ABSTRACT

Title of Dissertation: IMPROVING ROUND AND COMMUNICATION METRICS IN CONSENSUS

Georgios Tsimos
Department of Electrical Engineering
Doctor of Philosophy, 2025

Dissertation Directed by: Professor Jonathan Katz
Department of Computer Science

Byzantine Consensus protocols, aka Byzantine Agreement (BA), Byzantine Broadcast (BB), allow a set of n mutually distrusting parties to share input values and agree on the same output value. Notable practical applications of consensus in blockchains and MPC require efficient, practical implementations of consensus protocols. The rise of blockchain systems and the general trend towards decentralized services, which inherently utilize consensus, as well as the increased demand of consensus as a primitive for implementing other cryptographic protocols (e.g. in MPC), brought once again this topic in the forefront of research.

A recurring trilemma of consensus when utilized in practice is striking the balance in order to construct protocols that are: i) *efficient*, ii) *resilient* under harsh adversarial conditions, and iii) with *minimal assumptions* for the corresponding setting. These properties are usually negatively associated; efficient protocols often either require stronger assumptions, or are less resilient to strong adversaries.

In this dissertation we aim to improve the efficiency of consensus protocols under harsh adversarial conditions and weak assumptions. We explore directions towards improving the two major metrics of efficiency of such protocols; i.e. their *communication* and *round* complexities. We focus on protocols operating in a synchronous communication network, and show how to achieve efficiency of either rounds or communication under different assumptions, against weakly adaptive adversaries with high corruption threshold. We will construct i) (Parallel) Broadcast protocols with improved state of the art communication, ii) the first Broadcast protocol with sublinear rounds without trusted setup, and iii) the first Deterministic Byzantine Agreement protocol with adaptive $O(n \cdot f)$ communication.

IMPROVING ROUND AND COMMUNICATION METRICS IN CONSENSUS

by

Georgios Tsimos

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2025

Advisory Committee:
Professor Jonathan Katz, Chair/Advisor
Professor Charalampos Papamanthou, Co-Chair/Advisor
Professor Julian Loss
Professor Dana Dachman-Soled
Professor Tudor Dumitraş
Professor Lawrence C. Washington, Dean's Representative

Chapter 3 © IACR, 2022. https://doi.org/10.1007/978-3-031-15982-4_15
Chapter 4 © SIAM, 2025. https://doi.org/10.1137/1.9781611978322.141
Chapter 5 © SIAM, 2024. https://doi.org/10.1137/1.9781611977912.43
The materials of Ch. 3 are a minor revision of the version published by Springer-Verlag. The copyright is held by IACR. The materials of Ch. 4, 5 are minor revisions of versions published by SIAM. The copyrights are held by SIAM. Tables used in Ch. 1 also fall under the copyrights. All materials are used in compliance with IACR and SIAM copyright policies respectively. All other materials:
© Copyright by
Georgios Tsimos
2025

Acknowledgments

Despite all the research efforts carried behind this text, this part is the most challenging for me. If I were to extend my sincere thanks to every single person that positively affected my efforts during these years, this would probably need to be the longest chapter of this thesis. As such, I first extend my thanks to every single friend, mentor, colleague and family member, who have supported me, each their own way, on this difficult, yet rewarding part of my life. You all know who you are, and that I owe you my gratitude!

First I'd like to thank my advisor, Charalampos Papamanthou for the amazing opportunity to study and do research at Maryland and Yale, and learn how to be a researcher alongside him. Babis is the special kind of person who is simultaneously smart, driven and meticulous in research, while radiating kindness and joy towards what we are doing. His mentorship and long hours of guidance give me confidence that I will be as precise as him in my future research.

I would also like to thank my co-advisor, Jonathan Katz. During my time in Maryland, Jon's course was the most interesting and challenging course in cryptography I've taken; the reading groups he led were also invaluable intros to so many interesting research topics. His intuition for problems that seem hard is truly enviable. Also, his uncanny ability for an academic to always respond fast to emails is always extrememly appreciated.

Julian Loss has been the definition of a mentor for me. He has been both a friend and a bright example of what a talented and motivated academic should be, always full of ideas and

with a mind racing ahead towards solutions. His firm stance and example as to what ideals a truth-seeking person must uphold are admirable. I will always cherish our long discussions at Maryland, and our long jogs during COVID, that helped me maintain my mental fortitude.

I extend my gratitude to the rest of my defense committee at UMD– Professors Tudor Dumitraş, Dana Dachman-Soled and Lawrence Washington– for taking some of their invaluable time to read and help me improve this thesis.

I was very fortunate to have many amazing mentors! Thank you Andreea, Lefteris, Chrysa, Alberto, Foteini, Shravan, Giannis and Leo– some for opportunities to learn beside you during internships and all of you for invaluable pieces of advice regarding research and how our field operates. I am also grateful to my collaborators for all their inspiring insights. To many more!

During these years I've made so many friends to whom I am indebted; Angelos, Chris, Leo, Panos and the rest of the Greek gang at Maryland, with whom we've had a lot of fun times worth of their own sit-com, and with whom we made the best out of a weird period of quarantines, anxiety and turmoil. My ECE cohort: Hunter, Lambros, Panayiotis, Zair and everyone else, for all the cool math we've discussed, both during coursework, and with drinks on our hands. My friends at Yale; Fatima, Arthur, Giannis, Weijie and more, with whom we've discussed so many things about research and life. I am especially thankful to Varun and Bhagya for their hospitality and all the fun times we spent playing board games and eating delicious Indian food!

Some people in my alma mater mean a lot to me. Professor Eugenie Foustoukou showed me the first steps towards uncompromising research– I hope I can pursue research guided by the same philosophical interest. Professor Stavros Toumpis, thank you for letting me know early what can be accomplished; if not for both of you my academic life would have been so different.

I owe my deepest thanks to my friends and family in Greece- I count you all as family. Alexi,

Niko, Iasona, Kosta, Alex, Niki, Melina, Maria and all the rest; we've grown up together and you've always been there for me. Lambro, Chloe, Thanasi, you still provide the most insightful discussions on so many topics. Thank you to all my close relatives for your support that was a ray of light whenever I've felt lonely and distraught.

My brother Niko, the unconditional love we've shared since the first time I laid eyes on you still guides me; you've always been the one person I know that will understand me and that feeling is invaluable. Foteini, I was lucky to meet true love during challenging times, thank you for everything left untold. My parents, you have been a guiding and loving force through my life; whatever I achieve can always be traced back to you, and as such, I will always strive to do and be better. Finally, to my grandparents, the two people who I cannot even hope to surpass: When you left the small village more than 60 years ago, you had far fewer tools in hand and far scarcer opportunities; yet you always strove for the betterment of your family. I hope my efforts will be a small testament as to the growth that a few generations of determined, capable and well-raised people can lead to.

I love you all and I am in your debt.

Table of Contents

List of Figures

Chapter 1: Introduction

1.1 A Brief History of Consensus

The seminal works by Lamport, Pease and Shostak [1, 2] introduced the area of Byzantine Consensus and formalized the two main problems in the area; *Byzantine Agreement* (BA) and *Byzantine Broadcast* (BB). In BA, there is a set of parties, some of which honest, and each party has some input value. Parties want to run the protocol and reach agreement on one output value, depending on all inputs. In BB, only one party, the designated sender, has some input. Parties want to reach agreement on the same output value, and especially if the sender is honest, parties must output the sender's value.

Consensus thus explores the question of how a set of parties (machines, actors etc.) can reliably communicate values, even in the presence of faults. The problem of reliably communicating some value arises naturally in several areas, such as Verifiable Secret Sharing (VSS) and Multiparty Computation (MPC) (e.g., [3, 4]). In recent years, research in the area of consensus has gained even more traction, due to its close relation with blockchains (see works related to BFT SMR, e.g. [5, 6]). As the practical applications requiring Byzantine consensus primitives increase, so does the importance of acquiring efficient solutions.

Defining the correct security model for a respective practical setting of consensus is an inherently complex task. In order to formally define a security model related to consensus,

several factors need to be accounted for; what are the adversarial capabilities (and the inherent limitations in terms of solutions as said capabilities increase), what are the network assumptions and what cryptographic tools are being allowed, just to name a few. The usefulness of consensus as a primitive, combined with its plentiful problem definitions, has led to an extensive corpus of scientific literature. We cover in more detail part of the literature in the related work section 1.3.

One way to distinguish different results in the area of consensus, is with respect to the *network synchrony assumptions* one is willing to make. Initial results in the area [1, 2, 7, 8] assumed a *synchronous network*, where parties have their internal clocks synchronized within up to a Δ deviation from a global clock. This allows parties to establish and execute protocols that occur within rounds dependent on time elapsed, rather than on occurring events. The latter is the trademark of *asynchronous networks*, where parties have no guarantees on when they will send/receive messages. Protocols on each setting are called *synchronous* or *asynchronous* respectively. The famous FLP impossibility result [9] showed how more potentially restricting the asynchronous model could be; however a large set of results since using cryptography and randomization have improved the feasibility of solutions (see related work 1.3). An interesting balance between the two is the model of partial synchrony, while there are even more models that are being explored up to this day.

However, in this text we focus our attention in the synchronous communication model. The tradeoff is that the assumption of synchrony is strong (for some practical applications it might be too strong), but at the same time it allows for positive results even in the presence of stronger adversaries (mostly in terms of corruption threshold). Research interest in this model has also remained high, since it resonates in settings where the adversary might control a substantial part of the network, as well as in settings with participants in close geographical proximity.

Before we start describing our contributions, we would like to provide some more clarification on what key points one considers when assessing a consensus protocol. For a detailed exposition of the several parameters used in defining system models, we also find the related discussion in [10] insightful.

The first key point is the specific problem that the protocol aims to tackle, be it either BA or BB. Any such protocol is executed among a set of parties, out of which some are honest and some are faulty. It is revealed that natural thresholds of *resilience* arise in any such protocol, in terms of the upper bound of faulty parties the protocol can tolerate– this threshold of say t faulty parties is called the protocol's resilience.

Resilience depends on the problem itself; for example BA naturally cannot be defined for $t \geq n/2$ — in that case, faulty parties, specifically malicious ones, can decide the majority value arbitrarily. On the other hand, BB can be defined even for any arbitrary $t < n$ [7]. The use (or lack) of cryptographic primitives as well as the adversarial behavior (crash-only vs. Byzantine/static vs. adaptive) and the determinism of the protocol (deterministic vs. randomized) all play a role with regards to a protocol's resilience [2, 7, 11–13].

The efficiency metrics of such a protocol are mainly i) its communication complexity, i.e. the total number of bits exchanged between honest parties during the protocol's execution and ii) its round complexity, i.e. the number of rounds it takes for the protocol to terminate. Also, some solutions are based on inherently stronger assumptions than others; this is yet another way to compare different protocols. After this brief discussion on the basics of consensus, we are ready to present our results.

1.2 Our scope: Efficiency with high resilience using fewer assumptions

Our published results so far study the feasibility of protocols with improved communication or round complexity in their setting, compared to prior works. We answer positively on whether it is feasible to have protocols with improved communication or round complexity in three different settings, two of which are regarding BB and one regarding BA. We study both deterministic and randomized solutions; our set of protocols regarding BB are all randomized, while our BA protocol is deterministic. We also strive to provide security against adversaries with strong capabilities.

1.2.1 Gossiping for Communication-Efficient Broadcast.

We first focus on the setting of synchronous Byzantine Broadcast (BB). The seminal work of Dolev and Strong [7] has been the point of reference for decades, since the protocol is rather elegant, deterministic, requires minimal setup assumptions (only a PKI) and tolerates any corruption threshold $t < n$. However, its communication complexity is $O(n^2 \cdot t \cdot \kappa)$, and its round complexity is $O(t)$, where κ refers to the computational security parameter and t can be $O(n)$; such communication is near-prohibitive from a practical viewpoint of a single party broadcasting a single bit, and the rounds could scale badly with n. Recent lines of work [14–16] achieved improved communication but are not with their own limitations; they either work under a much more restrictive threshold of honest majority [14, 15], or they require additional trust assumptions [14, 16], namely the PKI to be setup by a trusted third party, so that it satisfies some specific structure.

In our first contribution, we attempt to combine the best of all these protocols above, i.e. achieve a Broadcast protocol with $O(n^2)$ communication, while maintaining the corruption

threshold to almost optimal (i.e. $t < a \cdot n$, where a is a constant fraction), without incurring additional trust assumptions. In other words, we aim to either match the best communication complexity of protocols for that threshold while lowering trust assumptions, or keep the trust assumptions and find a way to improve communication further. As we will see, we will achieve both directions.

We utilize a randomized approach for message propagation (i.e. gossiping), which ensures that a message will reach all parties after a few rounds while we also keep communication costs low. This results to our BULLETINBB which compares favorably in several terms to each of the previous works. However, the protocol is secure against only static adversaries.

This initial attempt guides us as to how to attack a closely related, and equally interesting problem from a practical perspective; that of *interactive consistency* [1], or *Parallel Broadcast (PBB)*. In PBB, all parties start with some value and all parties want to broadcast their value. Clearly, existing broadcast protocols solve PBB trivially; each party simply broadcasts its own value separately. This however leads to PBB protocols with communication $n \cdot CC$, where CC is the communication of the broadcast protocol used. Instead, we combine a more sophisticated gossiping analysis with a technique to efficiently hide the communication patterns of honest parties. This approach yields two novel PBB protocols (TRUSTEDPBB and BULLETINPBB), one without trust assumption and one assuming trusted PKI, each being the best known PBB protocols up to date for their settings respectively (i.e. taking into account protocols with similar or better trust assumptions and resilience), in terms of communication complexity. We provide a comparison between our work and existing work in Table 1.1.

Notice that these results are mostly of theoretical interest, mainly due to the high factors in the security parameter and the fact that they are for single bit messages. Concurrently with the

Protocol	PKI	CC	RC	Adversary	Corruptions t	Problem
Abraham *et al.* [14]	trusted	$\tilde{O}(n \cdot \kappa)$	$O(1)$	adaptive	$< n/2$	BB
Momose, Ren [15]	bulletin	$\tilde{O}(n^2 \cdot \kappa)$	$O(n)$	adaptive	$< n/2$	BB
Chan, Pass, Shi [16]	trusted	$O(n^2 \cdot \kappa^2)$	$O(\kappa)$	adaptive	$< (1-\epsilon) \cdot n$	BB
Dolev, Strong [7]	bulletin	$O(n^3 \cdot \kappa)$	$O(n)$	adaptive	$< n$	BB
BULLETINBB §3.1	bulletin	$O(n^2 \cdot \kappa^2)$	$O(n)$	static	$< (1-\epsilon)n$	BB
BULLETINPBB §3.3	bulletin	$\tilde{O}(n^3 \cdot \kappa^2)$	$O(n \log n)$	adaptive	$< (1-\epsilon)n$	PBB
TRUSTEDPBB §3.4	trusted	$\tilde{O}(n^2 \cdot \kappa^4)$	$O(\kappa \log n)$	adaptive	$< (1-\epsilon)n$	PBB

Table 1.1: Comparison of our protocols from Chapter 3 in terms of communication complexity in bits (CC) and round complexity (RC), to existing work. Number of parties is n. Also $\epsilon < 1$.

writing of this thesis, we published a new result [17] that, among others, addresses these issues for the the Trusted-PKI PBB protocol, lowering the security parameter factors and efficiently extending it to the multi-bit setting.

1.2.2 Sublinear Round Broadcast without Trusted Setup.

Staying in the same problem direction, we also aim to improve the round complexity of synchronous Broadcast; specifically we are asking the question of whether we can achieve synchronous Broadcast running in sublinear rounds in n, against a dishonest majority, without trusted setup.

Towards this approach, there is a clear starting point; Chan *et al.* [16] utilize small committee election to reduce the round complexity to $O(\kappa)$. However, to achieve that they utilize VRFs and require trusted setup for the crs and the keys' generation. This is what we aim to avoid.

For that, we ask each party to choose their own randomness and provide their own key for committee election. Of course, against malicious parties this does not work straightforward. What we do is construct a detailed committee election process, that utilizes randomness shared by parties via Time-Lock Puzzles in a timely manner, as well as weaker than Broadcast primitives

to guarantee that honest parties will eventually have the same view for the elected committee.

After proving the correctness of our committee election, we run modified version of the Broadcast protocol by Chan *et al.* [16], where committee membership is checked and proven via our schemes.

In Table 1.2 we provide a comparison between works for synchronous broadcast in the dishonest majority case. Here, n is the number of parties, ϵ is the honest *constant* fraction, and κ is the security parameter. Also, "*" denotes the strong adaptive adversarial model, while the others are in the weak adaptive adversarial model.

Protocol	Corruptions t	Assumptions	Rounds
Dolev & Strong [7]	$< n^*$	bulletin-PKI	$O(n)$
Chan *et al.* [16]	$< (1-\epsilon)n$	trusted-PKI	$O(\kappa)$
Wan *et al.* [18]	$< (1-\epsilon)n^*$	trusted-PKI	$O(\kappa)$
Wan *et al.* [19]	$< (1-\epsilon)n$	trusted-PKI	$O(\kappa)$
BULLETINPBB §3.3	$< (1-\epsilon)n$	bulletin-PKI	$O(n \log n)$
TRUSTEDPBB §3.4	$< (1-\epsilon)n$	trusted-PKI	$O(\kappa \log n)$
Momose *et al.* [20]	$< n^*$	bulletin-PKI	$O(n)$
Π_{BC} § 4.3	$< (1-\epsilon)n$	bulletin-PKI, NIZKs,TLPs	$O(\kappa)$

Table 1.2: Comparison of our sublinear round BB protocol from Chapter 4 to other BB protocols in the synchronous, dishonest majority setting.

1.2.3 Deterministic BA with Adaptive $O(n \cdot f)$ Communication.

We finally turn to synchronous Byzantine Agreement (BA). Even though this setting has been studied for decades, an exciting, fairly recent work [15] established the best deterministic BA protocol in terms of communication prior to our result, achieving $O(n^2)$ communication, for any number of corruptions $t < n/2$ (BA is restricted to honest majority).

Since then, another work by Cohen, Keidar and Spiegelman [21] explored the notion of

adaptive communication complexity for BA and BB. By adaptive communication complexity, we refer to protocols with proven communication depending on the *actual* number of corrupted parties during execution, which we denote by f, to distinguish from the corruption threshold t. Since $f \leq t$ and f can be arbitrarily small, in executions where the adversary is "lazy", this notion allows for novel interesting constructions that attempt to at least substitute t with f.

Cohen *et al.* achieve deterministic, synchronous BB with adaptive $O(n \cdot f)$ communication. To do so, they first solve a weaker notion of BA, called weak BA (WBA), which substitutes validity with a weaker definition depending on predicates. They then extend their WBA protocol to a full BB protocol. They also attempt to use their ideas to achieve BA with the same communication and, even though not successful, they provide interesting incentive through their attempted techniques.

Motivated by their ideas, we construct the *first* deterministic, synchronous BA protocol with adaptive $O(n \cdot f)$ communication. In order to do so, we define and solve two subproblems, namely *Reliable Voting* and a modified definition of WBA, and combine them to achieve BA. Our construction operates using the idea of *silent phases* from [21]; the protocol is executed in n round-robin phases, where in every phase r, party P_r acts as leader. Parties communicate either solely with the leader, or with the rest of parties via an efficient, backdoor communication method defined by an expander graph. In doing so they maintain low communication complexity of $O(n)$ overall in each phase of the protocol. By guaranteeing that the protocol requires asymptotically at most as many phases as the active dishonest parties, we reach the communication of $O(n \cdot f)$. To achieve this guarantee, leaders are forced to generate certificates in order to convince parties to commit on an output value. The structure of these certificates is non-trivial and in our solution requires use of threshold signatures, cryptographic hash functions and succinct non-interactive proof systems. We compare our result to existing protocols in Table 1.3.

