Assume that the VRF scheme satisfies pseudorandomness under selective opening. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: if for some non-uniform $p.p.t.$ $\mathcal{A}$, Type-A bad events happen with probability μ in Hyb'_R, then there must exist a non-uniform $p.p.t.$ $\mathcal{A}'$ such that Type-A bad events happen with probability at least $\mu - \mathsf{negl}(\lambda)$ in Hyb_R.*

Proof. First, disclosing all honest secret keys for the Distribute^b protocol at the beginning of the second round of phase R discloses strictly more information to $\mathcal{A}$ than in the earlier Hyb'_R. Next, we can repeat the same argument as in the proof of Lemma 5.3.2, i.e., we can switch to using random coins to decide whether the following nodes are members of the b-committee: the nodes that remain honest at the beginning of the second round of phase R and have not yet made any mining attempts for b. □

Claim 5.3.10. *Assume that the* Distribute *protocol satisfies momentary secrecy, and that the VRF scheme satisfies pseudorandomness under selective opening. Then, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that the following holds: if for some non-uniform $p.p.t.$ $\mathcal{A}$, Type-A bad events happen with probability μ in $\mathsf{Hyb}_\star$, then there must exist a non-uniform $p.p.t.$ $\mathcal{A}'$ such that Type-A bad events happen with probability at least $\mu - \mathsf{negl}(\lambda)$ in Hyb_1.*

Proof. The proof works through a sequence of hybrids from Hyb_R to Hyb'_{R-1}, to Hyb_{R-1}, to Hyb'_{R-2} and so on, where to argue each adjacent pair of hybrids we use either the same proof as Claim 5.3.8 or the same proof as Claim 5.3.9. $\square$

Lemma 5.3.11. *Let b be an arbitrary bit. For any non-uniform $p.p.t.$ parallel machine $\mathcal{A}$, the probability of a Type-A bad event for b in* Real *is negligibly small.*

Proof. Notice that in Hyb_1, even for an unbounded adversary making $f = n - h$ corruptions, the expected number of forever-honest nodes that belong to the b-committee is $\Theta(\log^2 \lambda)$. By the Chernoff bound, the probability that $\mathcal{A}$ succeeds in guessing and corrupting all members of the b-committee is upper bounded by $\mathsf{negl}(\lambda)$.

Now, the earlier sequence of hybrids established that the probability of a Type-A bad event for b in Real must be upper bounded by the probability of a Type-A bad event in Hyb_1 against an unbounded adversary plus $\mathsf{negl}(\lambda)$. $\square$

5.3.4 Consistency Proofs

The following lemma bounds the probability of a Type-B bad event for either $b = 0$ or $b = 1$.

Lemma 5.3.12. *Fix an arbitrary $b \in \{0, 1\}$. Assume that the VRF scheme satisfies pseudorandomness under selective opening. Then, Type-B bad events do not happen except with negligible probability.*

Proof. If we used random coins to decide if each node is a member of the b-committee, then the probability that at least $R = 6 \log^2 \lambda \cdot \frac{n}{h}$ nodes are members of the b-committee is upper bounded by a negligible function in λ by the Chernoff bound. Similarly, the probability that no node is a member of the b-committee is also negligibly small in λ. Now, rather than true randomness, we are using the VRF which gives pseudorandomness, and because the function that determines how many nodes are in the b-committee is a polynomial function on the outcomes of the VRF, it holds that the same holds when the true random coins are replaced with pseudorandom ones. $\square$

So far, we have concluded that neither Type-A nor Type-B bad events happen except with negligible probability, for either bit $b \in \{0, 1\}$. To prove consistency, it suffices to show that if neither types of bad events happen for both bits except with negligible probability, then consistency follows. This is stated and proven in the following theorem.

Theorem 5.3.13 (Consistency). *Assume that $n \geq \log^2 \lambda$, that the VRF scheme satisfies pseudorandomness under selective opening as well as unforgeability, and that the* Distribute *protocol satisfies liveness and momentary secrecy. Then, the Byzantine Broadcast protocol defined earlier in this section satisfies consistency.*

Proof. Due to Lemmas 5.3.11 and 5.3.12, the liveness of the Distribute protocol, as well as the unforgeability of the VRF, it suffices to prove that the following hold in some execution, then the execution satisfies consistency.

1. for either $b = 0$ or $b = 1$, neither Type-A nor Type-B bad events happen for b;
2. the liveness property of Distribute is never broken; and

3. for either $b = 0$ or $b = 1$, if any so-far honest node u has not made a mining attempt for b, then there is no valid vote from u for b in any honest node's view.