Protocol	Corruptions t	Communication	Problem
Dolev and Strong [7]	$< n$	$O(n^2 \cdot t)$	BB
Momose and Ren [15]	$< n/2$	$O(n^2)$	BA
Cohen, Keidar, Spiegelman [21]	$< n/2$	$O(n \cdot f)$	BB
Π_{BA} §5.1	$< (1/2 - \epsilon)n$	$O(n \cdot f)$	BA

Table 1.3: Comparison of existing literature with our work. We measure communication of all protocols in *words*, where a word comprises a constant number of signatures and values.

1.3 Related Work

Byzantine Agreement and Byzantine Broadcast were introduced in two celebrated papers by Lamport, Shostak, and Pease [1, 2]. Dolev and Strong [7] proposed the first BB protocol with polynomial communication complexity of $O(n^2 \cdot t)$ (amount of messages exchanged). Their result was proven by Dolev and Reishuk [8] to be asymptotically optimal. However the Dolev-Strong protocol came at the cost of cubic communication complexity due to message length of $O(t)$ signatures. Several lines of work have aimed to improve the communication efficiency, since for several different settings [14, 16, 22–28]. Momose and Ren proposed a deterministic protocol with optimal $O(n^2)$ complexity [15].

Somewhat surprisingly, in the setting with setup (for $t < n$), any efficiency improvement to the early work of Dolev and Strong has been aimed exclusively at improving the *round complexity* rather than the communication complexity. This has been the subject of several works [16, 19, 29, 30]. Round complexity has been improved in general via several lines of work [13, 31–34]. Wan *et al.* [18] tolerates stronger adversaries that can also erase messages, still with a sublinear number of rounds.

The problem of interactive consistency or parallel broadcast was originally introduced by

Pease et al. [2]. Recent results have rekindled the interest towards parallel broadcast [35–37]. Gossip (or flooding) protocols have been used in various settings related to consensus [38–43]. Cohen *et al.* [44] studied the feasibility of property-based/simulation-based adaptive broadcast.

Several lines of work [6, 14, 23, 28, 45–47] have also considered improving the communication complexity of byzantine agreement, via various techniques such as randomization, optimistic execution and early-stopping [48–50]. More recently, results regarding the adaptive communication complexity of consensus appeared [21, 45]; specifically, Cohen *et al.* [21] achieve the first deterministic broadcast protocol with adaptive communication complexity of $O(n \cdot f)$.

The round complexity of byzantine agreement has also been extensively studied [12, 13, 15, 34, 51–53]. So has the line of work related to graded broadcast (also graded consensus, graded verifiable secret sharing) [12, 30, 34, 54, 55]. Also, time-lock puzzles (TLP) [56–62] and similar constructions have allowed for novel results related to consensus [63–65].

Results related to part of our work, but using different techniques, include Das *et al.* [65], who describe a Byzantine agreement protocol using VDFs in the ROM. Their construction achieves an expected constant round complexity without trusted setup, tolerating $t < n/2$ adaptive corruptions. Also, Srinivasan *et al.* [63] provide a compiler based on TLPs, that achieves expected constant-round Broadcast against even strongly adaptive adversaries for the dishonest majority setting; however their construction assumes trusted setup.

The topic of consensus for long-input messages has also been studied extensively [2, 17, 66–74]. Practical solutions of consensus have been the center of research in the area of blockchain [75–86] in the more recent years, focusing mostly in the asynchronous and partially synchronous model [74, 87–90]. For an even more fine-grained analysis of the related work we refer the reader to the respective related work sections of our published papers and preprints.

1.4 Thesis Outline

In Chapter 2 we discuss related work, preliminary information necessary for the rest of the text, as well as definitions and primitives.

In Chapter 3 we present the gossiping techniques that allow us to construct communication efficient (Parallel) Broadcast protocols.

Chapter 4 presents the first sublinear round Broadcast protocol against dishonest majority, that does not assume trusted setup.

In Chapter 5 we construct the first Deterministic Byzantine Agreement protocol with adaptive communication complexity that is $O(n \cdot f)$.

This thesis includes material that are minor revisions of works already published in research conferences. The construction in Chapter 3 appears on a paper co-authored with Julian Loss and Charalampos Papamanthou and published in the proceedings of Advances in Cryptology, CRYPTO 2022 [35]. Chapter 4 is based on a paper co-authored with Andreea Alexandru, Julian Loss Charalampos Papamanthou and Benedikt Wagner and published in the proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025 [91]. Chapter 5 is based on a paper co-authored with Fatima Elsheimy and Charalampos Papamanthou and published in the proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2024 [92].

For completeness, any proof ommitted by the main body to assist with the overall exposition, can be found in the Appendix.

Chapter 2: Preliminaries

In this section, we provide definitions and notational choices that will be used throughout the text. We first describe our network setting, then provide definitions of the several consensus protocols that will be discussed later and finally we provide formal definitions of the cryptographic primitives that will be used.

2.1 System Settings

2.1.1 Communication Model

We consider the standard synchronous model of communication. In this model, n parties $P_1, \ldots, P_n$[1] are assumed to share a global clock that progresses at the same rate for all parties. Furthermore, we consider that parties are connected via pairwise, authenticated channels (in short, messages from honest parties can be overheard but not tampered by malicious actors). Any message that is sent by an honest party at time T is guaranteed to arrive at every honest party at time $T + \Delta$, where Δ is the finite, known, network delay. Therefore, messages of honest parties can not be dropped from the network and are always delivered within Δ. As such, we consider protocols that execute in a round-based fashion, where every round in the protocol is of length Δ

[1] Throughout the text the notation for parties might be interchangeably either $P_i, i \in [n]$, or simply by using the party's index $i \in [n]$.

and parties start executing the r-th round of a protocol at time $(r-1)\cdot\Delta$. Round complexity is defined naturally as the number of rounds it takes for all honest party in the protocols to terminate.

Let $\mathcal{M}$ be a set of messages and $\mathcal{P}$ be a set of parties. When a party i calls $\textsc{Send}(\mathcal{M},\mathcal{P})$ at round r, then the set of messages $\mathcal{M}$ is delivered to parties in $\mathcal{P}$ during at most round $r+1$. When a party i calls $\textsc{Receive}()$ in round r, then all messages that were sent to i in round $r-1$ via $\textsc{Send}$ commands are stored in i's local storage. In Chapter 3, we additionally assume secure erasures, i.e. an honest party p can safely erase their state so that the adversary cannot access it whenever the adversary corrupts p (after the state has been erased), e.g. [23, 28].

2.1.2 Adversary Model

We consider a *Byzantine fault* model, i.e. a probabilistic polynomial-time (PPT) adversary who can corrupt up to t parties in a malicious fashion. This means that the adversary controls the messages and state of any corrupted party, and also can coordinate the actions of all byzantine parties. Parties that are not corrupted during the protocol execution, are considered *honest*. We denote the actual (runtime), unknown number of corruptions by the adversary by $f \in [0,t]$. The adversary can make parties deviate from the protocol description arbitrarily. Our adversary is also rushing, being able to observe the honest parties' messages in any synchronous round r of a protocol, and delay them until the end of that round. In this way, it can choose its own messages for that round before delivering any of the honest messages.

In Section 3.1, we will consider a *static adversary*, who must choose which parties to corrupt before the protocol execution starts. For the rest of the chapters, we consider an *adaptive adversary*, able to corrupt up to t parties, each at any point during the execution of the protocol,

learning its internal state which consists of any secret keys and ephemeral values that have not been deleted at that point. However, we do not consider *strongly* adaptive adversaries that could, after corrupting a party, observe what message that party attempted to send during that same round and then replace its message with another one.

We note here not to confuse the terms *adaptive adversary* and *adaptive communication*. With the latter, we refer to protocols with communication proven to depend on the *actual* number of corrupted parties during execution, denoted by f, to distinguish from the corruption threshold t. Since $f \leq t$ and f can be arbitrarily small, in executions where the adversary corrupts only a handful of parties, this notion allows for novel interesting constructions that substitute t with f.

We also assume the adversary cannot perform *after the fact removals*, i.e., once a message is sent by an honest party, the adversary cannot prevent the eventual delivery of that same message to honest recipients, unless it corrupts them. Moreover, we assume *atomic sends* [28]: an honest party p can send to multiple parties simultaneously, without the adversary being able to corrupt p in between sending to two parties.

Security Parameters. Throughout the text we denote with κ the computational security parameter. We let λ be a statistical security parameter, which we assume to be smaller than κ. Naturally, the number of parties n is polynomial, i.e., it satisfies $n \leq \mathsf{poly}(\kappa)$ and $n \leq \mathsf{poly}(\lambda)$. This also means that we have $n \cdot \mathsf{negl}(\lambda) = \mathsf{negl}(\lambda)$ and $n \cdot \mathsf{negl}(\kappa) = \mathsf{negl}(\kappa)$. Similarly, we assume $n \geq \omega(\log \lambda)$.

2.1.3 Notation

With $x \leftarrow\!\!\$\, X$ we denote that x is chosen uniformly at random from some finite set X. Also, with $y \leftarrow \mathcal{A}(x)$ we denote value y being output by a probabilistic algorithm $\mathcal{A}$ that runs on input x with uniformly chosen coins. If we write $y := \mathcal{A}(x; \rho)$, then we consider coins that are explicitly input by the adversary, allowing us to capture a deterministic algorithm. Also, $\mathbf{T}(\mathcal{A})$ denotes the runtime of algorithm $\mathcal{A}$. Throughout the text, we use negl to denote a negligible function, which is a function that goes to zero faster than every inverse polynomial. Similarly, we use poly and $\mathsf{polylog}$ to denote functions that are polynomial or polylogarithmic. Throughout the text, $\log$ denotes the natural logarithm.

Bulletin PKI vs. Trusted PKI. We briefly recall the difference between Bulletin PKI and Trusted PKI here. In sections 3.1, 3.3 and 4, we assume a bulletin PKI; that is, each party i has a secret key sk_i and a public key pk_i, where pk_i is known to all parties. The secret key sk_i and the public key pk_i are not assumed to be computed in a trusted manner; instead, we assume only that each party i generates its keys $(\mathsf{sk}_i, \mathsf{pk}_i)$ locally and then makes pk_i known to all other parties by using the public bulletin board. In section 3.4 and chapter 5, we assume a trusted PKI, where a trusted third party computes and distributes secret/public key pairs to each party, enabling the use of more powerful cryptographic primitives in our constructions, such as verifiable random functions 3.4 and threshold signatures 5. For a definition of a signature scheme that supports threshold signatures, we refer the reader to [93, 94]. Since our constructions in chapters 4, 5 make use of NIZKs, we further assume there that all parties have access to a (uniform) common reference string.

We use the notation $\mathsf{Sig}_i(m)$ to denote a signature of party P_i using sk_i on message m

and $\mathsf{SigVer}_i(s, m)$ to denote the verification of signature s on message m using public key pk_i. Signatures should achieve *correctness*: for any message m, it holds that $\mathsf{SigVer}_i(\mathsf{Sig}_i(m), m) = 1$, and *unforgeability under chosen-message attack*: for a pair of honestly generated keys $(\mathsf{sk}_i, \mathsf{pk}_i)$, a party that does not have access to sk_i, cannot generate a signature s for a new message m such that $\mathsf{SigVer}_i(s, m) = 1$. As standard in the consensus literature, we assume idealized signatures that satisfy correctness and unforgeability perfectly.

Signatures. Party i computes a *signature* σ on a message m via $\sigma \leftarrow \mathsf{sig}(sk_i, m) =: \mathsf{sig}_i(m)$. Later σ can be verified via $ver(pk_i, \sigma, m)$. As is standard for this line of work, we assume signatures are *idealized* in the sense that it is impossible, without sk_i, to create a signature σ on a message m such that $ver(pk_i, \sigma, m) = 1$. We also assume *perfect correctness*, i.e., for any m, $ver(pk_i, \mathsf{sig}(sk_i, m), m) = 1$. We write $\mathsf{sig}_i(m)$ to indicate $\mathsf{sig}(sk_i, m)$ and $ver_i(\sigma, m)$ to indicate $ver(pk_i, \sigma, m)$.

Throughout the text, we might use the abbreviated notation $\langle m \rangle_{\sigma_i}$ to refer to triples $(m, \mathsf{sig}_i(m), i)$, where m is some message and $\mathsf{sig}_i(m)$ is the valid signature on that message for party p_i with public key pk_i. In this notation, any malformed messages, or invalid signatures are implied to simply not be considered by honest parties.

We also recall the discrete logarithm assumption.

Definition 1 (Discrete Logarithm Assumption)**.** *Let* GGen *be an algorithm that outputs the description of a cyclic group* $\mathbb{G}$ *with generator* g *of prime order* p*. We say that the discrete logarithm assumption holds relative to* GGen *if for any PPT algorithm* $\mathcal{A}$ *the following probability is negligible:*

$$\Pr\left[\mathcal{A}(\mathbb{G}, g, p, g^x) = x \mid (\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\kappa),\ x \leftarrow_s \mathbb{Z}_p\right].$$

2.2 Protocol Definitions

In all the following protocol definitions, we implicitly assume the definitions are for protocols tolerating up to t byzantine parties, i.e., the conditions hold if there are at most t corrupted parties. Also in all following protocols we omit to write, but imply, the *Termination* property, which states that all honest parties must terminate.

Definition 2 (Byzantine Broadcast)**.** *Let Π be a protocol executed by n parties. Let a designated party P_s (the sender) with initial value $\mathsf{v} \in \{0,1\}$. Let each honest party P_i upon termination output value $\mathsf{y}_i \in \{0,1\}$. A protocol Π achieves* Byzantine Broadcast (BB)*, if it achieves the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- ***Soundness:*** *If P_s is honest, then each honest party P_i outputs $\mathsf{y}_i = \mathsf{v}$;*
- ***Consistency:*** *All honest parties output the same value;*

Parallel Broadcast (PBB) — introduced in [2] as interactive consistency— is the natural extension of BB in the case where there are more than one designated senders.

Definition 3 (Parallel Byzantine Broadcast)**.** *Let Π be a protocol executed by n parties, where each honest party P_i starts with initial value $\mathsf{v}_i \in \{0,1\}$, and upon termination outputs an n-value vector $\mathbf{Y}_i \in \{0,1\}^n$. A protocol Π achieves* Parallel Byzantine Broadcast (PBB)*, if it achieves the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- ***Parallel Soundness:*** *If a party P_i is honest, then each honest party P_j outputs $\mathbf{Y}_j(i) = \mathsf{v}_i$;*
- ***Parallel Consistency:*** *All honest parties output the same vector;*

Gradecast was introduced by Feldman and Micali in [13] and later generalized for an arbitrary grade by Garay *et al.* [30]. It is a weaker form of broadcast, where honest parties might

not reach "full" agreement (consistency).

Definition 4 (Gradecast)**.** *Let* Π *be a protocol executed by* n *parties. Let a designated party* P_s *(the sender) with initial value* $\mathsf{v} \in \{0,1\}$*. Let each honest party* P_i *upon termination output* $(\mathsf{y}_i, g_i) \in \{0,1\} \times \{0,\ldots,g^*\}$*. A protocol* Π *achieves* g^*-gradecast *(GC), if it achieves the following properties, except with negligible probability* $\mathsf{negl}(\lambda)$*:*

- ***Graded Soundness:*** *If* P_s *is honest, then each honest party* P_i *outputs* (v, g^*)*;*
- ***Graded Consistency:*** *Let honest parties* P_i, P_j *outputting* (y_i, g_i) *and* (y_j, g_j)*, respectively. If* $g_i \geq 2$*, then* $\mathsf{y}_i = \mathsf{y}_j$ *and* $|g_i - g_j| \leq 1$*. If* $g_i = 1$*, then either* $\mathsf{y}_i = \mathsf{y}_j$ *or* $g_j = 0$*.*

We also define a useful extension of gradecast, namely *Moderated Gradecast* (MGC) where a moderator P_M re-gradecasts the value it received from the sender in gradecast P_s. The goal is for the honest parties to use the two pieces of information coming from the two gradecasts with different senders, to obtain "similar" outputs and to grade the moderator.

Definition 5 (Moderated Gradecast)**.** *Let* Π *be a protocol executed by* n *parties. Let a designated party* P_s *(the sender) with initial value* $\mathsf{v} \in \{0,1\}$*, and a* moderator P_M *who moderates for the sender* P_s*. Let each honest party* P_i *upon termination output* $(\mathsf{y}_i, g_i) \in \{0,1\} \times \{0,\ldots,g^*\}$*. A protocol* Π *achieves* g^*-moderated gradecast *(MGC), if it achieves the following properties, except with negligible probability* $\mathsf{negl}(\lambda)$*:*

- ***Graded Soundness:*** *If* P_s *is honest and* P_M *is honest, then each honest party* P_i *outputs* (v, g^*)*;*
- ***M-Soundness:*** *If moderator* P_M *is honest, then each honest party* P_i *outputs* (y, g_i) *for some* y *and for* $g_i \in \{g^* - 1, g^*\}$*;*
- ***Graded Consistency:*** *Let honest parties* P_i, P_j *outputting* (y_i, g_i) *and* (y_j, g_j)*, respectively.*

If $g_i \geq 2$*, then* $\mathsf{y}_i = \mathsf{y}_j$ *and* $|g_i - g_j| \leq 1$*. If* $g_i = 1$*, then either* $\mathsf{y}_i = \mathsf{y}_j$ *or* $g_j = 0$*.*

We now give related definitions with respect to Byzantine Agreement problems. Since we will later provide deterministic protocols for BA, we give the deterministic variants of the respective definitions.

Definition 6 (Byzantine Agreement)**.** *Let* Π *be a protocol executed by* n *parties, where each honest party* P_i *starts with initial value* $\mathsf{v}_i \in \{0,1\}$*, and upon termination outputs value* $\mathsf{y}_i \in \{0,1\}$*. A protocol* Π *achieves* Byzantine Agreement (BA)*, if it achieves the following properties:*

- ***Validity:*** *If all honest parties* P_i *have* $\mathsf{v}_i = \mathsf{v}$*, then each honest party* P_i *outputs* $\mathsf{y}_i = \mathsf{v}$*;*
- ***Consistency:*** *All honest parties output the same value;*

The next two definitions of the Reliable Voting and Weak Byzantine Agreement problems are defined in our work in [92]. Our WBA definition is a slight modification of the definition introduced in [21]. Both definitions are naturally used to separate the problem of Byzantine Agreement into smaller subproblems, that are easier to analyze and solve.

Definition 7 (Reliable Voting)**.** *Let* Π *be a protocol executed by* n *parties, where each honest party* P_i *starts with initial value* $\mathsf{v}_i \in \{0,1\}$*, and upon termination outputs value* $\mathsf{y}_i \in \{0,1\}$ *and auxiliary information* $\mathsf{aux}_i \in \{0,1\}^*$*. Let* $\mathsf{validate}(\cdot,\cdot)$ *denote a Boolean predicate on a party's output. A protocol* Π *achieves* Reliable Voting *(RV) with respect to* validate*, if it achieves the following property:*

- ***Provable Validity****: If all honest parties have initial value* $\mathsf{v}_i = \mathsf{v}$*, then (1) all honest parties output* $\mathsf{y}_i = \mathsf{v}$ *and* aux_i *such that* $\mathsf{validate}(\mathsf{y}_i, \mathsf{aux}_i) = \mathsf{true}$ *and (2) no party outputs* $\mathsf{y}_j \neq \mathsf{v}$ *and auxiliary information* aux_j *such that* $\mathsf{validate}(\mathsf{y}_j, \mathsf{aux}_j) = \mathsf{true}$*;*

Definition 8 (Weak Byzantine Agreement)**.** *Let* Π *be a protocol executed by* n *parties, where each honest party* P_i *starts with input* $(\mathsf{v}_i, \mathsf{aux}_i) \in \{0,1\} \times \{0,1\}^*$*, and outputs value* $\mathsf{y}_i \in \{0,1\}$*. Let* $\mathsf{validate}(\cdot,\cdot)$ *denote a Boolean predicate on a party's input. A protocol* Π *achieves Weak Byzantine Agreement with respect to* validate*, with up to* t *malicious parties, if it achieves the following properties:*

- ***Restricted Validity:*** *If all honest parties have input* $(\mathsf{v}, \mathsf{aux}_i)$ *such that* $\mathsf{validate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$*, and no party inputs* $(\mathsf{v}' \neq \mathsf{v}, \mathsf{aux}_i)$ *such that* $\mathsf{validate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$*, then all honest parties output* $\mathsf{y}_i = \mathsf{v}$*;*
- ***Consistency:*** *All honest parties output the same value;*

2.3 Cryptographic Primitives

2.3.1 Threshold Signatures.

We assume a signature scheme that supports threshold signatures (e.g. [93]). We will require such a signature scheme for Chapter 5.

Definition 9 (Threshold Signatures)**.** *An* $(n-t)$*-out-of-*n *threshold signature scheme* SS *is a tuple of efficient algorithms* (KeyGen, SignShare, VerShare, Combine, Verify) *defined as follows.*

- $\mathsf{KeyGen}(\kappa, n, n-t)$*: The randomized key generation algorithm takes as input the security parameter* κ *and the number of parties* n *in the protocol, it outputs a public and secret key share* $(\mathsf{sk}_i, \mathsf{pk}_i)$ *for each party, and a public key* pk_{ss}*;*
- $\mathsf{SignShare}(x, \mathsf{sk}_i)$*: On input a message* x *and a secret key share* sk_i*, it outputs a signature share* σ_i*;*
- $\mathsf{VerShare}(\mathsf{pk}_i, x, \sigma_i)$*: On input a public key share* pk_i*, a message* x*, and a signature share*

σ_i, *it outputs 1 if the signature is correct or 0 otherwise;*

- $\mathsf{Combine}((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), x, (\sigma_1, \ldots, \sigma_{n-t}))$*: On input a vector* $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ *of public key shares, a message* x*, and a set S of* $n - t$ *signature shares* (σ_i, i) *(with corresponding indices), it outputs a signature* σ *or* $\perp$*;*
- $\mathsf{Verify}(\mathsf{pk}_{ss}, x, \sigma)$*: On input a public key* pk_{ss}*, a message* x*, and a signature* σ*. It outputs 1 if the signature is correct or 0 otherwise.*

The signature scheme should satisfy the standard notions of Correctness, Unforgeability and Robustness against a PPT adversary.

2.3.2 Collision-Resistant Hash Functions.

We use a cryptographic collision-resistant hash function Hash, which maps an arbitrary length message to a fixed length output. For more details on cryptographic hash functions and their properties, we refer the reader to [95].

Definition 10 (Collision-Resistant Hash Functions)**.** *We say a family of hash functions* $\mathcal{H}$*, is collision-resistant if for a random* $\mathsf{Hash} \in_R \mathcal{H}$ *it is hard for a polynomially bounded adversary to come up with two preimages* $x \neq y$ *that collide* $(\mathsf{Hash}(x) = \mathsf{Hash}(y))$*.*

We utilize the function Hash to compute the hash value of a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_r$. To achieve this, sort the list in ascending order and then hash the concatenation of the sorted list.

2.3.3 Expander Graphs

Our protocols in Section 5.2 and Section 5.3 use expanders. Expanders are sparse graphs with $O(1)$ degree and with good connectivity. We use expanders as defined in [15], where

$\alpha = 2\epsilon, \beta = 1 - \alpha$, for constant $\epsilon \in (0, 1/2)$. In [15] they prove that such expanders exist with non-negligible probability, and can be obtained with overwhelming probability via a probabilistic algorithm within at most polynomially many attempts. We follow their notation and define a $(n, 2\epsilon, 1 - 2\epsilon)$ expander graph as $\mathcal{G}_{n,\epsilon}$.

Definition 11 (Expanders)**.** *Let α and β be constants satisfying $0 < \alpha < \beta < 1$. We define an (n, α, β)-expander to be a graph of n vertices such that, for any set S of αn vertices, the number of neighbors of S is greater than βn.*

2.3.4 Non-Interactive Proof Systems

In Sections 4, 5 we utilize NIZKs, so we now review their definitions.

An **NP** relation is a relation $\mathcal{R}$ containing pairs (x, w) of statements and witnesses that satisfies the following properties: (1) there is a polynomial q with $|w| \leq q(|x|)$ for all $(x, w) \in \mathcal{R}$, and (2) the relation is decidable in deterministic polynomial time. Moreover, $\mathcal{R}$ can be implicitly parameterized by the security parameter s.t. $|x| \leq \mathsf{poly}(\kappa)$ for all $(x, w) \in \mathcal{R}$. We now provide the definition of non-interactive proof systems.

Definition 12 (Non-Interactive Proof System)**.** *Let $\mathcal{R}$ be an* **NP** *relation. A non-interactive proof system for $\mathcal{R}$ is a triple* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *of PPT algorithms s.t.:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{crs}$*; on input the security parameter (in unary), outputs a common reference string* crs.
- $\mathsf{Prove}(\mathsf{crs}, x, w) \to \pi$*; on input a common reference string* crs*, a statement x and a witness w, outputs a proof π.*
- $\mathsf{Ver}(\mathsf{crs}, x, \pi) \to b$*; deterministic, on input a common reference string* crs*, a statement x*

and a proof π*, outputs a bit* $b \in \{0, 1\}$.

Further, we require the following completeness property: For any pair $(x, w) \in \mathcal{R}$*, we have*

$$\Pr\left[\mathsf{Ver}(\mathsf{crs}, x, \pi) = 1 \mid \mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa),\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)\right] = 1.$$

Through the text, we use the term proof system for such systems. Proof systems need to be sound. We now define computational soundness.

Definition 13 (Soundness)**.** *Consider an* $\mathbf{NP}$ *relation* $\mathcal{R}$ *and a non-interactive proof system* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *for* $\mathcal{R}$*. We say that* PS *is (computationally) sound, if for every PPT algorithm* $\mathcal{A}$*, the probability that the following game outputs* 1 *is negligible:*

1. *Run* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$.
2. *Run* $\mathcal{A}$ *on input* crs *and receive a pair* (x, π).
3. *Terminate with output* 1 *if the following two conditions hold; else, terminate with* 0*:*
   (a) *It is* $\mathsf{Ver}(\mathsf{crs}, x, \pi) = 1$.
   (b) *There is no witness* w *such that* $(x, w) \in \mathcal{R}$.

We require a proof system that is zero-knowledge and simulation-extractable, according to the following definitions.

Definition 14 (Zero-Knowledge)**.** *Consider an* $\mathbf{NP}$ *relation* $\mathcal{R}$ *and a non-interactive proof system* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *for* $\mathcal{R}$*. We say that* PS *is zero-knowledge, if there exist PPT algorithms* $\mathsf{TrapSetup}$*,* Sim *such that for every PPT algorithm* $\mathcal{A}$*, the probability that the following game outputs* 1 *is negligibly close to* $1/2$*:*

1. *Sample* $b \leftarrow_{\$} \{0, 1\}$.

2. *Generate a common reference string* crs *as follows:*

   (a) *If* $b = 0$*, run* $\mathsf{crs} \leftarrow \mathsf{Setup}(1^\kappa)$.

   (b) *If* $b = 1$*, run* $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{TrapSetup}(1^\kappa)$.

3. *Run* $\mathcal{A}$ *on input* crs *and obtain state* st *and pairs of statements and witnesses* $(x_1, w_1), \ldots, (x_L, w_L)$.

4. *If there exists some* $j \in [L]$ *with* $(x_j, w_j) \notin \mathcal{R}$*, terminate and output* 0.

5. *Otherwise, for each* $j \in [L]$*, compute* π_j *as follows:*

   (a) *If* $b = 0$*, run* $\pi_j \leftarrow \mathsf{Prove}(\mathsf{crs}, x_j, w_j)$.

   (b) *If* $b = 1$*, run* $\pi_j \leftarrow \mathsf{Sim}(\mathsf{trap}, x_j)$.

6. *Run* $\mathcal{A}$ *on input* st *and* $\pi_1, \ldots, \pi_L$ *and obtain a bit* b'.

7. *Output* 1 *if* $b = b'$*. Otherwise, output* 0.

In this case, we say that $\mathsf{TrapSetup}$ *is the trapdoor setup algorithm and* Sim *is the zero-knowledge simulator for* PS.

Definition 15 (Simulation-Extractability)**.** *Let* $\mathcal{R}$ *be an* $\mathbf{NP}$ *relation and let* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *denote a non-interactive proof system for* $\mathcal{R}$*. Assume that* PS *is zero-knowledge with trapdoor setup algorithm* $\mathsf{TrapSetup}$ *and zero-knowledge simulator* Sim*. We say that* PS *is simulation-extractable, if there is a PPT algorithm* Ext *such that for every PPT algorithm* $\mathcal{A}$*, the probability that the following game outputs* 1 *is negligible:*

1. *Generate* $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{TrapSetup}(1^\kappa)$.

2. *Run* $\mathcal{A}$ *on input* crs *and obtain a state* st *and statements* $x_1, \ldots, x_L$.

3. *For each* $j \in [L]$*, run* $\pi_j \leftarrow \mathsf{Sim}(\mathsf{trap}, x_j)$.

4. *Run* $\mathcal{A}$ *on input* st *and* $\pi_1, \ldots, \pi_L$ *and obtain pairs* $(x_1^*, \pi_1^*), \ldots, (x_T^*, \pi_T^*)$ *of statements and proofs.*

5. *If there exist* $i \in [T]$ *and* $j \in [L]$ *with* $x_i^* = x_j$*, then terminate and output* 0.

6. *For each* $i \in [T]$*, run* $w_i^* := \mathsf{Ext}(\mathsf{trap}, x_i^*, \pi_i^*)$.

7. *If there is an* $i \in [T]$ *s.t.* $\mathsf{Ver}(\mathsf{crs}, x_i^*, \pi_i^*) = 1$ *and* $(x_i^*, w_i^*) \notin \mathcal{R}$*, terminate and output* 1.

8. *Otherwise, output* 0.

In this case, we refer to Ext *as the knowledge extractor of* PS.

There are constructions of non-interactive proof systems satisfying our notion with a common *random* string, e.g., [96]. Depending on the application, we might require also *succinctness* of the proof, which would naturally lead to zk-SNARKs (e.g. [97]).

Definition 16 (Succinctness)**.** *Let* $\mathcal{R}$ *be an* **NP** *relation and let* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *denote a non-interactive proof system for* $\mathcal{R}$*. We say that* PS *is* succinct*, if for any pair of statements and witnesses* $(x, w) \in \mathcal{R}$*, and for arbitrary value of* crs*, i) the length of the proof* $\pi \leftarrow \mathsf{Prove}(\mathsf{crs}, x, w)$ *is given by:*

$$|\pi| = \mathsf{poly}(\kappa, \log |w|, \log |x|),$$

and ii) the verifier runs on time polynomial in $\kappa + |x|$.

Definition 17 (SNARKs)**.** *Let* $\mathcal{R}$ *be an* **NP** *relation and let* $\mathsf{PS} = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *denote a non-interactive proof system for* $\mathcal{R}$*. If* PS *is i) Sound, ii) Zero-Knowledge, iii) Simulation-Extractable and iv) Succinct, then* PS *is a* Succinct Non-Interactive Argument of Knowledge.

2.3.5 Time-Lock Puzzles

We now introduce time-lock puzzles (TLPs) with batch solving following [62, 63].

Definition 18 (Time-Lock Puzzles with Batch Solving)**.** *A* time-lock puzzle *(TLP) with batch*

solving is a tuple of PPT algorithms $\mathsf{TLP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve})$ *with the following syntax:*

- $\mathsf{Setup}(1^\kappa) \to \mathsf{tlpar}$ *takes as input the security parameter and outputs parameters* tlpar.
- $\mathsf{Gen}(\mathsf{tlpar}, s) \to Z$ *takes as input parameters* tlpar *and a solution* s *and outputs a puzzle* Z.
- $\mathsf{Solve}(\mathsf{tlpar}, (Z_i)_{i=1}^L) \to (s_i)_{i=1}^L$ *is deterministic, takes as input parameters* tlpar *and a list of puzzles* Z_i *and outputs a list of solutions* s_i.

Further, we require the following completeness property:

- *For every* $\mathsf{tlpar} \in \mathsf{Setup}(1^\kappa)$*, every* $L \in \mathbb{N}$*, and every list of solutions* $s_1, \ldots, s_L$*, and puzzles* $Z_i \in \mathsf{Gen}(\mathsf{tlpar}, s_i)$ *for all* $i \in [L]$*, we have* $\mathsf{Solve}(\mathsf{tlpar}, (Z_i)_{i=1}^L) = (s_i)_{i=1}^L$.

Definition 19 (CPA Security of TLPs)**.** *Let* $\mathsf{TLP} = (\mathsf{Setup}, \mathsf{Gen}, \mathsf{Solve})$ *be a time-lock puzzle with batch solving and* Δ *be a parameter. We say that* TLP *is* Δ*-CPA secure, if for every PPT algorithm* $\mathcal{A}_{pre}$ *and every PPT algorithm* $\mathcal{A}_{on}$ *such that* $\mathcal{A}_{on}$ *runs in time at most* Δ*, the probability that the following game outputs* 1 *is negligibly close to* $1/2$*:*

1. *Sample* $b \leftarrow_s \{0,1\}$ *and generate* $\mathsf{tlpar} \leftarrow \mathsf{Setup}(1^\kappa)$.
2. *Run* $\mathcal{A}_{pre}$ *on input* tlpar *to obtain a state* st *and two solutions* s_0, s_1.
3. *Generate* $Z \leftarrow \mathsf{Gen}(\mathsf{tlpar}, s_b)$ *and run* $\mathcal{A}_{on}$ *on input* (st, Z).
4. *Obtain a bit* $b' \in \{0,1\}$ *from* $\mathcal{A}_{on}$ *and output* 1 *if* $b = b'$*. Else, output* 0.

We assume that Setup and Gen run in time substantially faster than Δ, and we also assume that running Solve honestly requires time $q_1(\kappa, \Delta) + q_2(\kappa, L)$ for some polynomials q_1, q_2. Note that the running time of Solve does not scale with $\Delta \cdot L$. We can obtain time-lock puzzles with batch opening from any homomorphic time-lock puzzles [58]. There also exists a construction of CPA secure homomorphic time-lock puzzles with transparent setup [59], without random oracles, based on the hardness of computing the order of class groups of imaginary order [98].

Chapter 3: Communication-Efficient Broadcast via Gossiping

In this chapter we will attempt to improve the communication complexity of Broadcast protocols in specific settings, mainly via *gossiping* techniques. We first define probabilistic dissemination techniques for efficient communication, and construct Broadcast protocols that utilize such techniques.

In Section 3.1 we will define a simple dissemination technique, and use it to improve Dolev-Strong's communication almost by a factor of n. However, we note that our protocol is secure only against static adversaries. In Section 3.2 we show how to transform our dissemination process to a similar one, that is secure against adaptive corruptions, given some specific requirements. We use this process to resolve a useful problem which we also define on that chapter, that of $\mathcal{M}$-$Converge$; intuitively this problem aims to match the views of all participating honest parties. Finally, in Sections 3.3 and 3.4 we provide communication-efficient PBB protocols in the bulletin-board and in the trusted PKI model respectively. Both protocols utilize the properties of $\mathcal{M}$-$Converge$.

3.1 Single-Sender Broadcast

We begin by describing our communication-efficient BC protocol in detail. Our protocol replaces Dolev-Strong's SEND-ALL instruction with a SEND-RANDOM instruction. We model SEND-RANDOM with a randomized procedure that we call AddRandomEdges (see Figure 3.1)

1: **procedure** $G \leftarrow$ AddRandomEdges(V, S_2, S_3, S, m)
Input: Set of n nodes V; disjoint subsets of V: S_2, S_3; $S \subseteq V - (S_2 \cup S_3)$; Integer $m \leq n$.
Output: A graph G.

2: Let G be an empty graph with node set V;
3: **for** every node $v \in S$ **do**
4: **for** every node $u \in V$ **do**
5: Add an edge (v, u) to G with probability $\frac{m}{n}$;
6: **return** G;

Figure 3.1: The AddRandomEdges procedure.

that simulates the propagation of messages from honest nodes to the rest of the network in our protocol, between two consecutive rounds. Analyzing it seperately, allows us to argue about the consistency and soundness of our protocol BulletinBB in a structured manner. Any proof that is omitted throughout this section can be found in [35].

3.1.1 The Procedure AddRandomEdges

AddRandomEdges works over a graph G whose set of n vertices V is partitioned into three disjoint sets and which is initially empty, i.e., has no edges. Given an arbitrary partition of V into three disjoint sets S_1, S_2, S_3, and a set $S \subseteq S_1$, AddRandomEdges adds the edge (v, u) to the graph G with probability m/n for every pair of nodes $v \in S$ and $u \in V$. Note that S_1 is fully defined by S_2, S_3 and V as $S_1 = V - (S_2 \cup S_3)$, therefore S_1 is not an input to AddRandomEdges.

The procedure outputs the resulting graph G (i.e., with all the added edges). As we will later see, S represents the set of parties that send a message q at a specific round r and S_2 represents the set of parties that have not received q in a previous round. An edge from $v \in S$ to $u \in V$ represents that party v sends q to party u in round r. We want to compute how many parties in S_2 receive q

(for the first time) during round r, so we study the degree of nodes in S_2. We define the following indicator random variables.

Definition 20. *Let $G \leftarrow$ AddRandomEdges(V, S_2, S_3, S, m). For all $u \in S_2$ let $Z_u \in \{0,1\}$ such that $Z_u = 1$ if and only if u has nonzero degree in G.*

In Lemma 1 we show that the number of nodes in S_2 that acquire an edge in G is at least twice the number of nodes in S. Intuitively, this allows us to show that messages propagate very quickly in our broadcast protocol.

Lemma 1. *Let (S_1, S_2, S_3) be a partition of n nodes into disjoint sets with $\tau = |S_1| \leq \epsilon \cdot n/3$, $|S_2| = \epsilon \cdot n - |S_1|$, $|S_3| = n - \epsilon \cdot n$, where $\epsilon \in (0,1)$ is a constant. Let also $S \subseteq S_1$ with $|S| \geq 2 \cdot \tau/3$ and let $\{Z_u\}_{u \in S_2}$ be the random variables defined by AddRandomEdges(V, S_2, S_3, S, m) per Definition 20. Then for $m \geq 15/\epsilon$,*

$$\Pr\left[\sum_{u \in S_2} Z_u \geq 2 \cdot \tau\right] \geq 1 - p, \text{ where } p = \max\left\{\epsilon \cdot n \cdot e^{-\epsilon \cdot m/9}, \left(\frac{e}{2}\right)^{-\epsilon \cdot m/4}\right\}.$$

3.1.2 Communication-Efficient Broadcast against static adversaries

We now describe our protocol BulletinBB. For simplicity, we describe our protocol for the case where values agreed upon are from the binary domain.