Observe that an inconsistency can only take place if some forever-honest node u includes a bit b in its $\mathsf{Extracted}_u$ set but some other honest node v does not include b in its $\mathsf{Extracted}_v$ set. We now consider the following cases:

- *Case 1*: u first added the bit b to its $\mathsf{Extracted}_u$ set in some phase r but *not* in the first round of phase r. According to our protocol, in phase r, u must have observed an r-batch of votes for b, made a successful mining attempt for the bit b, and moreover, it must have tried to distribute an $(r + 1)$-batch of votes for b. Since Type-B bad events do not happen and votes are not forged, it must be that $r < R$.

  Since u is forever honest, by the liveness property of Distribute, it must be that by the end of the phase-r Distribute protocol, all forever-honest nodes will have observed the $(r + 1)$-batch of votes from u. Therefore, every forever-honest node v will have added b to its $\mathsf{Extracted}_v$ set in the first round of phase $r + 1 \leq R$.

- *Case 2*: u first added the bit b to its $\mathsf{Extracted}_u$ set in some phase r but in the first round of phase r. This means that u has observed an r-batch of votes for b in the first round of phase r. Since Type-B bad events do not happen and votes are not forged, it must be that $r < R$.

  Because u is forever honest, it must be that at the beginning of the second round of phase r, all so-far nodes have observed the same r-batch of votes for b that u saw. Now, all so-far honest nodes will make a mining attempt for b if they have not done so already. Now, since Type-A and Type-B bad events do not happen, there must exist a so-far honest node v that made a successful mining attempt for the bit b in the second round of some phase $r' \leq r$, and moreover, the adversary did not yet corrupt v at the beginning of the third round of phase r'. Now, by liveness of the Distribute protocol, by the beginning of phase $r' + 1 \leq R$, all forever-honest nodes will have observed the $(r' + 1)$ batch of votes for b that v tried to distribute in phase r', and therefore, by the end of the first round of phase $r' + 1$, every forever-honest node w will have added the bit b to its $\mathsf{Extracted}_w$ set.

□

5.3.5 Validity Proofs

Theorem 5.3.14 (Validity). *Suppose that $n \geq \log^2 \lambda$, that the VRF scheme satisfies pseudorandomness under selective opening as well as unforgeability, and that the* Distribute *protocol satisfies liveness and momentary secrecy. Suppose also that the designated sender is forever-honest and its input is $b' \in \{0, 1\}$. Then, except with negligible probability, if any forever-honest node outputs b at the end of the protocol, it must be that $b = b'$.*

Proof. If the designated sender $u = 1$ is forever-honest, let b be its input bit, then node $u = 1$ must distribute a valid 1-batch of votes for b in the first round of the first phase. Thus, by the beginning of the second round of the first phase, all so-far honest nodes will have made a mining attempt for b. Because Type-A and Type-B bad events do not happen except with negligible probability, it must be that except with negligible probability, at least one node u successfully mines a vote for b in phase 1

and the node u remains honest till at least the beginning of the third round of the first phase. By the liveness property of Distribute, it must be that except with negligible probability, by the beginning of the second phase, every so-far honest node v will have observed a valid 2-batch of votes for b, and will have added the bit b to its $\mathsf{Extracted}_v$ set. Finally, validity follows by observing that due to the unforgeability of the VRF, no valid batch of votes for $1-b$ can appear in any honest node's view except with negligible probability. $\square$

Chapter 6

Conclusion

In this thesis, we presented:

1. an expected constant round Byzantine Broadcast protocol secure in the presence of corrupt majority and weakly adaptive corruptions.

2. a sublinear round Byzantine Broadcast protocol secure in the presence of corrupt majority and strongly adaptive corruptions.

3. an amortized linear communication complexity Byzantine Broadcast protocol secure in the presence of honest majority and strongly adaptive corruptions.

4. an amortized quadratic communication complexity Byzantine Broadcast protocol secure in the presence of honest majority and strongly adaptive corruptions.

Our work leaves open several exciting directions for future work:

- Recall that Garay et al. [14] show an $\Omega(n/h)$ lower bound even for randomized protocols and static corruptions. Our round complexity is $(n^2/h^2) \cdot \mathsf{poly}\log\lambda$ under strongly adaptive corruptions and $\Omega(n^2/h^2)$ under weakly adaptive corruptions. It is an intriguing question whether we can match the elegant lower bound of Garay et al. [14].

- As mentioned earlier, it would be interesting to understand how general our current Distribute primitive is, and whether one can devise a general compiler that upgrades any weakly adaptive protocol to a strongly adaptive one while preserving its security.