Intuition: From SEND-ALL to Gossiping. As we mentioned in the introduction, our protocol is inspired by the protocol of Dolev and Strong [7] that achieves $O(n^3 \cdot \kappa)$ communication complexity in the bulletin board PKI model. We give a detailed description of the Dolev-Strong protocol here. Each party $i \in [n]$ maintains a set $\mathsf{Extracted}_i$ that is initialized as empty. The protocol proceeds in $t+1$ rounds as follows (Again, $t < n$ is the number of corrupted parties.) In the first

round, the designated sender s signs her input bit and sends the signature to all $n-1$ parties. In rounds $2 \leq r \leq t$, for each bit $b \in \{0,1\}$, if an honest party i has seen at least r signatures on b (including a signature from the designated sender) and b is not in her extracted set, then party i adds b to her local extracted set, signs b and sends the $r+1$ signatures to all $n-1$ parties. In the final round $t+1$, i accepts a bit b iff b is the only bit in $\mathsf{Extracted}_i$.

What makes the above protocol work is the fact that when party i sends the $r+1$ signatures, *all* honest parties see these signatures in the next round, since these signatures are sent to *all* other $n-1$ parties. In the final round, it is not necessary to send again, since holding $t+1$ signatures on b means that at least one honest party has sent $r+1$ signatures upon receiving r signatures on b in round $r < t+1$. Hence, all parties must have received these $r+1$ signatures in round $r+1$ and added b to their extracted sets in that round. In terms of communication complexity, note that all honest parties send an $O(t \cdot \kappa)$-sized message to $n-1$ parties (κ is due to the size of the signature), which results in $O(n^2 \cdot t \cdot \kappa)$ communication. Given that $t = O(n)$, this is $O(n^3 \cdot \kappa)$.

Our protocol does away with the SEND-ALL instructions, and introduces a form of gossiping: it does not require an honest party to send the $r+1$ signatures to all $n-1$ parties. Instead, an honest party sends the $r+1$ signatures to each other party with probability $\frac{m}{n}$. The hope is that after a certain number of rounds, enough honest parties see these messages. As expected, the total number of rounds must now increase. Fortunately, our protocol requires just an additional $R = O(\log n)$ rounds, yielding a communication complexity of $O(n^2 \cdot m \cdot \kappa) = O(n^2 \cdot \lambda\kappa)$ and a round complexity of $O(n)$.

Formal Description and Proof of BulletinBB. Figure 3.2 contains the pseudocode of our protocol from the view of an honest party p (i.e., this is the algorithm that runs at each distributed

1: **procedure** $b' \leftarrow \textsc{BulletinBB}_p()$
Input: A bit b if $p = s$, no input otherwise.
Output: Decision bit b'.

2: $\mathsf{Extracted}_p = \mathsf{Local}_p = \emptyset$;
3: **if** p is the designated sender s **then**
4: $\textsc{Send}(\mathsf{sig}_s(b), [n])$;
5: **for** round $r = 1$ to $t + R$ **do**
6: $\mathsf{Local}_p \leftarrow \mathsf{Local}_p \cup \textsc{Receive}()$;
7: **for** bit $x \in \{0, 1\}$ **do**
8: $S \leftarrow \textsc{DistinctSigs}(x, \mathsf{Local}_p, s)$;
9: **if** $|S| \geq \min\{r, t + 1\} \wedge x \notin \mathsf{Extracted}_p$ **then**
10: $\mathsf{Extracted}_p = \mathsf{Extracted}_p \cup x$;
11: **for** party $i = 1$ to n **do**
12: $\textsc{Send}(\mathsf{sig}_p(x) \cup S, i)$ with probability $\frac{m}{n}$;
13: **return** $b' \in \mathsf{Extracted}_p$ if $|\mathsf{Extracted}_p| = 1$, otherwise **return** canonical bit 0;

Figure 3.2: Our $\textsc{BulletinBB}_p$ protocol for party p.

(honest) node p). It takes as input an initial bit b in the case of the designated sender (no input else) and returns the final bit b'. Whenever the protocol calls $\textsc{DistinctSigs}(x, \mathsf{Local}_p, s)$, the function returns the set of valid signatures from distinct signers on x contained in Local_p if this set includes a signature from s, or it returns $\emptyset$ otherwise. Note three major differences from the Dolev-Strong protocol [7]: (1) we increase the number of rounds from t to $t + R$ (Line 5); (2) instead of sending to all parties we send to each party randomly with probability m/n (Line 12); (3) for all rounds $r \geq t + 1$ we do not require $r + 1$ signatures to add to the extracted set but just $t + 1$—that is why we use the expression $\min\{r, t+1\}$ in Line 9. We now continue with the proof of consistency and soundness of BulletinBB. We first define, using notation consistent with AddRandomEdges, the following sets of parties (w.r.t. a bit b and a round r):

1. $S(b, r)$: honest parties i that added b to their $\mathsf{Extracted}_i$ set *at* round r;
2. $S_1(b, r)$: honest parties i that added b to their $\mathsf{Extracted}_i$ set *by* round r;
3. $S_2(b, r)$: honest parties i that have *not* added b to their $\mathsf{Extracted}_i$ set by r.

Define S_3 be the set of malicious parties ($|S_3| = n - \epsilon n$). We show (roughly) that the number of parties that receive a message at round r' that was sent at round $r < r'$ increases exponentially with $r' - r$ with overwhelming probability.

Lemma 2 (Gossiping bounds)**.** *For a specific bit b, let r be the first round of* BULLETINBB *where an honest party i adds b to* $\mathsf{Extracted}_i$*. Let $R = \lceil \log_3(\epsilon \cdot n) \rceil$. Let p be the probability defined in Lemma 1. Then:*

1. *For all rounds ρ such that $r \leq \rho \leq r + R$ and $|S_1(b, \rho - 1)| \leq \epsilon \cdot n/3$, we have that with probability at least $(1-p)^{\rho - r}$:*

$$|S(b, \rho)| \geq (2/3) \cdot |S_1(b, \rho)| \text{ and } |S_1(b, \rho)| \geq 3^{\rho - r}.$$

2. *Let $r^* > r$ be a round such that $|S_1(b, r^* - 1)| > \epsilon \cdot n/3$. Then, $|S_1(b, r^*)| = \epsilon \cdot n$ with probability at least $(1 - p^*) \cdot (1-p)^{r^* - r - 1}$, where $p^* = \epsilon \cdot n \cdot e^{-2\epsilon \cdot m/9}$.*

Lemma 3 (Consistency: BULLETINBB)**.** *Let $R = \lceil \log_3(\epsilon n) \rceil$, $m = \Theta(\lambda)$.* BULLETINBB *satisfies consistency (see Definition 2).*

Proof. Suppose an honest party i adds bit b to $\mathsf{Extracted}_i$ at some round r. We prove that by the end of the protocol all honest parties j add b to their $\mathsf{Extracted}_j$ sets with probability $1 - \mathsf{negl}(\lambda)$—this means that all honest parties have identical Extracted sets by the end of the protocol, which is equivalent to consistency. We distinguish the two cases:

Case $r < t + 1$: We distinguish two cases. If $S_1(b, r) > \epsilon \cdot n/3$, then by Item (2) of Lemma 2, all $\epsilon \cdot n$ honest parties add bit b in their extracted set by the next round with probability at least $1 - \epsilon \cdot n \cdot e^{-2\epsilon \cdot m/9} = 1 - \mathsf{negl}(\lambda)$, since $n = poly(\lambda)$. Now, if $S_1(b, r) \leq \epsilon \cdot n/3$, let $\mathbf{r}$ be the round

$r+R-1$. If $S_1(b, \mathbf{r}) > \epsilon \cdot n/3$, the previous case applies. Otherwise, Item (1) of Lemma 2 applies, meaning that at round $\mathbf{r}+1 = r+R$ we have $S_1(b, r+R) \geq 3^R = 3^{\lceil \log_3(\epsilon \cdot n) \rceil} \geq 3^{\log_3(\epsilon \cdot n)} = \epsilon \cdot n$, with probability at least $(1-p)^R \geq 1-R \cdot p$, by Bernoulli's inequality[1] and since $-p \geq -1, R \geq 1$. Note that for $m = \Theta(\kappa)$, $R \cdot p$ is $\mathsf{negl}(k)$, since $n = poly(\kappa)$.

Case $r \geq t+1$: Suppose an honest party i adds bit b to $\mathsf{Extracted}_i$ at some round $r \geq t+1$. Then, i has received valid signatures on b from $t+1$ distinct parties. Thus, an honest party j added bit b to $\mathsf{Extracted}_j$ at some round $r'' < t+1$. So, case $r'' < t+1$ applies to honest party j. Thus all honest parties add b to their $\mathsf{Extracted}$ sets by the end of the protocol, with probability $1 - \mathsf{negl}(\kappa)$. □

Lemma 4 (Soundness: BULLETINBB)**.** *Let $R = \lceil \log_3(\epsilon \cdot n) \rceil$, $m = \Theta(\kappa)$. BULLETINBB satisfies soundness, per Definition 2.*

Proof. Follows from the proof of consistency in Lemma 3. After $R = \lceil \log_3(\epsilon \cdot n) \rceil$ rounds, all honest parties have received the bit of the honest sender, with probability $1 - \mathsf{negl}(\kappa)$. □

Theorem 1 (Communication complexity: BULLETINBB)**.** *Let $m = \Theta(\kappa)$ and $R = \lceil \log_3(\epsilon n) \rceil$. The total number of bits exchanged by all parties in BULLETINBB is $O(n^2 \cdot \kappa^2)$, with probability $1 - \mathsf{negl}(\kappa)$.*

Proof. Every honest party sends at most one time for each bit to a number of $O(m) = O(\kappa)$ parties with overwhelming probability in κ, a message of at most t signatures. Since there are $O(n)$ honest parties, $t = O(n)$ and the size of each signature is κ, the total number of bits exchanged is $O(n^2 \cdot m \cdot \kappa) = O(n^2 \cdot \kappa^2)$. □

[1] For every $x, r \in \Re, x \geq -1, r \geq 1$ it holds that $(1+x)^r \geq 1 + rx$.

3.2 The $\mathcal{M}$-$Converge$ Problem

In this section we introduce the $\mathcal{M}$-$Converge$ problem, an efficient solution of which is used as a black box in our parallel broadcast protocols. Informally, in the $\mathcal{M}$-$Converge$ problem, honest parties begin with individual subsets (of a fixed set $\mathcal{M}$) as inputs and in the end of the protocol all honest parties must have a superset of the union of all the initial honest-owned subsets. We want to design $\mathcal{M}$-$Converge$ protocols in the presence of an adversary that can corrupt at most t parties, adaptively. We now define the $\mathcal{M}$-$Converge$ problem formally.

Definition 21 (t-secure $\mathcal{M}$-$Converge$ problem)**.** *Let $\mathcal{M} \subseteq \{0,1\}^*$ be an efficiently recognizable set (This is a set for which membership can be efficiently decided.) A protocol Π executed by n parties where every honest party p initially holds input set $M_p \subseteq \mathcal{M}$ and constraint set $\mathcal{C}_p \subseteq \mathcal{M}$ is a* t-secure $\mathcal{M}$-$Converge$ protocol *if all remaining honest parties upon termination, with probability $1 - \mathsf{negl}(\kappa)$, output a set S_p s.t.*

$$S_p \supseteq \bigcup_{p \in \mathcal{H}} M_p - \bigcup_{p \in \mathcal{H}} \mathcal{C}_p \,,$$

whenever at most t parties are corrupted and where $\mathcal{H}$ is the set of honest parties in the beginning of the protocol.

We note that the trivial solution of outputting $\mathcal{M}$ does not work: We cannot necessarily enumerate the elements of $\mathcal{M}$ since we only assume that membership can be efficiently decided. An example of such a setting is where $\mathcal{M}$ consists of signatures of n parties on a bit b, computed with the parties' secret keys. In such a setting, PPT parties can easily verify membership in $\mathcal{M}$ but cannot enumerate $\mathcal{M}$, even if $\mathcal{M}$ is polynomial-sized (Note that the latter could be

procedure $(G, H') \leftarrow \text{AddRandomEdgesAdaptive}_{\mathcal{A}}(V, S, H, m)$
Input: Set of n nodes V; sets S and H with $S \subseteq H \subseteq V$; Integer $m \leq n$.
Output: A graph G a set of nodes $H' \subseteq H$.

Let G be an empty graph with node set V;
for every node $v \in S$ **do**
  for every node $u \in V$ **do**
    Add a directed edge (v, u) to G with probability m/n;
Initialize RevealedEdges to be all edges incident to nodes in $v \in V - H$;
while $|H| \geq \epsilon \cdot n$ **do**
  $v_i \leftarrow \mathcal{A}(V, H, \text{RevealedEdges})$ such that $v_i \in H$;
  if $v_i \neq null$ **then**
    Add $\{(u, v_i) : u \in \text{in\_neighbor}(v_i)\}$ to RevealedEdges;
    $H = H - v_i$;
  else break;
Set $H' = H$;
return (G, H');

Figure 3.3: The experiment with an adaptive adversary, AddRandomEdgesAdaptive.

true depending on the signature scheme.) There exists a very simple t-secure $\mathcal{M}$-$Converge$ protocol: All honest parties just send their local sets M_p to all other parties, in one round. Unfortunately, the communication complexity of this protocol is $\Omega(n^2 \cdot \min_{p \in \mathcal{H}}\{|M_p - \mathcal{C}_p|\})$, which is prohibitively expensive. We later propose a protocol, $\mathcal{M}$-ConvergeRandom (see Figure 3.6) that uses gossiping to reduce communication to $\tilde{O}(n \cdot |\mathcal{M}| + n^2)$. First, we will introduce some necessary tools for the analysis and clear exposition of our protocol; the functionality $\mathcal{F}_{\text{prop}}$ and the procedure AddRandomEdgesAdaptive, required for analyzing $\mathcal{F}_{\text{prop}}$.

3.2.1 The Procedure AddRandomEdgesAdaptive

In AddRandomEdgesAdaptive, a set of honest senders $S \subseteq H$ (H is the set of initial honest nodes) is sending a message x to the rest of the parties, by picking each party independently to be a recipient of the message with probability m/n. The set of malicious nodes are not fixed a

priori and the adversary $\mathcal{A}$ decides who to corrupt *after* the edges have been placed. Let's define a specific set of honest parties $S_2 \subseteq H$ as the *target set* which does not contain any of the senders in S (The meaning of target set will depend on our application.) Our goal is to prove that even after the adversary has finished with the adaptive corruptions, a sufficient number (in particular $2 \cdot |H - S_2|$) of the remaining honest nodes in S_2 is receiving message x with overwhelming probability. Clearly, if we do not confine the view of the adversary, we cannot prove anything meaningful; the adversary can go ahead and corrupt exactly the nodes that received x. For this reason, in AddRandomEdgesAdaptive we strategically allow the adversary to access just the list RevealedEdges (see Line 8) before making the next corruption—these are the edges that correspond to nodes that *have already been corrupted* (Note that this restriction of the adversary is enforced by the implementation of our protocol later through public key encryption of sent messages.) We formalize this intuition in Lemma 5, the proof of which can be found in the Appendix.

Definition 22. *Let* $(G, H') \leftarrow$ *AddRandomEdgesAdaptive*$_{\mathcal{A}}(V, S, H, m)$. *Let* $S_2 \subseteq H$. *Then for all* $u \in S_2$ *define random variables* $Z_u \in \{0, 1\}$ *such that* $Z_u = 1$ *if and only if* u *has nonzero degree in* G *and* $u \in H'$.

Lemma 5. *Let* $\epsilon \in (0, 1)$, $S_2 \subseteq H$ *with* $|H - S_2| \leq \epsilon \cdot n/3$ *and* $S \subseteq H - S_2$ *with* $|S| \geq 2|H - S_2|/3$. *Let* $(G, H') \leftarrow$ *AddRandomEdgesAdaptive*$_{\mathcal{A}}(V, S, H, m)$. *Then for* $m \geq 19/\epsilon$,

$$\Pr\left[\sum_{u \in S_2} Z_u \geq 2|H - S_2|\right] \geq 1 - p,$$

where $p = \max\{n \cdot e^{-4\epsilon \cdot m/45}, (\frac{e}{2})^{-\epsilon \cdot m/2}\}$.

Functionality: $\mathcal{F}_{\mathsf{prop}}$

Let n be the number of parties and $m = 10/\epsilon + \kappa$. For every party $i \in [n]$, $\mathcal{F}_{\mathsf{prop}}$ keeps a set O_i which is initialized to $\emptyset$. Let M_i be party i's input messages' set.

- On input $(\mathsf{SendRandom}, M_i)$ by honest party i:
  - For all $x \in M_i$ and for all $j \in [n]$ add (i, x) to O_j with probability m/n;
  - **return** M_i to adversary $\mathcal{A}$;
  - **return** O_i to party i.
- On input $(\mathsf{SendDirect}, \mathbf{x}, J)$ by adversary $\mathcal{A}$ (for a corrupted party i):
  - Add $(i, x[j])$ to O_j for all $j \in J$;
  - **return** O_i to adversary $\mathcal{A}$.

Figure 3.4: Functionality $\mathcal{F}_{\mathsf{prop}}$.

3.2.2 The ideal functionality $\mathcal{F}_{\mathsf{prop}}$

To facilitate the exposition of our $\mathcal{M}$-ConvergeRandom protocol, we use an ideal functionality $\mathcal{F}_{\mathsf{prop}}$ (Figure 3.4). In short, $\mathcal{F}_{\mathsf{prop}}$ enables a party i to send a set of messages M to, on average, m out of n randomly selected parties without leaking *which the recipients are* to the adversary. Note that each message in M is sent to a different set of parties. In our protocol $\mathcal{F}_{\mathsf{prop}}$ is called, via $(\mathsf{SendRandom}, M)$, by all honest parties i in the beginning of every round β and returns a set O_i, in the end of round β, which contains messages sent from other parties to i in the beginning of round β. The adversary in $\mathcal{F}_{\mathsf{prop}}$ is given a special interface to the functionality via the instruction $(\mathsf{SendDirect}, \mathbf{x}, J)$. This captures the adversary's ability of sending specific messages to specific parties. Through $(\mathsf{SendDirect}, \mathbf{x}, J)$, the adversary can send messages in $\mathbf{x}$ to parties specified in the vector J directly rather than randomly. We also assume that the adversary learns the input set of an honest party to $\mathcal{F}_{\mathsf{prop}}$. Finally the adversary gets access to all the sets O_i for the parties i that have been corrupted. We note here that $\mathcal{F}_{\mathsf{prop}}$ does not maintain state across calls and therefore all $O_i = \emptyset$, in the beginning of every round.

Implementing the ideal functionality $\mathcal{F}_{\mathsf{prop}}$ with PROPAGATE(). In Figure 3.5 we define the process that instantiates $\mathcal{F}_{\mathsf{prop}}$ and give it the fitting name PROPAGATE(). As usual, we describe the protocol PROPAGATE() from the view of a party p. Consistent with the interface of $\mathcal{F}_{\mathsf{prop}}$, PROPAGATE() takes as input a set of messages M_p (that are to be sent out to other parties) and returns a set of messages O_p that were sent to p. In the first step of PROPAGATE(), every party creates a fresh pair of secret and public keys.

Next, note that PROPAGATE() does not send a message $x \in M_p$ directly to party j (with probability m/n) since this would reveal the recipient of x to the adversary. Instead, before sending, it locally computes a list $\mathcal{L}_j$, for every party $j \in [n]$, and adds x to $\mathcal{L}_j$ with probability m/n. All lists $\mathcal{L}_j$ are padded to the size of the maximum sized list for the current call of PROPAGATE by the party (say party p), which we call Λ_p and are encrypted using the fresh public keys (Recall that $\mathcal{M}$ is the fixed set of the $\mathcal{M}$-$Converge$ problem.) Then *the plaintext lists are erased from memory* and only after erasure the encrypted lists are sent out. In the end, the fresh secret key is also erased from memory. Intuitively, security is guaranteed because (i) the communication pattern of each sender does not reveal anything (irrespectively of who the recipient of x is, the adversary just sees equally-sized encrypted lists sent to all parties from each sender); and (ii) even if the adversary adaptively corrupts a party, no information about who that party sent to is revealed (because of erasure of plaintext lists and secret key)—the only information that is revealed is from whom that party received, which is harmless. We show that the lists $\mathcal{L}_j$ constructed by PROPAGATE$_p$() is of the same order as the message list M_p of party p with overwhelming probability.

Lemma 6. *If M_p is the input set of party p, the number of elements added to list $\mathcal{L}_j$ at Line 7 of* PROPAGATE() *is $\leq |M_p| \cdot 2m/n \leq \Lambda_p$ with probability $1 - \mathsf{negl}(\kappa)$.*

1: **procedure** PROPAGATE$_p$(SendRandom, M_p)
Input: A set of of messages M_p.
Output: A set of messages O_p.

2: Set $(sk_p, pk_p) \leftarrow \mathsf{KeyGen}(1^\kappa)$; ▷ at time 0
3: SEND$(pk_p, [n])$;
4: RECEIVE(); ▷ at time Δ
5: **for** all $x \in M_p$ **do**
6: **for** $j = 1$ to n **do**
7: Add x to list $\mathcal{L}_j$ with probability m/n;
8: Let $\Lambda_p = 2m \left\lceil \frac{|M_p|}{n} \right\rceil$;
9: **for** $j = 1$ to n **do**
10: Pad list $\mathcal{L}_j$ to maximum size Λ_p;
11: $ct_j \leftarrow \mathsf{Enc}(pk_j, \mathcal{L}_j)$;
12: Erase $\mathcal{L}_j$ from memory;
13: **for** $j = 1$ to n **do**
14: SEND(ct_j, j);
15: $\mathcal{C} \leftarrow$ RECEIVE(); ▷ at time 2Δ
16: **for** all $ct \in \mathcal{C}$ **do**
17: Decrypt ct using sk_p and output a list $\mathcal{L}$;
18: Add $\mathcal{L}$ to O_p;
19: Erase sk_p from memory;
20: **return** O_p;

Figure 3.5: PROPAGATE(), a secure instantiation of $\mathcal{F}_{\mathsf{prop}}$.

Lemma 7. *Let s be the length in bits of a message in $\mathcal{M}$. The communication complexity of* PROPAGATE() *for party p with input M_p, is $O(\max\{n, |M_p|\} \cdot m \cdot s)$.*

Security of PROPAGATE(). Finally we need to prove that PROPAGATE() securely instantiates $\mathcal{F}_{\mathsf{prop}}$. The key property that we leverage is that each sender sends the same amount of information to every other party, regardless of their inputs. This trivializes the simulation of honest parties' communication in the protocol, since it is independent of the actual values they input to PROPAGATE(). To show that a specific implementation realizes $\mathcal{F}_{\mathsf{prop}}$, we are using the definition of synchronous MPC in the standalone model by Canetti [99]. For both the definition and the full proof, we direct

1: **procedure** $S_p \leftarrow \mathcal{M}$-ConvergeRandom$_p(M_p, \mathcal{C}_p)$
Input: Sets $M_p \subseteq \mathcal{M}, \mathcal{C}_p \subseteq \mathcal{M}$
Output: A set S_p.

2: **for** round $\beta = 1$ to $\lceil \log(\epsilon \cdot n) \rceil$ **do**
3: Receive$^{\mathcal{F}_{\mathsf{prop}}} \leftarrow \mathcal{F}_{\mathsf{prop}}(\mathsf{SendRandom}, M_p - \mathcal{C}_p)$;
4: $\mathsf{Local}_p \leftarrow \mathsf{Local}_p \cup$ Receive$^{\mathcal{F}_{\mathsf{prop}}}$;
5: $\mathcal{C}_p = \mathcal{C}_p \cup M_p$;
6: $M_p = \mathsf{Local}_p \cap \mathcal{M}$;
7: **return** M_p;

Figure 3.6: Our $\mathcal{M}$-ConvergeRandom$_p$ protocol.

the reader to [35].

Lemma 8. *Assuming a CPA-secure PKE scheme* (KeyGen, Enc, Dec) *and secure erasure,* Propagate() *t-securely computes $\mathcal{F}_{\mathsf{prop}}$.*

3.2.3 The $\mathcal{M}$-ConvergeRandom Protocol

We now analyze the $\mathcal{M}$-ConvergeRandom protocol, depicted in Figure 3.6, solving the $\mathcal{M}$-$Converge$ problem with improved communication. Our protocol proceeds in $\lceil \log(\epsilon \cdot n) \rceil$ rounds. In each round, every honest party $p \in [n]$ uses $\mathcal{F}_{\mathsf{prop}}$ to send messages that are have not been sent by p before, to a few randomly selected parties (Line 3). To decide what to send in the next round, p gathers all newly received messages (Line 4), and keeps only the valid messages that p has not sent before. Messages $m \notin \mathcal{M}$, sent by the adversary can be safely discarded and so can any message $m \in \mathcal{C}_p$.

In Lemma 9 we prove the single-shot security of $\mathcal{M}$-ConvergeRandom (for one message from one party). Then, in Theorem 2 we prove the full security of the protocol. The proof of Lemma 9, which provides an interesting intuition as to how to analyze such message dissemination, can be found in the Appendix.

Lemma 9 (Propagation in presence of an adaptive adversary)**.** *Fix an initially-honest party p and a message $\mathsf{m}^* \in M_p - \bigcup_{h \in \mathcal{H}} \mathcal{C}_h$. Then, with probability $1 - \mathsf{negl}(\kappa)$, all remaining honest parties after $\mathcal{M}$-ConvergeRandom terminates have received m^*.*

Theorem 2. *Protocol $\mathcal{M}$-ConvergeRandom from Figure 3.6 is an adaptively t-secure $\mathcal{M}$-Converge protocol for all $t < (1-\epsilon) \cdot n$ and fixed $\epsilon \in (0,1)$. The number of bits sent by one party is*

$$O(n \log n \cdot m \cdot s + \sum_{i=1}^{\lceil \log \epsilon n \rceil} |M_p^{(i)} - \mathcal{C}_p^{(i)}| \cdot m \cdot s),$$

where s is the length in bits of a message in $\mathcal{M}$ and $M_p^{(i)} - \mathcal{C}_p^{(i)}$ denotes the set given as input to the i-th call of Propagate.

Proof. Security follows by applying Lemma 9 for all initially-honest parties and for all of their initial messages. Every message m^* is delivered to all honest parties that remain after the termination of the protocol, as required by Definition 21. The communication complexity (CC) is dominated by the calls to $\mathcal{F}_{\mathsf{prop}}$. Each call to $\mathcal{F}_{\mathsf{prop}}$ is securely instantiated by executing Propagate with the same input set (Lemma 8). By Lemma 7, the i-th such call to Propagate invokes a total of $O(\max\{n, |M_p^i - \mathcal{C}_p^i|\} \cdot m \cdot s) = O(n \cdot m \cdot s + |M_p^{(i)} - \mathcal{C}_p^{(i)}| \cdot m \cdot s)$ communication, where $M_p^i - \mathcal{C}_p^i$ is the corresponding set of messages input to Propagate at that call. Thus, the number of bits sent by one party is

$$\sum_{i=1}^{\lceil \log \epsilon n \rceil} O(n \cdot m \cdot s + |M_p^{(i)} - \mathcal{C}_p^{(i)}| \cdot m \cdot s) = O(n \log n \cdot m \cdot s + \sum_{i=1}^{\lceil \log \epsilon n \rceil} |M_p^{(i)} - \mathcal{C}_p^{(i)}| \cdot m \cdot s)$$

with probability $1 - \mathsf{negl}(\kappa)$. □

3.3 Parallel Broadcast in the Bulletin PKI Model

We present a protocol to achieve PBC in the Bulletin PKI model.

```
1: procedure {b_1, ..., b_n} ← BulletinPBB_p(b_p)
       Input: Local bit b_p.
       Output: Decision bits b_1, ..., b_n.
   ▷ Fix set M = {sig_v([b, s]) such that b ∈ {0,1}, s, v ∈ [n]}

2:     Extracted_p^i = ∅, for i = 1, ..., n;                    ▷ global variable
3:     Local_p = ∅;                                             ▷ global variable
4:     Send(Sig_p([b_p, p]), [n]);
5:     for round r = 1 to t + 1 do                              ▷ t: bound on dishonest parties
6:         AddMySignature_p(r);
7:         ForwardSignatures_p(r);
8:     for slot i = 1, ..., n do
9:         return b_i ∈ Extracted_p^i if |Extracted_p^i| = 1, else return canonical bit 0;
```

Figure 3.7: The PBB protocol BulletinPBB in the Bulletin-Board PKI model; It invokes the procedures AddMySignature and ForwardSignatures.

```
1: procedure AddMySignature_p(r)

2:     Local_p ← Local_p ∪ Receive();
3:     for all [b, s] such that b ∉ Extracted_p^s do
4:         if Local_p contains ≥ r valid signatures on [b, s] (including Sig_s([b, s])) then
5:             Add b to Extracted_p^s;
6:             Add Sig_p([b, s]) to Local_p;
```

Figure 3.8: The procedure AddMySignature, which performs checks and adds the party's signature to a list of signatures.

This lack of trusted setup induces additional difficulty to the problem. Still, the proposed protocol uses communication cubic in n. We describe the high-level idea of the protocol. Each party internally maintains state for every signature. Each triplet (b, s, j) (for $b \in \{0, 1\}, s, j \in [n]$) defines a specific signature, so there could be $2 \cdot n^2$ signatures overall, each of size κ (where κ denotes the security parameter). The propagation of a specific signature $\sigma_j := \mathsf{Sig}_j(b, s)$ is independent of whether b was added to $\mathsf{Extracted}_i^s$ at that given round. Instead, whenever a party i observes a new signature, i.e. $\mathsf{Sig}_j(b, s)$ for some (b, s, j) that i never observed before, they proceed to propagate the signature exactly twice. If the signature was observed during the

1: **procedure** ForwardSignatures$_p(r)$

2: **if** $r \leq t$ **then**

3: Let C_p contain all signatures that p has propagated exactly twice via ConvergeRandom;

4: $\mathsf{Local}_p \leftarrow \mathcal{M}$-ConvergeRandom$(\mathsf{Local}_p, C_p)$;

Figure 3.9: The procedureForwardSignatures; calls $\mathcal{M}$-ConvergeRandom, so that each honest party conveys its internal view to all parties.

subround of some $\mathcal{M}$-ConvergeRandom, then they continue propagating it only during the next subround (due to how $\mathcal{M}$-ConvergeRandom updates C_p). They also initiate the next round's $\mathcal{M}$-ConvergeRandom with the signature being on their input set but not on their constraint set. Notice that $\mathcal{M}$-ConvergeRandom is defined with respect to the set $\mathcal{M}$. Here, we fix $\mathcal{M}$ to consist of all of the parties' valid signatures for messages $[b, s]$, where b is a bit and s a slot in $[n]$. Next, we prove the security of our protocol.

Lemma 10 (Parallel Consistency: BulletinPBB)**.** *BulletinPBB satisfies parallel consistency (Def. 3) in the* $(\mathcal{F}_{\mathsf{prop}})$*-hybrid world, with probability* $1 - \mathsf{negl}(\kappa)$.

Proof. Suppose for some slot s, an honest party p adds bit b to $\mathsf{Extracted}_p^s$ at some round r. We prove that by the end of the protocol, all honest parties j add b to their $\mathsf{Extracted}_j^s$ sets with probability at least $1 - \mathsf{negl}(\kappa)$. We distinguish the following cases according to the round when p adds bit b to $\mathsf{Extracted}_p^s$ (We sometimes omit "with probability $1 - \mathsf{negl}(\kappa)$" when it is clear from the context.)

If $r \leq t$, then p has at least r distinct, valid signatures for (b, s) and also p creates its own signature. So, p observed each of these $\geq r + 1$ signatures for the first time at some round $r - k$ where $k \in \{0, \ldots, r - 1\}$. For every signature σ with $k \geq 1$, p called a $\mathcal{M}$-ConvergeRandom with σ in its initial input set M_p and not in its constraint set C_p. (Suppose not. Then σ is observed

during some $\mathcal{M}$-ConvergeRandom by p, sent once with Propagate and never sent again by p, a contradiction, since only signatures sent twice go in $\mathcal{C}_p$.) For the signatures with $k = 0$, p initiates a $\mathcal{M}$-ConvergeRandom during round r. From Theorem 2, by round $r + 1$ every honest party j has $\geq r + 1$ valid signatures for (b, s) and adds b to their $\mathsf{Extracted}_j^s$ set during $\textsc{AddMySignature}_j(r + 1)$.

Else, if $r > t$ then, p observes $t + 1$ valid signatures for (b, s). At least one of the signatures comes from an honest party h. So, h has added b to $\mathsf{Extracted}_h^s$ by some round $r' \leq t$. Thus, for honest party h the case $r' \leq t$ can be applied, and so all honest parties j add b to their $\mathsf{Extracted}_j^s$ sets by round $r' + 1$. □

Lemma 11 (Parallel Soundness: BulletinPBB)**.** *BulletinPBB satisfies parallel soundness (Definition 3), in the $(\mathcal{F}_{\mathsf{prop}})$-hybrid world.*

Proof. Follows directly from the proof of consistency (Lemma 10) and the idealized signature scheme. Every honest sender s with input bit b_s, executes Line 4 and thus sends to all parties their signature for (b_s, s) at the start of the protocol. So, all honest parties h add b_s to their $\mathsf{Extracted}_h^s$ sets. By the idealized signature scheme, no other bit b' could bare a valid signature from honest sender s, thus no other bit b' could be in an h's $\mathsf{Extracted}_h^s$ set. □

The proof of Theorem 3 can be found in the Appendix.

Theorem 3 (Communication Complexity of BulletinPBB)**.** *The total number of bits exchanged by all parties in BulletinPBB is $\tilde{O}(n^3 \cdot \kappa^2)$.*

Functionality: $\mathcal{F}_{\mathsf{mine}}$

$\mathcal{F}_{\mathsf{mine}}$ is parameterized by parties $1, \ldots, n$ and "mining" probability p_{mine}. Let $s \in [n]$ and $b \in \{0, 1\}$. Let $call$ be vector of n entries initialized with -1.

- On input (Mine, b, s) from party i:
  - If $call_i = -1$ output $b = 1$ with probability p_{mine} or $b = 0$ with probability $1 - p_{\text{mine}}$ and set $call_i = b$;
  - Else output $call_i$.
- On input $(\mathsf{Verify}, b, s, j)$ from party i output 1 if $call_j = 1$ and 0 otherwise.

Figure 3.10: Functionality $\mathcal{F}_{\mathsf{mine}}$.

3.4 Parallel Broadcast in the Trusted PKI Model

In this section we use trusted setup to reduce the communication complexity of PBC from cubic to quadratic, at the cost of a stronger assumption. Our new protocol, TrustedPBB, uses a slight modification of protocol $\mathcal{M}$-ConvergeRandom. TrustedPBB (Figure 3.11) is described in a hybrid world where two functionalities exist. The first functionality is $\mathcal{F}_{\mathsf{prop}}$ which was presented and instantiated in Section 3.2.2.

The second functionality is $\mathcal{F}_{\mathsf{mine}}$ (Figure 3.10), which was also presented in Chan et al. [16] and was shown to be instantiated from standard assumptions (with setup) by [14]. We assume that when a party calls $\mathcal{F}_{\mathsf{mine}}$ then the functionality returns instantaneously. A party P_i in our protocol queries $\mathcal{F}_{\mathsf{mine}}$ on input (Mine, b, s) in some round r, where $s \in [n]$ refers to one of the slots and $b \in \{0, 1\}$. If it receives response 1, it considers itself a member of a randomly selected subset of all the parties, which we refer to as the "(b, s)-*committee*". More concretely, when $\mathcal{F}_{\mathsf{mine}}$ receives such a query, it flips a random coin to decide whether the party p is in that committee. $\mathcal{F}_{\mathsf{mine}}$ keeps the information and returns the same answer to all future identical queries by any party. Next, we provide some definitions and necessary results.

Definition 23 ((b, s)-committee)**.** *For each pair of bit b and slot s, the (b, s)-committee is a subset of parties such that for each party c in the (b, s)-committee, whenever the $\mathcal{F}_{\mathsf{mine}}$ is queried on input $(\mathsf{Verify}, b, s, c)$, $\mathcal{F}_{\mathsf{mine}}$ outputs 1.*

Lemma 12 (Honest Committees)**.** *Let $R = 2\kappa/\epsilon$ and let the probability of success for $\mathcal{F}_{\mathsf{mine}}$ be $p_{\mathsf{mine}} = \min\{1, \kappa/(\epsilon \cdot n)\}$. Then, with probability $1 - \mathsf{negl}(\kappa)$, for each bit $b \in \{0, 1\}$ and slot $s \in [n]$, the (b, s)-committee contains (i) at least one honest party and (ii) at most R dishonest parties.*

Definition 24 (Valid r-batch)**.** *A valid r-batch on pair (b, s) is the element*

$$b||s||\mathsf{SIG}_r,$$

where SIG_r is a set of at least r signatures on $[b, s]$ consisting of one signature from party s and at least $r - 1$ signatures from parties in the (b, s)-committee.

Our protocol (see Figure 3.11) is inspired by the single-sender protocol of Chan et al. [16]. In particular, our protocol works in $R+1$ rounds as follows Every party P_j maintains n $\mathsf{Extracted}_s^j$ and Voted_s^j sets, $s \in [n]$, that are initialized as $\emptyset$. Roughly speaking, a bit $b \in \mathsf{Extracted}_s^j$ if P_j has observed a valid r-batch on $[b, s]$; a bit $b \in \mathsf{Voted}_s^j$ if it has already been revealed that P_j is part of the (b, s)-committee (i.e., p has "voted"). In round 0, party i (for all $i \in [n]$) signs her input bit b_i, adds it to her $\mathsf{Extracted}_i^i$ set and sends b_i to all $n - 1$ parties along with its signature on b_i, $\mathsf{Sig}_i([b_i, i])$. Afterwards, the distribution and voting phases follow, for every round $r = 1, \ldots, R$. These are non-trivial modifications (for many slots and using parallel gossiping) of the distribution and voting phases of ChBC. Our new distribution/voting phases, for round $r \leq R$, work as follows.

1. (Distribution) An honest party p first collects the set $\mathcal{V}$ of valid r-batches $v_1, \ldots, v_w$ on

1: **procedure** $\{b_1, \ldots, b_n\} \leftarrow$ TRUSTEDPBB$_p(b_p)$
Input: Local bit b_p.
Output: Decision bits $b_1, \ldots, b_n$.