- Finally, for communication complexity, our protocol for the dishonest majority case is not known to be optimal. It is an interesting question whether quadratic communication is necessary or linear complexity is possible under a dishonest majority. Another question is whether we can also achieve the same complexity under partial synchrony or asynchrony. Since nodes can accuse honest senders in an asynchronous network, we need a mechanism to refresh the trust information (e.g., the history of accusations) maintained, which is an interesting technical challenge.

References

[1] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer, “Farsite: Federated, available, and reliable storage for an incompletely trusted environment,” *ACM SIGOPS Operating Systems Review*, vol. 36, no. SI, pp. 1–14, 2002.

[2] M. Castro and B. Liskov, “Practical byzantine fault tolerance,” in *OSDI*, vol. 99, 1999, pp. 173–186.

[3] L. Lamport, “The part-time parliament,” *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998.

[4] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, “Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus,” *CoRR*, vol. abs/1612.02916, 2016.

[5] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, “Algorand: Scaling byzantine agreements for cryptocurrencies,” in *Proceedings of the 26th symposium on operating systems principles*, 2017, pp. 51–68.

[6] R. Farell, “An analysis of the cryptocurrency industry,” 2015.

[7] L. Lamport, R. Shostak, and M. Pease, “The byzantine generals problem,” *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.

[8] M. J. Fischer and N. A. Lynch, “A lower bound for the time to assure interactive consistency,” *Information Processing Letters*, vol. 14, no. 4, pp. 183–186, 1982, ISSN: 0020-0190. DOI: https://doi.org/10.1016/0020-0190(82)90033-3. URL: https://www.sciencedirect.com/science/article/pii/0020019082900333.

[9] D. Dolev and H. R. Strong, “Authenticated algorithms for byzantine agreement,” *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[10] I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren, “Synchronous byzantine agreement with optimal resilience, expected $o(n^2)$ communication, and expected $o(1)$ rounds,” in *Financial Cryptography and Data Security (FC)*, 2019.

[11] P. Feldman and S. Micali, “Byzantine agreement in constant expected time,” *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pp. 267–276, 1985.

[12] P. Feldman and S. Micali, “An optimal probabilistic protocol for synchronous byzantine agreement,” in *SIAM Journal of Computing*, 1997.

[13] J. Katz and C.-Y. Koo, “On expected constant-round protocols for byzantine agreement,” in *Annual International Cryptology Conference*, Springer, 2006, pp. 445–462.

[14] J. A. Garay, J. Katz, C.-Y. Koo, and R. Ostrovsky, “Round complexity of authenticated broadcast with a dishonest majority,” in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS’07)*, IEEE, 2007, pp. 658–668.

[15] M. Fitzi and J. B. Nielsen, “On the number of synchronous rounds sufficient for authenticated byzantine agreement,” in *International Symposium on Distributed Computing*, Springer, 2009, pp. 449–463.

[16] T.-H. H. Chan, R. Pass, and E. Shi, “Sublinear-round byzantine agreement under corrupt majority,” in *PKC*, 2020.

[17] D. Dolev and H. R. Strong, “Authenticated algorithms for byzantine agreement,” *Siam Journal on Computing - SIAMCOMP*, vol. 12, no. 4, pp. 656–666, 1983.

[18] J. Garay, J. Katz, C.-Y. Koo, and R. Ostrovsky, “Round complexity of authenticated broadcast with a dishonest majority,” in *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, Nov. 2007.

[19] M. Fitzi and J. B. Nielsen, “On the number of synchronous rounds sufficient for authenticated byzantine agreement,” in *Proceedings of the 23rd International Conference on Distributed Computing*, ser. DISC’09, Elche, Spain: Springer-Verlag, 2009, pp. 449–463, ISBN: 3-642-04354-2, 978-3-642-04354-3. URL: http://dl.acm.org/citation.cfm?id=1813164.1813222.

[20] J. Groth, R. Ostrovsky, and A. Sahai, “New techniques for noninteractive zero-knowledge,” *J. ACM*, vol. 59, no. 3, 11:1–11:35, Jun. 2012, ISSN: 0004-5411. DOI: 10.1145/2220357.2220358. URL: http://doi.acm.org/10.1145/2220357.2220358.

[21] G. Malavolta and S. A. K. Thyagarajan, “Homomorphic time-lock puzzles and applications,” Lecture Notes in Computer Science, vol. 11692, pp. 620–649, 2019.

[22] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, “Verifiable delay functions,” in *CRYPTO*, 2018.

[23] D. Dolev and R. Reischuk, “Bounds on information exchange for byzantine agreement,” *Journal of the ACM (JACM)*, vol. 32, no. 1, pp. 191–204, 1985.

[24] P. Berman, J. A. Garay, and K. J. Perry, “Bit optimal distributed consensus,” in *Computer science*, Springer, 1992, pp. 313–321.