2: $\mathsf{Extracted}_p^i = \emptyset$, for $i = 1, \ldots, n$; ▷ global variable
3: $\mathsf{Voted}_p^i = \emptyset$, for $i = 1, \ldots, n$; ▷ global variable
4: $\mathsf{Local}_p = \emptyset$; ▷ global variable
5: SEND$(\mathsf{Sig}_p([b_p, p]), [n])$;
6: **for** round $r = 1$ to $R + 1$ **do** ▷ R is also a global variable
7: DISTRIBUTE$_p(r)$;
8: VOTE$_p(r)$;
9: **for** slot $i = 1, \ldots, n$ **do**
10: **return** $b_i \in \mathsf{Extracted}_p^i$ if $|\mathsf{Extracted}_p^i| = 1$ else **return** canonical bit 0;

1: **procedure** DISTRIBUTE$_p(r)$

2: $\mathsf{Local}_p \leftarrow \mathsf{Local}_p \cup$ RECEIVE();
3: Let $\mathcal{V} = \{v_i\}$ be valid r-batches (in Local_p) on $\{[b_i, s_i]\}$ s.t. $b_i \notin \mathsf{Extracted}_p^{s_i}$;
4: **for** all $v_i \in \mathcal{V}$ **do** Add b_i to $\mathsf{Extracted}_p^{s_i}$;
5: **if** $r \leq R$ **then** $\mathsf{Local}_p \leftarrow \mathcal{M}$-CONVERGERANDOM$_p(\mathcal{V}, \emptyset, k^*)$;

1: **procedure** VOTE$_p(r)$

2: **if** $r \leq R$ **then**
3: Let $\mathcal{V} = \{v_i\}$ be valid r-batches (in Local_p) on $\{[b_i, s_i]\}$ s.t. $b_i \notin \mathsf{Voted}_p^{s_i}$;
4: **for** all $v_i \in \mathcal{V}$ s.t. $\mathcal{F}_{\mathsf{mine}}(\mathsf{Mine}, b_i, s_i) = 1$ **do**
5: Add b_i to $\mathsf{Voted}_p^{s_i}$;
6: Add b_i in $\mathsf{Extracted}_p^{s_i}$ if $b_i \notin \mathsf{Extracted}_p^{s_i}$;
7: Extend v_i to a valid $(r+1)$-batch v_i' by adding p's signature on $[b_i, s_i]$;
8: SEND$(v_i', [n])$;

Figure 3.11: TRUSTEDPBB protocol, which calls two procedures, DISTRIBUTE and VOTE; DISTRIBUTE utilizes $\mathcal{M}$-$Converge$, for each honest party to convey its view.

messages $[b_1, s_1], \ldots, [b_w, s_w]$ that are not in the respective p's Extracted sets. Instead of every node using a SEND-ALL to send each v_i (as ChBC would do), all nodes run $\mathcal{M}$-CONVERGERANDOM, with their initial constraint sets $\mathcal{C}_p$ empty. $\mathcal{M}$-CONVERGERANDOM assures that all parties eventually see the other parties' inputs but since there is overlap between input messages, it uses less communication.

2. (Voting) An honest party p checks which valid r-batches in their local set correspond to

pairs $(b, s) : b \notin \mathsf{Voted}_p^s$. For each such pair, they check whether they are members of the respective committee via $\mathcal{F}_{\mathsf{mine}}$. If they are, they add their own signature to extend the valid r-batch to a valid $(r+1)$-batch.

Lemma 13 (Parallel Consistency)**.** *Let $R = 2\kappa/\epsilon$. TrustedPBB satisfies parallel consistency, per Definition 3, in the $(\mathcal{F}_{\mathsf{mine}}, \mathcal{F}_{\mathsf{prop}})$-hybrid world.*

Lemma 14 (Parallel Soundness)**.** *Let $R = 2\kappa/\epsilon$. TrustedPBB satisfies parallel soundness, per Definition 3, in the $(\mathcal{F}_{\mathsf{mine}}, \mathcal{F}_{\mathsf{prop}})$-hybrid world.*

Theorem 4 (Communication)**.** *Let $R = 2\kappa/\epsilon$. The total number of bits exchanged by all parties in TrustedPBB is $\tilde{O}(n^2 \cdot \kappa^4)$ with probability $1 - \mathsf{negl}(\kappa)$.*

Chapter 4: Sublinear-Round Broadcast without Trusted Setup

While we stay in the topic of Broadcast, we now focus on improving the *round* complexity of Broadcast protocols. This chapter will follow us in our attempt to emulate standard committee election with VRFs, but in a setting where we do not have trusted setup. Our goal will be to allow parties to elect a random, unbiased committee by sharing randomly generated values with each other. The general committee election will be covered in Section 4.1. Moreover, we need to guarantee that all honest parties will agree on the committee members; this becomes additionally hard when we consider that parties do not have access to a Broadcast primitive (since this is what we want to construct). In Section 4.2 we show how to overcome this via (Moderated) Gradecast and achieve a graded committee election across all parties.

Finally, in Section 4.3 we describe in detail our Broadcast protocol, which we construct by combining our graded committee election with ideas from other Broadcast protocols that utilized committees.

4.1 Algorithms for Committee Election

In this section, we construct the cryptographic core of our sublinear round Broadcast protocol, namely, algorithms to run an unbiased committee election. We will specify two security games and show that these games allow for the adversary only a negligible probability of winning

them. Intuitively, they model that—assuming that the outputs of the algorithms are distributed consistently—the resulting committee contains at least one honest party and is not too large.

4.1.1 Formal Definition

We define our committee election scheme CES by the following algorithms:

- $\mathsf{CES.Setup}(1^\kappa) \to \mathsf{par}$:
  1. $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\kappa), \;\; h \leftarrow\!\!\$\, \mathbb{G}$
  2. $\mathsf{tlpar} \leftarrow \mathsf{TLP.Setup}(1^\kappa)$
  3. $\mathsf{crs} \leftarrow \mathsf{PS.Setup}(1^\kappa), \;\; \tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{PS}}.\mathsf{Setup}(1^\kappa)$
  4. $\mathsf{par} := (\mathbb{G}, g, p, h, \mathsf{tlpar}, \mathsf{crs}, \tilde{\mathsf{crs}})$
- $\mathsf{CES.Gen}(\mathsf{par}) \to (\mathsf{pk}_i, \mathsf{sk}_i)$:
  1. $x_i \leftarrow\!\!\$\, \mathbb{Z}_p, \;\; r_i \leftarrow\!\!\$\, \mathbb{Z}_p, \;\; X_i := g^{x_i} h^{r_i}$
  2. $\pi_i \leftarrow \mathsf{PS.Prove}(\mathsf{crs}, X_i, (x_i, r_i))$
  3. $\mathsf{pk}_i := (X_i, \pi_i), \;\; \mathsf{sk}_i := (x_i, r_i)$
- $\mathsf{CES.Prepare}(\mathsf{par}) \to \mathsf{pretags}_i$:
  1. For all $j \in [n]$: $T_{i,j} \leftarrow\!\!\$\, \mathbb{Z}_p, \;\; \gamma_{i,j} \leftarrow\!\!\$\, \mathbb{Z}_p, \;\; c_{i,j} := g^{T_{i,j}} h^{\gamma_{i,j}}$
  2. Sample random coins ρ_i for algorithm $\mathsf{TLP.Gen}$
  3. $Z_i := \mathsf{TLP.Gen}(\mathsf{tlpar}, (T_{i,j}, \gamma_{i,j})_{j=1}^n; \rho_i)$
  4. $\tilde{\pi}_i \leftarrow \tilde{\mathsf{PS}}.\mathsf{Prove}(\tilde{\mathsf{crs}}, (Z_i, (c_{i,j})_{j=1}^n), ((T_{i,j}, \gamma_{i,j})_{j=1}^n, \rho_i))$
  5. $\mathsf{pretags}_i := (Z_i, (c_{i,j})_{j=1}^n, \tilde{\pi}_i)$
- $\mathsf{CES.TryElect}(\mathsf{par}, \mathsf{sk}_k, (\mathsf{pretags}_i)_{i=1}^n) \to (\mathsf{res}, \mathsf{ticket}_k)$:
  1. Parse $\mathsf{sk}_k := (x_k, r_k)$

2. For all $i \in [n]$: Parse $\mathsf{pretags}_i = (Z_i, (c_{i,j})_{j=1}^n, \tilde{\pi}_i)$
3. $\mathsf{ValidPre} := \{i \in [n] \mid \tilde{\mathsf{PS}}.\mathsf{Ver}(\tilde{\mathsf{crs}}, (Z_i, (c_{i,j})_{j=1}^n), \tilde{\pi}_i) = 1\}$
4. $((T_{i,j}, \gamma_{i,j})_{j=1}^n)_{i \in \mathsf{ValidPre}} := \mathsf{TLP}.\mathsf{Solve}(\mathsf{tlpar}, (Z_i)_{i \in \mathsf{ValidPre}})$
5. $\mathsf{id}_k := x_k + \sum_{i \in \mathsf{ValidPre}} T_{i,k}$
6. $\mathsf{ticket}_k := (x_k, r_k, (T_{i,k}, \gamma_{i,k})_{i \in \mathsf{ValidPre}})$
7. If $\mathsf{Pred}(\mathsf{id}_k) = 0$: return $(0, \perp)$
8. Else: return $(1, \mathsf{ticket}_k)$

- $\mathsf{CES}.\mathsf{VerElect}(\mathsf{par}, \mathsf{pk}_k, (\mathsf{pretags}_i)_{i=1}^n, \mathsf{ticket}_k) \to \mathsf{res}$:
  1. Parse $\mathsf{pk}_k = (X_k, \pi_k)$. If $\mathsf{PS}.\mathsf{Ver}(\mathsf{crs}, X_k, \pi_k) = 0$: return 0
  2. For all $i \in [n]$: Parse $\mathsf{pretags}_i = (Z_i, (c_{i,j})_{j=1}^n, \tilde{\pi}_i)$
  3. $\mathsf{ValidPre} := \{i \in [n] \mid \tilde{\mathsf{PS}}.\mathsf{Ver}(\tilde{\mathsf{crs}}, (Z_i, (c_{i,j})_{j=1}^n), \tilde{\pi}_i) = 1\}$
  4. Parse $\mathsf{ticket}_k = (x_k, r_k, (T_{i,k}, \gamma_{i,k})_{i \in \mathsf{ValidPre}})$
  5. If $g^{x_k} h^{r_k} \neq X_k$: return 0
  6. If there is an $i \in \mathsf{ValidPre}$ such that $g^{T_{i,k}} h^{\gamma_{i,k}} \neq c_{i,k}$: return 0
  7. $\mathsf{id}_k := x_k + \sum_{i \in \mathsf{ValidPre}} T_{i,k}$
  8. Return $\mathsf{res} := \mathsf{Pred}(\mathsf{id}_k)$

4.1.2 Informal Description

We now provide an informal explanation of how we expect the algorithms of our committee election scheme to operate.

Setup. We assume parties have access to a common set of system parameters par generated by an algorithm CES.Setup. These include the description of a cyclic group $\mathbb{G}$ with generator g

and prime order p, namely, $(\mathbb{G}, g, p) \leftarrow \mathsf{GGen}(1^\kappa)$. In addition, the parameters contain a group element $h \leftarrow\!\!\$\, \mathbb{G}$, time-lock parameters $\mathsf{tlpar} \leftarrow \mathsf{TLP.Setup}(1^\kappa)$, as well as common reference strings $\mathsf{crs} \leftarrow \mathsf{PS.Setup}(1^\kappa)$ and $\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{PS}}.\mathsf{Setup}(1^\kappa)$ as public parameters. Here, PS is a proof system for relation

$$\mathcal{R} := \{(X, (x, r)) \mid X = g^x h^r\},$$

and $\tilde{\mathsf{PS}}$ is a proof system for the relation

$$\tilde{\mathcal{R}} := \left\{ ((Z, (c_j)_{j=1}^n), ((T_j, \gamma_j)_{j=1}^n, \rho)) \;\middle|\; \begin{array}{r} Z = \mathsf{TLP.Gen}(\mathsf{tlpar}, (T_j, \gamma_j)_{j=1}^n; \rho) \\ \wedge \quad \forall j \in [n] : c_j = g^{T_j} h^{\gamma_j} \end{array} \right\}.$$

Looking ahead, with PS parties will prove that their keys are well-formed, and with $\tilde{\mathsf{PS}}$ parties prove that their time-lock puzzles are well-formed. We will elaborate on the rationale for defining the relation $\tilde{\mathcal{R}}$ like this below.

Predicates. We utilize a predicate $\mathsf{Pred}\colon \mathbb{Z}_p \to \{0, 1\}$, the instantiation of which we leave for the end of this section. For convenience, we define the *bias* of the predicate and a bit $b \in \{0, 1\}$ as

$$\mathsf{bias}(\mathsf{Pred}, b) := \Pr\left[\mathsf{Pred}(\nu) = b \mid \nu \leftarrow\!\!\$\, \mathbb{Z}_p\right].$$

We also define the *tail* of the predicate as

$$\mathsf{tail}(\mathsf{Pred}, n, B) := \Pr\left[|\{j \in [n] \mid \mathsf{Pred}(\nu_j) = 1\}| > B \mid \nu_1, \ldots, \nu_n \leftarrow\!\!\$\, \mathbb{Z}_p\right].$$

That is, the tail denotes the probability that among n uniform and independent values, more than B of them satisfy the predicate.

Key Generation. Before running the committee election, each party P_i generates a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ via a key generation algorithm $\mathsf{CES.Gen}$. This is done as follows: the secret key sk_i contains two random exponents $x_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and $r_i \leftarrow\!\!\$\, \mathbb{Z}_p$. The public key pk_i consists of a

commitment $X_i := g^{x_i} h^{r_i}$ and a proof of well-formedness $\pi_i \leftarrow \mathsf{PS.Prove}(\mathsf{crs}, X_i, (x_i, r_i))$.

Sampling Tags. To prevent the election from being biased by the adversary, all parties jointly generate randomness. This is done using what we call tags and pretags. In more detail, we assume that each party P_i generate $\mathsf{pretags}_i$ via an algorithm $\mathsf{CES.Prepare}$. The set of all these $\mathsf{pretags}_i$ will later help determine who is in the committee.

Let us now explain how $\mathsf{pretags}_i$ is generated: party P_i locally samples a list of tags $T_{i,j} \leftarrow\!\!\$\, \mathbb{Z}_p$, one tag for each other party P_j, $j \in [n]$. Then, P_i commits to these tags by sampling $\gamma_{i,j} \leftarrow\!\!\$\, \mathbb{Z}_p$ for all $j \in [n]$ and setting $c_{i,j} := g^{T_{i,j}} h^{\gamma_{i,j}}$, and sets $\mathsf{pretags}_i$ to consist of the commitments and time-lock puzzles of the tags. More concretely, P_i samples random coins ρ_i for the algorithm $\mathsf{TLP.Gen}$. It then computes a puzzle $Z_i := \mathsf{TLP.Gen}(\mathsf{tlpar}, (T_{i,j}, \gamma_{i,j})_{j=1}^n; \rho_i)$ and a proof $\tilde{\pi}_i \leftarrow \tilde{\mathsf{PS}}.\mathsf{Prove}(\tilde{\mathsf{crs}}, (Z_i, (c_{i,j})_{j=1}^n), ((T_{i,j}, \gamma_{i,j})_{j=1}^n, \rho_i))$. Recall that $\tilde{\pi}_i$ proves that Z_i is a valid puzzle for the tags $T_{i,j}$ that are committed in the commitments $c_{i,j}$. That is, the proof links the puzzles to the commitments. Intuitively, to ensure that malicious parties cannot make their tags depend on other tags, we will make sure that all parties output their $\mathsf{pretags}_i$ within a certain time interval that is related to the hardness of the puzzles.

Winning the election. After all pretags are distributed, each party P_k will be associated with a random value

$$\mathsf{id}_k = x_k + \sum_i T_{i,k}.$$

If a party provides an invalid proof, all tags provided by that party are treated as zero. Any tag that is included to the sum, must have consistent commitments and puzzles respectively (by the soundness of $\tilde{\mathsf{PS}}$). A party P_k with access to the list $(\mathsf{pretags}_i)_{i=1}^n$ can run algorithm $\mathsf{CES.TryElect}$ to evaluate if it has been elected into the committee: P_k batch solves all puzzles with valid proofs

$\tilde{\pi}_i$, receiving the tags $T_{i,k}$ and the corresponding $\gamma_{i,k}$. Then, P_k computes the value id_k as above; Finally, it wins the election if and only if $\mathsf{Pred}(\mathsf{id}_k) = 1$.

Convincing Others of Committee Membership. Say that a party P_k with public key $\mathsf{pk}_j = (X_k, \pi_k)$ holding the list $(\mathsf{pretags}_i)_{i=1}^n$ wants to convince a second party P_l that it has been elected to the committee. First of all, if π_k is invalid, P_l will reject immediately and not consider P_k as a committee member. Otherwise, P_l expects P_k to provide its secret key $\mathsf{sk}_k = (x_k, r_k)$. Party P_l then checks that (x_k, r_k) is indeed consistent with the commitment X_k, and that $\mathsf{Pred}(\mathsf{id}_k) = 1$. Needless to say, to compute id_k, party P_l also needs the tags $T_{i,k}$ (for all i for which $\tilde{\pi}_i$ is valid).

Note that, if we assume that parties have a consistent view of the $(\mathsf{pretags}_i)_{i=1}^n$ and of pk_k, then by solving the puzzles for itself, P_l learns the tags $T_{i,k}$ and does not need P_k to provide its solved tags as proof of committee membership. However, since we need to avoid this strong assumption when constructing our protocol in the subsequent sections, we also weaken the setup here. In particular, we allow the possibility that corrupted parties provide different pretags to different honest parties. We will go into details about how to solve this in Section 4.2, but for the moment, this translates into the following. In order to verify that a party is in the committee, the verifying party P_l needs to know the tags (and the secret key) for each such party P_k, and so P_l would need to resolve the puzzles contained in the pretags for every party P_k that is claiming to be in the committee. This naive solution would require each verifying party to solve $\Theta(n)$ batch puzzles, which is too costly. Therefore, our goal is to avoid this. To do so, we rely on the soundness of $\tilde{\mathsf{PS}}$: we know that the output of a puzzle Z_i is consistent with the commitments $(c_{i,j})_{i=1}^n$. Therefore, we let P_k provide to P_l the tags $T_{i,k}$ along with their openings $\gamma_{i,k}$. Party P_l can then check the correctness of the tags by comparing its local view on $c_{i,k}$ with the provided

$T_{i,k}$ and $\gamma_{i,k}$. To summarize, party P_k convinces party P_l of its membership in the committee by providing ticket_k, which contains x_k, r_k and $T_{i,k}, \gamma_{i,k}$ for all indices i which have valid proofs $\tilde{\pi}_i$. We model verifying such a claim and ticket_k by algorithm $\mathsf{CES.VerElect}$.

4.1.3 Committee Corruption Game

The first security property that we show informally states that no adversary can corrupt all members of the committee. The game is strong in a sense that we allow the adversary to select *all* tags. Also, for each honest party j, we allow the adversary to choose an entirely different set of pretags $(\mathsf{pretags}_i^{(j)})_i$. These are then used (1) by honest party j to check if it is in the committee, and (2) by other honest parties to verify if party j is in the committee. To avoid clutter, the state of $\mathcal{A}$ is kept implicit in the game. Formally, the *committee corruption game* for a number of parties n and a corruption threshold t is defined as follows:

1. **Setup.** The game runs $\mathsf{par} \leftarrow \mathsf{CES.Setup}(1^\kappa)$.
2. **Initialization of Parties.** In the initialization phase, the adversary can declare its initial set of corrupted parties and learns the public keys of honest parties. More precisely:
   (a) The game runs $\mathcal{A}$ on input par. Then, $\mathcal{A}$ declares a set $\mathsf{Corr}_0 \subseteq [n]$. If $|\mathsf{Corr}_0| > t$, the game terminates and outputs 0.
   (b) The game sets $\mathsf{Corr} := \mathsf{Corr}_0$ and $\mathsf{Hon}_0 := [n] \setminus \mathsf{Corr}_0$.
   (c) For each $i \in \mathsf{Hon}_0$, the game runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CES.Gen}(\mathsf{par})$.
   (d) The game gives all pk_i for $i \in \mathsf{Hon}_0$ to $\mathcal{A}$.
   (e) At any time from this point on, $\mathcal{A}$ gets access to a corruption oracle. On input $i \in [n]$, if $i \in \mathsf{Corr}$ or if $|\mathsf{Corr}| = t$, then the oracle outputs $\perp$. Otherwise, the oracle adds i to

Corr, and outputs sk_i.

3. **Tag Phase.** The game obtains $\mathsf{pretags}_i^{(k)}$ for all $i \in [n]$ and $k \in [n]$ from $\mathcal{A}$.

4. **Winning Condition.** The game ends as soon as $|\mathsf{Corr}| = t$. The adversary wins if none of the remaining honest parties is elected:

   (a) Let $\mathsf{Hon}^* := [n] \setminus \mathsf{Corr}$ be the set of $n - t$ remaining honest parties.

   (b) For every $k \in \mathsf{Hon}^*$, the game runs $(\mathsf{res}_k, \mathsf{ticket}_k) \leftarrow \mathsf{CES.TryElect}(\mathsf{par}, \mathsf{sk}_k, (\mathsf{pretags}_i^{(k)})_{i=1}^n)$.

   (c) The game defines the set of honest committee members as

   $$\mathsf{HonComm} := \left\{ k \in \mathsf{Hon}^* \mid \mathsf{res}_k = 1 \wedge \mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k, (\mathsf{pretags}_i^{(k)})_{i=1}^n, \mathsf{ticket}_k) = 1 \right\}.$$

   (d) The game outputs 1 if $\mathsf{HonComm} = \emptyset$. Otherwise, it outputs 0.

Next, we show that no efficient adversary can win the committee corruption game for the protocol described above. That is, no efficient algorithm can corrupt all parties of the committee. We do not need any assumptions on the time-lock puzzles here.

Lemma 15. *Assume that* PS *is zero-knowledge (see Definition 14), that* $\tilde{\mathsf{PS}}$ *is sound (see Definition 13), and that* $p > \omega(\log \kappa)$. *Let* $\mathcal{A}$ *be a PPT algorithm in the committee corruption game for a number of parties* n *and a corruption threshold* t. *Then, the probability that the committee corruption game outputs* 1 *is at most* $\mathsf{negl}(\kappa) + \mathsf{bias}(P, 0)^{n-t}$.

4.1.4 Large Committee Game

The second property we prove is that the adversary cannot make the committee too large. This holds even if the adversary controls all parties and can choose tags for each party independently. Importantly, however, one of the tags for each party is chosen honestly by the game. We

can for now think of this tag as being the tag output by the party that later verifies committee membership. Again, the state of $\mathcal{A}$ is kept implicit in the description of the game. More formally, the *large committee game* for a number of parties n and a committee bound B is defined as follows:

1. **Setup.** The game runs $\mathsf{par} \leftarrow \mathsf{CES.Setup}(1^\kappa)$.
2. **Initialization of Parties.** The adversary declares all keys. More precisely:
   (a) The game runs $\mathcal{A}$ on input par.
   (b) Then, $\mathcal{A}$ outputs pk_k for all $k \in [n]$.
3. **Tag Phase.** In the tag phase, the tags are chosen:
   (a) The game runs $\mathsf{pretags}_1 \leftarrow \mathsf{CES.Prepare}(\mathsf{par})$ and sets $\mathsf{pretags}_{1,k} := \mathsf{pretags}_1$ for all $k \in [n]$.
   (b) The game continues running $\mathcal{A}$ on input $\mathsf{pretags}_1$.
   (c) The game obtains $\mathsf{pretags}_{i,k}$ for all $i \in [n] \setminus \{1\}$ and all $k \in [n]$ from $\mathcal{A}$. If the time between this output and the previous step is larger than Δ', the game terminates and outputs 0.
4. **Winning Condition.** The adversary wins if too many parties are elected. More precisely:
   (a) The game runs $\mathcal{A}$ and gets a set $\mathsf{Committee} \subseteq [n]$ and ticket_k for all $k \in \mathsf{Committee}$.
   (b) The game outputs 1 if the following two conditions hold. Otherwise, it outputs 0:
      i. We have $|\mathsf{Committee}| > B$.
      ii. For all $k \in \mathsf{Committee}$, we have $\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k, (\mathsf{pretags}_{i,k})_{i=1}^n, \mathsf{ticket}_k) = 1$.

We now show that no efficient adversary can win the large committee game, i.e., the committee is never too large.

Lemma 16. *Assume that* PS *and* $\tilde{\mathsf{PS}}$ *are simulation-extractable (see Definition 15), assume that the discrete logarithm assumption holds relative to* GGen*, and assume that* TLP *is* Δ*-CPA secure, where* Δ *is sufficiently larger than* $\mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Sim}) + \Delta' + n \cdot \mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Ext})$*. Let* $\mathcal{A}$ *be a PPT algorithm in the large committee game for a number of parties* $n < p$ *and a committee bound* B*. Then, the probability that the large committee game outputs* 1 *is at most* $\mathsf{negl}(\kappa) + \mathsf{tail}(P, n, B)$*.*

Remark 1. *For conciseness, from now on we hide the terms depending on the proof system and on computations independent of the time-lock puzzle by using the notation*

$$\Delta(\Delta') := \mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Sim}) + \Delta' + n \cdot \mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Ext}) + t, \tag{4.1}$$

where t *is the negligible amount of time it takes to define the sets* $\mathsf{ValidPre}_k$ *and* Copied_k*, and to compute the size of* $\mathsf{Committee}^*$ *given all relevant* $\hat{x}_k$ *and* $\hat{T}_{i,k,k}$*, as in the proof of Lemma 16. In particular, this shifts the focus on needing to define* Δ'*, which is the time by which the adversary needs to submit all its pretags.*

We also note that $\mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Sim})$ and $\mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Ext})$ are independent of a round duration Δ_r. The same holds for the additional time t above. As such, we have that $(\Delta(\Delta') - \Delta')/\Delta_r = O(1)$, i.e., the extra time apart from Δ' in (4.1) takes a constant number of rounds. We select Δ' to take a number of rounds that is sublinear in n, such that an honest party solving a TLP with difficulty $\Delta(\Delta')$ will be able to obtain the solution in a sublinear number of rounds.

4.1.5 Predicate Instantiation

We now instantiate the predicate $\mathsf{Pred}\colon \mathbb{Z}_p \to \{0,1\}$ and the committee bound B. To use the previous two lemmata in a meaningful way, we need that $\mathsf{tail}(\mathsf{Pred}, n, B)$ and $\mathsf{bias}(\mathsf{Pred}, 0)^{n-t}$

are negligible in λ. We let $\mathsf{Pred}(\nu)$ output 1 if and only if $\nu < \mathsf{bound}_{\epsilon,\delta}$, where

$$p_{\text{mine}} := \min\left\{1, \frac{1}{\epsilon n}\log\left(\frac{1}{\delta}\right)\right\}, \quad \mathsf{bound}_{\epsilon,\delta} := p_{\text{mine}} \cdot p.$$

Here, p denotes the size of the field $\mathbb{Z}_p$, p_{mine} roughly corresponds to the probability of a party being elected, ϵ is a constant in $(0,1)$ denoting the fraction of honest parties (i.e. $t = (1-\epsilon)n$), and δ is a failure probability. Below, we will focus on the case where $\delta = \exp(-\omega(\log\lambda))$ and $\log(\delta^{-1}) \leq \epsilon n$.[1] Further, we define the committee bound B as

$$B := \left\lceil \frac{3}{\epsilon}\log\left(\frac{1}{\delta}\right)\right\rceil = O(\lambda/\epsilon).$$

For this choice of parameters, we show that $\mathsf{tail}(P, n, B)$ and $\mathsf{bias}(P, 0)^{n-t}$ are both negligible in the security parameter.

Lemma 17. *Consider the predicate* Pred *defined as above. Then, it holds that*

$$\mathsf{bias}(\mathsf{Pred}, 0)^{n-t} \leq \delta \leq \mathsf{negl}(\lambda).$$

Proof. By definition, we have $\mathsf{bias}(\mathsf{Pred}, 0) = 1 - p_{\text{mine}}$. If $p_{\text{mine}} = 1$, then $\mathsf{bias}(\mathsf{Pred}, 0) = 0$, so the statement holds trivially. We now focus on the other case. From Bernoulli's inequality (Lemma 36), we get

$$\mathsf{bias}(\mathsf{Pred}, 0)^{n-t} = (1 - p_{\text{mine}})^{\epsilon \cdot n} \leq \exp(-\epsilon n \cdot p_{\text{mine}}) = \exp(-\log(\delta^{-1})) = \delta,$$

where the first inequality follows from Bernoulli's inequality. □

Lemma 18. *Consider the predicate* Pred *and the committee bound* B *as above. Then, it holds that*

$$\mathsf{tail}(\mathsf{Pred}, n, B) \leq \delta \leq \mathsf{negl}(\lambda).$$

[1] I.e., pick δ such that $\delta \geq \exp(-\epsilon n)$ and $\delta = \exp(-\omega(\log\lambda))$, which can be done if $\epsilon n > \log\lambda$.

Proof. Let $X_i, i \in [n]$ be Bernoulli random variables, denoting the event that $\mathsf{Pred}(\mathsf{id}_i) = 1$. Since id_i are chosen independently and uniformly, the X_i's are independent and identically distributed with $\Pr[X_i = 1] = p_{\text{mine}}$. In that notation, $\mathsf{tail}(\mathsf{Pred}, n, B) = \Pr[\sum_{i\in[n]} X_i > B]$. The expected value of $X := \sum_{i\in[n]} X_i$ is $\mathbb{E}[X] = n \cdot p_{\text{mine}}$.

$$\mathbb{E}[X] = n \cdot p_{\text{mine}} = n \cdot \min\left\{1, \frac{1}{\epsilon n}\log\left(\delta^{-1}\right)\right\}$$
$$= \begin{cases} n & \text{if } \log\left(\delta^{-1}\right) > \epsilon n \\ \frac{1}{\epsilon}\log\left(\delta^{-1}\right) & \text{if } \log\left(\delta^{-1}\right) \leq \epsilon n \end{cases}.$$

The first case is not interesting, since it corresponds to the case where the committee is as large as the total number of parties. This motivates our assumption of δ above, namely $\log(\delta^{-1}) \leq \epsilon n$, so we are in the second case. We continue with Chernoff's inequality (Lemma 37).

$$\Pr[X > (1+\zeta)\cdot\mathbb{E}[X]] \leq \exp\left(-\frac{\zeta^2\cdot\mathbb{E}[X]}{2+\zeta}\right) = \exp\left(-\frac{\zeta^2}{2+\zeta}\cdot\frac{1}{\epsilon}\cdot\log\left(\frac{1}{\delta}\right)\right)$$
$$\overset{*}{\leq} \exp\left(-\log\left(\delta^{-1}\right)\right) = \delta,$$

where $*$ holds if

$$\frac{\zeta^2}{2+\zeta}\cdot\frac{1}{\epsilon} \geq 1. \tag{4.2}$$

We want to pick a ζ that satisfies (4.2) for any value of $\epsilon \in (0,1)$ and from it, define $B_0 := (1+\zeta)\cdot\mathbb{E}[X]$, such that $\Pr[X \geq B_0] \leq \delta$.

The roots of (4.2) are $(\epsilon - \sqrt{\epsilon^2+8\epsilon})/2$ and $(\epsilon + \sqrt{\epsilon^2+8\epsilon})/2$ and inequality holds outside of the roots. Since $\zeta \geq 0$, we only need $\zeta \geq (\epsilon + \sqrt{\epsilon^2+8\epsilon})/2$. To account for any value of

$\epsilon \in (0,1)$, we choose to set

$$B_0 = \left(1 + \frac{\epsilon + \sqrt{\epsilon^2 + 8\epsilon}}{2}\right) \cdot \mathbb{E}[X] = \frac{2 + \epsilon + \sqrt{\epsilon^2 + 8\epsilon}}{2} \cdot \frac{1}{\epsilon} \cdot \log\left(\frac{1}{\delta}\right)$$
$$= \frac{2 + \epsilon + \sqrt{\epsilon^2 + 8\epsilon}}{2\epsilon} \cdot \log\left(\frac{1}{\delta}\right) \leq \frac{6}{2\epsilon} \cdot \log\left(\frac{1}{\delta}\right) = \frac{3}{\epsilon} \cdot \log\left(\frac{1}{\delta}\right).$$

Since it holds that for $B \geq B_0$, $\Pr[X > B_0] \leq \Pr[X > B] \leq \delta$, we can choose an integer B from the maximum value of B_0. Thus, $B = \lceil 3 \log(\delta^{-1})/\epsilon \rceil$, the value of B in the statement. □

4.2 Graded Committee Election

In this section, we introduce a committee election scheme that also accounts for inconsistent views between parties. This committee election scheme will be composed of both algorithms and protocols. The main difficulty of adapting the algorithms from Section 4.1 into protocols stems from the inconsistent views of the committee among different honest parties. Recall that a party can decide if it is in the committee based on an "identity", composed of its secret key and a set of so-called pretags. Other parties can then be convinced of the committee membership using a ticket provided by the party claiming membership. However, in Section 4.1, we assumed that the winning party and the verifying party agree on the set of pretags and on the winning party's public key. In this section, we show how to achieve this. The challenge in doing so is to deal with inconsistencies and to ensure that the process of distributing keys and pretags terminates.

Informally, in our proposed solution, parties will share their pretags and keys via several applications of a gradecast protocol. Recall that at the end of a gradecast protocol, parties hold an output they think corresponds to the sender's input as well as a grade associated to this output. Importantly, the grades that honest parties hold at the end of gradecast will quantify how much confidence they have that their outputs coincide. In particular, grades greater than 1 certify that

honest parties have the same view on the input pretags and hence, on the identities used for election.

Throughout, we call our solution a graded committee election scheme (GCES). It consists of the GCES.Gen protocol in Figure 4.1, which is a gradecast on the public keys, and the GCES.Toss protocol in Figure 4.2, which can be seen as a parallel moderated gradecast on the pretags. The final part of the graded committee election consists parties locally running CES.TryElect and CES.VerElect on appropriate inputs.

4.2.1 Graded Consensus on Pretags and Keys

We give a step-by-step description of the protocols, beginning with a strawman construction.

Strawman Construction. In this strawman construction, parties engage in a parallel gradecast protocol where each party inputs their pretags. At the end of the gradecast protocol, each party P_i outputs for each party P_j a pretag and a grade $(\mathsf{pretags}_j^{(i)}, g_j^{(i)})$. Here, we use the superscript to denote the receiving party and the subscript to denote the sending party. This is sufficient for party P_i to be able to run CES.TryElect. As a result, P_i may hold an evidence of election in the form of ticket_i. Let us now explore how other parties can verify that P_i is part of the committee. Party P_i will multicast its ticket. A party P_j will retrieve from memory its result of the parallel gradecasts and check if the pretags it has corresponding to P_i are also consistent with the ticket multicast by P_i. In other words, it would run CES.VerElect, ignoring for now the input corresponding to P_i's public key. To see why this strawman solution fails, assume that both P_i and P_j are honest and there is a malicious party P_k not following the protocol. If there is any pretag from P_k for which, at the end of the gradecast, P_i outputs grade 1 and P_j outputs grade 0, then P_j will not agree

with the committee election claim of P_i. In other words, when using a single instance of parallel gradecasts, the adversary has full power in causing honest parties to disagree in their views of the committee membership.

Distributing Pretags using Moderated Gradecast. To address this, we update the strawman construction with a second step of parallel gradecasts where parties *moderate* the values they received in the first step. The final set of pretags of a moderator P_j from the point of view of a party P_i will be thus made up of pretags that P_j gradecast itself in the first step, but also of pretags P_j gradecasts from other parties in the second step. Moreover, moderators will have their final grade penalized by the difference in the outputs of their moderated gradecast and the initial gradecast. This ensures that a moderator who honestly gradecasts a value sent by a malicious initial sender will not be penalized by more than 1 in its final grade. Thus, malicious parties with positive grades cannot arbitrarily cause two honest parties to disagree on their views of the committee membership. We call this protocol GCES.Toss and detail it in Figure 4.2.

Distributing Keys using Gradecast. The verification of election outcome does not only depend on the pretags and on the ticket of a party claiming membership, but also on a public key shared by that party in advance of sharing the pretags, as specified in CES.VerElect. Therefore, honest parties need to also account for inconsistencies in their views of the adversary's public keys, which are critical in validating the result of the election. To this end, we specify that parties have to also run a gradecast protocol for their public keys as an initial step, that we call GCES.Gen, see Figure 4.1. Note that here, a single gradecast is sufficient for sharing the public keys, since each public key describes a quantity originating from a single party and is used only in the election (and verification) of *that* party. In contrast, the pretags from each party are used to determine the

election of each *other* party, i.e., parties are elected based on an aggregated string that depends on quantities originating from all parties, and, as described previously, require more steps to build trust in the output. For both GCES.Gen and GCES.Toss, parties first read the system parameters $\mathsf{par} \leftarrow \mathsf{CES.Setup}(1^\kappa)$. In our instantiation, these parameters are transparent.

Graded Election and Verification. After a party P_i runs GCES.Gen and (the first two steps of) GCES.Toss, it can determine if it is part of the committee in the following manner. In particular, P_i will have the secret key sk_i and $(\mathsf{pretags}_{j,i}^{(i)})_{j\in[n]} =: (\mathsf{pretags}_j^{(i)})_{j\in[n]}$, where for pretags the notation is as follows: the superscript denotes the receiving party P_i, and the subscripts denote the original sender P_j and the moderator P_i in this order (or only the original sender P_j when there is a single subscript). Then, each party P_i will run CES.TryElect from Section 4.1 on input $(\mathsf{par}, \mathsf{sk}_i, (\mathsf{pretags}_j^{(i)})_{j\in[n]})$ and will output $(\mathsf{res}, \mathsf{ticket})$, where if $\mathsf{res} = 0$, then $\mathsf{ticket} = \perp$, else $\mathsf{res} = 1$ and ticket contains the evidence of election.

Note that the results from GCES.Gen and GCES.Toss are not necessarily identical for honest parties with respect to the party they want to verify membership for. To address this, a ticket containing the secret key and the tags obtained from solving the puzzle need to be multicast by each party claiming election.[2]

Here, multicasting the ticket is sufficient (in contrast to requiring gradecast), because honest parties already have the necessary commitments and proofs which they can use to validate the received ticket. Therefore, parties use CES.VerElect from Section 4.1 to verify the election result claimed by another party. Honest party P_i will have the graded out-

[2]As mentioned in Section 4.1, we aim for each party to only have to solve a single batch puzzle, namely, its own (done in CES.TryElect). Otherwise, given that solving a batch puzzle is a sequential action, evaluating n distinct batch puzzles would automatically refute the claim of a sublinear number of rounds. Multicasting the ticket which contains the solved puzzle thus resolves this issue.

puts of GCES.Gen and GCES.Toss, and will be able to check the validity of P_j's claimed membership upon receiving $\mathsf{ticket}_j^{(i)}$ from P_j. Concretely, P_i will run CES.VerElect on input $(\mathsf{par}, \mathsf{pk}_j^{(i)}, (\mathsf{pretags}_{k,j}^{(i)})_{k=1}^n, \mathsf{ticket}_j^{(i)})$ and output a bit $\mathsf{res} \in \{0,1\}$, where $\mathsf{res} = 1$ if P_j is a member of the committee from P_i's perspective and $\mathsf{res} = 0$ otherwise.[3]

GCES.Gen(par)

Step 0: Each party samples their key pair.
P_i runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CES.Gen}(\mathsf{par})$.

Step 1: Each party gradecasts their key.
1. P_i calls $\mathsf{GC}(\mathsf{pk}_i, g^*)$.
2. P_i receives $(\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})$ from the gradecast with sender P_j, where $\mathsf{pk}_j^{(i)}$ is the public key of P_j from the view of P_i and $\tilde{g}_j^{(i)}$ is the associated grade.

Step 3: Each party outputs the public parameters, its local key pair and public keys and associated grades for all other parties.
P_i outputs $\left((\mathsf{pk}_i, \mathsf{sk}_i), (\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})_{j \in [n] \setminus \{i\}}\right)$.