[25] A. Momose and L. Ren, “Optimal communication complexity of authenticated byzantine agreement,” in *35th International Symposium on Distributed Computing (DISC 2021)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[26] I. Abraham, T. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, “Communication complexity of byzantine agreement, revisited,” in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 317–326.

[27] E. Blum, J. Katz, C.-D. Liu-Zhang, and J. Loss, “Asynchronous byzantine agreement with subquadratic communication,” in *Theory of Cryptography Conference*, Springer, 2020, pp. 353–380.

[28] S. Cohen, I. Keidar, and A. Spiegelman, “Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp,” *arXiv preprint arXiv:2002.06545*, 2020.

[29] V. King and J. Saia, "Breaking the o (n 2) bit barrier: Scalable byzantine agreement with an adaptive adversary," *Journal of the ACM (JACM)*, vol. 58, no. 4, pp. 1–24, 2011.

[30] K. Kursawe and V. Shoup, "Optimistic asynchronous atomic broadcast," in *International Colloquium on Automata, Languages, and Programming*, Springer, 2005, pp. 204–215.

[31] H. V. Ramasamy and C. Cachin, "Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast," in *International Conference On Principles Of Distributed Systems*, Springer, 2005, pp. 88–102.

[32] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[33] A. Spiegelman, "In search for an optimal authenticated byzantine agreement," in *35th International Symposium on Distributed Computing*, 2021.

[34] R. Gelashvili, L. Kokoris-Kogias, A. Sonnino, A. Spiegelman, and Z. Xiang, "Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback," *arXiv preprint arXiv:2106.10362*, 2021.

[35] Y. Lu, Z. Lu, and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as pipelined bft," *arXiv preprint arXiv:2103.09425*, 2021.

[36] C. Ganesh and A. Patra, "Broadcast extensions with optimal communication and round complexity," in *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, ACM, 2016, pp. 371–380.

[37] ——, "Optimal extension protocols for byzantine broadcast and agreement," *Distributed Computing*, pp. 1–19, 2020.

[38] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. 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.

[39] F. Cristian, H. Aghili, R. Strong, and D. Dolev, "Atomic broadcast: From simple message diffusion to byzantine agreement," *Information and Computation*, vol. 118, no. 1, pp. 158–179, 1995.

[40] M. Ben-Or, S. Goldwasser, and A. Widgerson, "Completeness theorems for noncryptographic fault-tolerant distributed computations," in *Proceedings of the 20th Annual Symposium on the Theory of Computing (STOC'88)*, 1988, pp. 1–10.

[41] T. P. Pedersen, "A threshold cryptosystem without a trusted party," in *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 1991, pp. 522–526.

[42] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, 2007.

[43] D. Dolev and H. R. Strong, "Authenticated algorithms for byzantine agreement," *SIAM Journal on Computing*, vol. 12, no. 4, pp. 656–666, 1983.

[44] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *International Workshop on Public Key Cryptography*, Springer, 2003, pp. 31–46.

[45] J. Groth, “Non-interactive distributed key generation and key resharing,” *Cryptology ePrint Archive*, 2021.

[46] N. Bitansky, S. Goldwasser, A. Jain, O. Paneth, V. Vaikuntanathan, and B. Waters, “Time-lock puzzles from randomized encodings,” in *ITCS*, 2016.

[47] R. L. Rivest, A. Shamir, and D. A. Wagner, “Time-lock puzzles and timed-release crypto,” USA, Tech. Rep., 1996.

[48] H. Lin, R. Pass, and P. Soni, “Two-round and non-interactive concurrent non-malleable commitments from time-lock puzzles,” in *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, 2017, pp. 576–587.

[49] D. Boneh and M. Naor, “Timed commitments,” pp. 236–254, 2000.

[50] S. Micali, S. Vadhan, and M. Rabin, “Verifiable random functions,” in *FOCS*, 1999.

[51] A. Bishop, V. Pastro, R. Rajaraman, and D. Wichs, “Essentially optimal robust secret sharing with maximal corruptions,” in *Advances in Cryptology – EUROCRYPT 2016*, M. Fischlin and J.-S. Coron, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 58–86.

[52] R. Gennaro, Y. Ishai, E. Kushilevitz, and T. Rabin, “On 2-round secure multiparty computation,” in *CRYPTO*, 2002.

[53] G. Bracha, “Asynchronous byzantine agreement protocols,” *Inf. Comput.*, vol. 75, no. 2, 1987.

[54] I. Abraham, T.-H. H. Chan, D. Dolev, K. Nayak, R. Pass, L. Ren, and E. Shi, “Communication complexity of byzantine agreement, revisited,” in *PODC*, 2019.