Figure 4.1: The key generation and distribution protocol.

Round Complexity. The number of rounds of GCES.Gen is determined by running n-parallel instances of GC, which is $2g^* + 1$ rounds. GCES.Toss takes $4g^* + 2$ rounds, determined by running n-parallel instances of GC followed by n^2-parallel instances of GC (termination is proved in Section 4.2.2).

Communication Complexity. Let $\mathsf{CC}_{\mathsf{GC}}(\ell, \kappa, g^*)$ denote the total communication complexity of the gradecast protocol for a message size ℓ. The total communication complexity of GCES.Gen is $\mathsf{CC}_{\mathsf{GC}}(\kappa, \kappa, g^*) = O(g^* \cdot \kappa \cdot n^2)$, assuming the size of a signature, a public key X_i and a proof π_i is $O(\kappa)$. Assuming the size of a signature and a commitment (there are n of them) is $O(\kappa)$ and the size of a puzzle on n tags and of the corresponding proof is $O(\kappa n)$, the total communication complexity of GCES.Toss for n parties with inputs of length $O(\kappa n)$ is $n^2 \cdot \mathsf{CC}_{\mathsf{GC}}(\kappa n, \kappa, g^*) = O(g^* \kappa n^5)$. Using

[3]Note that P_i does not need to check here again whether $\mathsf{pretags}_{i,j}^{(i)} = \mathsf{pretags}_i$; we will only be interested in verifying membership claims coming from parties with a positive grade, which means this check at P_i already happened.

GCES.Toss(par)

Step 0: Each party generates their pretags.
$\mathsf{pretags}_i \leftarrow \mathsf{CES.Prepare}(\mathsf{par})$.

Step 1: Each party gradecasts their pretag.
1. P_i calls $\mathsf{GC}(\mathsf{pretags}_i, g^*)$.
2. P_i receives from gradecast from sender P_j $(\mathsf{pretags}_j^{(i)}, g_j^{(i)})$, where $g_j^{(i)}$ is the grade.

Step 2: Each party acts as moderator.
1. For each $j \in [n] \setminus \{i\}$ for which P_i participated in a gradecast with sender P_j, let $m_j^{(i)} := (\mathsf{pretags}_j^{(i)}, \mathsf{Sig}_j(\mathsf{pretags}_j^{(i)}))$.
2. P_i calls $\mathsf{GC}(m_j^{(i)}, g^*)$, for all $j \in [n] \setminus i$.
3. P_i outputs $(m_{k,j}^{(i)}, g_{k,j}^{(i)})$, for each moderator P_j and for each initial sender P_k, $k \in [n] \setminus \{j\}$ where $m_{k,j}^{(i)}$ contains the puzzle, commitments and proof gradecasted by P_j for P_k, and $g_{k,j}^{(i)}$ is the respective grade.[a]

Step 3: Each party decides its output value and grade for each moderator.
1. P_i sets for each moderator P_j and initial sender P_k the pretag: $\mathsf{pretags}_{k,j}^{(i)} = (Z_{k,j}^{(i)}, C_{k,j}^{(i)}, \pi'^{(i)}_{k,j}) \in m_{k,j}^{(i)}$.[b]
2. P_i sets for each moderator P_j and initial sender P_k the grade:

$$G_{k,j}^{(i)} = \begin{cases} g_{k,j}^{(i)} & , \text{ if } \mathsf{pretags}_{k,j}^{(i)} = \mathsf{pretags}_k^{(i)}, \\ \min\{g_{k,j}^{(i)}, g^* - g_k^{(i)}\} & , \text{ otherwise.} \end{cases}$$

3. P_i sets its general grade for each moderator P_j as $G_j^{(i)} := \min_{k\in[n]}\{G_{k,j}^{(i)}\}$ and outputs for each $j \in [n]$: $((\mathsf{pretags}_{k,j}^{(i)})_{k=1}^n, G_j^{(i)})$.

[a] The notation is as follows: the superscript denotes which party receives/observes the value, in this case P_i. The subscripts denotes the order of relays. In this case it is k, j, since moderator P_j relays a value sent by party P_k in the first step.

[b] We use the membership operator here to obscure the signatures that were part of $m_{k,j}^{(i)}$.

Figure 4.2: GCES.Toss protocol (or Parallel Moderated Gradecast).

gossiping for the parallel moderated gradecast as in [35], one can reduce the total communication complexity of GCES.Toss to $\tilde{O}(g^*\kappa n^4)$.

4.2.2 Agreement Properties

Note that GCES.Gen is essentially a parallel gradecast protocol, so for each of the gradecast instances, the properties from Definition 4 are trivially satisfied. We now show that GCES.Toss satisfies some generalized properties of the moderated gradecast protocols, namely *Graded Va-*

lidity, *Graded Agreement* and *Termination*, defined in the subsequent lemmata.

We first start with some helper results. Consider one moderator P_j and one sender P_i running the subprotocol in Figure 4.2 starting from Step 1. We show that this is an instance of a moderated gradecast protocol, which we call Mod-GC(pretags$_i$), which satisfies *Graded Soundness*, *M-Soundness*, *Graded Consistency* and *Termination* against an adaptive adversary controlling $t \leq (1-\epsilon)n$ parties.

Proposition 1. *Let* a_1, b_1, a_2, b_2, g *such that* $|a_1 - a_2| \leq 1$, $|b_1 - b_2| \leq 1$ *and* $g \geq \max\{a_1, a_2, b_1, b_2\}$. *Let* $G_i = \min\{a_i, g - b_i\}$, *for* $i = 1, 2$. *Then,* $|G_1 - G_2| \leq 1$.

Proof. We have the following cases:

Case 1. $a_1 \leq g - b_1$ and $a_2 \leq g - b_2$. Then $|G_1 - G_2| = |a_1 - a_2| \leq 1$.

Case 2. $g - b_1 \leq a_1$ and $g - b_2 \leq a_2$. Then $|G_1 - G_2| = |g - b_1 - (g - b_2)| = |b_2 - b_1| \leq 1$.

Case 3. $g - b_1 \leq a_1$ and $a_2 \leq g - b_2$. Then $|G_1 - G_2| = |g - b_1 - a_2|$. Notice that $a_2 + b_2 \leq g \leq a_1 + b_1$, so $b_2 - b_1 \leq g - b_1 - a_2 \leq a_1 - a_2$. Both the lower and upper bound can take values in $[-1, 1]$, which constrains $|G_1 - G_2| = |g - b_1 - a_2| \leq 1$.

Case 4. $a_1 \leq g - b_1$ and $g - b_2 \leq a_2$. This mirrors case 3. □

Lemma 19. *(Moderated Gradecast) Let* Mod-GC *be the subprotocol in Figure 4.2 from Step 1 to Step 3.2, for a single pair of sender* P_i *and moderator* P_j. *Then,* Mod-GC *is a moderated gradecast protocol as in Definition 5.*

Proof. We show that Mod-GC satisfies graded soundness, M-soundness, graded consistency, and termination.

***Graded Soundness and M-Soundness*:** Let P_m be the honest moderator and let P_j be an honest party. P_m will gradecast the value obtained from sender P_i correctly, thus $g_{i,m}^{(j)} = g^*$. If P_i is

also honest, then $\mathsf{pretags}_i = \mathsf{pretags}_i^{(j)} = \mathsf{pretags}_{i,m}^{(j)}$, which implies $G_{i,m}^{(j)} = g^*$. Thus, all honest parties P_j output the same tuple $(\mathsf{pretags}_i, g^*)$, and graded soundness holds. However, if P_i is dishonest, then it could be that $\mathsf{pretags}_i^{(j)} \neq \mathsf{pretags}_{i,m}^{(j)}$. If that is the case, then P_i gradecasts different values to P_m and P_j in Step 1, so from the consistency of GC we have $g_i^{(j)} \leq 1$. Therefore, honest party P_j has $G_{i,m}^{(j)} = \min\{g_{i,m}^{(j)}, g^* - g_i^{(j)}\} \geq g^* - 1$. Finally, since P_m is honest, all honest parties receive (and output) the same message from P_m, $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_i^{(m)}$, and M-soundness holds.

***Graded Consistency*:** For any moderator P_m and a sender P_i, let P_j, P_k be two honest parties obtaining $(\mathsf{pretags}_{i,m}^{(j)}, G_{i,m}^{(j)})$ and $(\mathsf{pretags}_{i,m}^{(k)}, G_{i,m}^{(k)})$ respectively. We have the following cases:

Case 1. $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_i^{(j)}$ and $\mathsf{pretags}_{i,m}^{(k)} = \mathsf{pretags}_i^{(k)}$. Then, $G_{i,m}^{(j)} = g_{i,m}^{(j)}$ and $G_{i,m}^{(k)} = g_{i,m}^{(k)}$, which originate both from the same gradecast and thus, by the graded consistency of GC, $|G_{i,m}^{(j)} - G_{i,m}^{(k)}| \leq 1$. If both $G_{i,m}^{(j)}, G_{i,m}^{(k)}$ are greater or equal to 1, then by GC graded consistency $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_{i,m}^{(k)}$.

Case 2. $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_i^{(j)}$ and $\mathsf{pretags}_{i,m}^{(k)} \neq \mathsf{pretags}_i^{(k)}$. Then, $G_{i,m}^{(j)} = g_{i,m}^{(j)}$ and $G_{i,m}^{(k)} = \min\{g_{i,m}^{(k)}, g^* - g_i^{(k)}\}$. Also, either $g_{i,m}^{(k)} \leq 1$ or $g_i^{(m)} \leq 1$, since one of the two gradecasts has two honest parties outputting different messages. Therefore,

- if $g_{i,m}^{(k)} \leq 1$, then $G_{i,m}^{(k)} \leq 1$ and by GC graded consistency, $|g_{i,m}^{(j)} - g_{i,m}^{(k)}| \leq 1$. There are two subcases. Subcase 1): $g_{i,m}^{(k)} = G_{i,m}^{(k)}$, which immediately implies $|G_{i,m}^{(j)} - G_{i,m}^{(k)}| \leq 1$. If both $G_{i,m}^{(j)}, G_{i,m}^{(k)}$ are greater or equal to 1, then by the moderator's GC graded consistency $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_{i,m}^{(k)}$. Subcase 2): $g_{i,m}^{(k)} = 1$ and $G_{i,m}^{(k)} = 0$, meaning that $g_i^{(k)} = g^*$. By the initial GC graded consistency, we also have $g_i^{(j)} \in \{g^* - 1, g^*\}$ and $\mathsf{pretags}_i^{(j)} = \mathsf{pretags}_i^{(k)}$. To obtain $|G_{i,m}^{(j)} - G_{i,m}^{(k)}| \leq 1$, we need to show it cannot hold that $g_{i,m}^{(j)} = 2$. If $g_{i,m}^{(j)} = 2$, then

$\mathsf{pretags}_{i,m}^{(k)} = \mathsf{pretags}_{i,k}^{(j)}$, which is also equal to $\mathsf{pretags}_i^{(j)}$, contradicting the assumption that $\mathsf{pretags}_{i,m}^{(k)} \neq \mathsf{pretags}_i^{(k)}$.

- if $g_i^{(k)} \leq 1$, then there are again two subcases. Subcase 1): $g_{i,m}^{(k)} = g^*$, in which case $G_{i,m}^{(k)} \in \{g^* - 1, g^*\}$ and $G_{i,m}^{(j)} \in \{g^* - 1, g^*\}$ and by the moderator's GC graded consistency, $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_{i,m}^{(k)}$. Subcase 2): $g_{i,m}^{(k)} \leq g^* - 1$, in which case $G_{i,m}^{(k)} = g_{i,m}^{(k)}$, and by GC graded consistency, it holds that $|G_{i,m}^{(j)} - G_{i,m}^{(k)}| \leq 1$, as well as that if both $G_{i,m}^{(j)}, G_{i,m}^{(k)}$ are greater or equal to 1, then $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_{i,m}^{(k)}$.

Case 3. $\mathsf{pretags}_{i,m}^{(j)} \neq \mathsf{pretags}_i^{(j)}$ and $\mathsf{pretags}_{i,m}^{(k)} \neq \mathsf{pretags}_i^{(k)}$. Then $G_{i,m}^{(j)} = \min\{g_{i,m}^{(j)}, g^* - g_i^{(j)}\}$ and $G_{i,m}^{(k)} = \min\{g_{i,m}^{(k)}, g^* - g_i^{(k)}\}$, and we can use Proposition 1 to get that $|G_{i,m}^{(j)} - G_{i,m}^{(k)}| \leq 1$. If both $G_{i,m}^{(j)}, G_{i,m}^{(k)}$ are greater or equal to 1, then following the graded consistency of the appropriate GC instance in the four cases in Proposition 1, i.e., the sender's or the moderator's, we obtain $\mathsf{pretags}_{i,m}^{(j)} = \mathsf{pretags}_{i,m}^{(k)}$.

To finish the proof of graded consistency for Mod-GC, we need to consider the case where where $G_{i,m}^{(j)} = 1$ and $\mathsf{pretags}_{i,m}^{(j)} \neq \mathsf{pretags}_{i,m}^{(k)}$. Then, by the graded consistency property of GC for sender P_m gradecasting its value $\mathsf{pretags}_i^{(m)}$, we know that $\mathsf{pretags}_{i,m}^{(j)} \neq \mathsf{pretags}_{i,m}^{(k)}$ implies $g_{i,m}^{(j)} = 0$ or $g_{i,m}^{(k)} = 0$. Assume $g_{i,m}^{(k)} \neq 0$, then $g_{i,m}^{(j)} = 0$ which would lead to $G_{i,m}^{(j)} = 0$, contradiction. Thus, $G_{i,m}^{(k)} = 0$ and the proof is complete.

***Termination*:** Each honest party generates a value and an accompanying grade for every moderator, by construction, regardless of the adversary's behaviour. □

We are now ready to prove the properties of the GCES.Toss protocol.

Lemma 20 (Graded Validity of GCES.Toss)**.** *If party P_j is honest, then every honest party P_i outputs $((\mathsf{pretags}_{k,j}^{(i)} := \mathsf{pretags}_k^{(j)})_{k=1}^n, G_j^{(i)})$ for $G_j^{(i)} \in \{g^* - 1, g^*\}$.*

Proof. Since each Mod-GC instance called by an honest moderator P_j achieves M-validity, it holds that for any honest moderator P_j, any honest party P_i outputs $(\mathsf{pretags}_k^{(j)}, G_{k,j}^{(i)})$ for any sender P_k, with $G_{k,j}^{(i)} \in \{g^*, g^* - 1\}$. This implies that all honest parties will also have $G_j^{(i)} \in \{g^*, g^* - 1\}$. Finally, all honest parties form their output strings with all $\mathsf{pretags}_k^{(j)}$ for $k \in [n]$ since all associated grades are strictly positive when the moderator is honest. Again, by M-validity of Mod-GC, another honest party P_m will have $\mathsf{pretags}_{k,j}^{(m)} = \mathsf{pretags}_k^{(j)}$ for all $k \in [n]$, implying the result for any honest moderator P_j. □

Lemma 21 (Graded Agreement of GCES.Toss)**.** *If P_i, P_j are two honest parties outputting $((\mathsf{pretags}_{k,m}^{(i)})_{k=1}^n, G_m^{(i)})$ and $((\mathsf{pretags}_{k,m}^{(j)})_{k=1}^n, G_m^{(j)})$, respectively, for any P_m, then it holds that $|G_m^{(i)} - G_m^{(j)}| \leq 1$. Moreover, if $G_m^{(i)} > 1$, then $\mathsf{pretags}_{k,m}^{(i)} = \mathsf{pretags}_{k,m}^{(j)}$, for all $k \in [n]$, otherwise if $G_m^{(i)} = 1$, then either $\mathsf{pretags}_{k,m}^{(i)} = \mathsf{pretags}_{k,m}^{(j)}$, for all $k \in [n]$, or $G_j^{(m)} = 0$.*

Proof. Let α such that $\min_{k\in[n]}\{G_{k,m}^{(i)}\} = G_{\alpha,m}^{(i)}$. Similarly, let β such that $\min_{k\in[n]}\{G_{k,m}^{(j)}\} = G_{\beta,m}^{(j)}$. Suppose without loss of generality that $G_{\beta,m}^{(j)} \geq G_{\alpha,m}^{(i)}$. This implies also that $G_{\alpha,m}^{(j)} \geq G_{\alpha,m}^{(i)}$, since otherwise, $G_{\alpha,m}^{(j)} < G_{\beta,m}^{(j)}$, contradiction. Then, $|G_m^{(j)} - G_m^{(i)}| = |\min_{k\in[n]}\{G_{k,m}^{(j)}\} - \min_{k\in[n]}\{G_{k,m}^{(i)}\}| = |G_{\beta,m}^{(j)} - G_{\alpha,m}^{(i)}| = G_{\beta,m}^{(j)} - G_{\alpha,m}^{(i)} \leq G_{\alpha,m}^{(j)} - G_{\alpha,m}^{(i)} = |G_{\alpha,m}^{(j)} - G_{\alpha,m}^{(i)}| \leq 1$, from the graded consistency part of Lemma 19.

If both $G_m^{(i)}, G_m^{(j)}$ are greater or equal to 1, then all $G_{k,m}^{(i)} \geq 1$ and $G_{k,m}^{(j)} \geq 1$ and then by the graded consistency of Mod-GC, it holds that for all parties P_k, $\mathsf{pretags}_{k,m}^{(i)} = \mathsf{pretags}_{k,m}^{(j)}$. To finish the proof, consider a case where $G_m^{(i)} = 1$ and some there exists some $k \in [n]$ for which $\mathsf{pretags}_{k,m}^{(i)} \neq \mathsf{pretags}_{k,m}^{(j)}$. Thus, from the graded consistency of Mod-GC, $G_{k,m}^{(i)} = 0$ or $G_{k,m}^{(j)} = 0$. Assume $G_{k,m}^{(j)} \neq 0$, then $G_{k,m}^{(i)} = 0$ which would lead to $G_m^{(i)} = 0$, contradiction. Thus, $G_{k,m}^{(j)} = 0$, meaning that $G_m^{(j)} = 0$ and the proof is complete. □

Lemma 22 (Termination of GCES.Toss)**.** *After running* GCES.Toss*, any honest party* P_i *terminates with output* $((\mathsf{pretags}_{k,j}^{(i)})_{k=1}^{n}, G_j^{(i)})$ *for every* $j \in [n]$.

Proof. This follows immediately from the termination of all Mod-GC instances and the fact that the sampling in Step 0 is guaranteed to return a result. □

We will need to use the fact that *all* honest tags have to be present in the adversary's identity id in order to be eligible for election. To this end, we prove that each value to which an honest party associates a strictly positive grade correctly uses the tags from the honest parties.

Lemma 23. *If a party* P_j *is graded as* $G_j^{(i)} \geq 1$ *by an honest party* P_i *in* Toss*, then* $\mathsf{pretags}_{k,j}^{(i)}$ *contains the puzzle* $\mathsf{pretags}_k$ *from each honest party* P_k.

Proof. Recall that the final grade for a party P_j is set by honest party P_i as $G_j^{(i)} = \min_{k\in[n]}\{G_{k,j}^{(i)}\}$. The fact that $G_j^{(i)} \geq 1$ means that for all $k \in [n]$, $G_{k,j}^{(i)} \geq 1$. For each index k corresponding to an honest party P_k, it means that honest party P_i has observed $\mathsf{pretags}_{k,j}^{(i)} = \mathsf{pretags}_k^{(i)}$, which implies that the honest value $\mathsf{pretags}_k^{(i)} = \mathsf{pretags}_k$ was included in $G_j^{(i)}$. To see why party P_i could not have observed $\mathsf{pretags}_{k,j}^{(i)} \neq \mathsf{pretags}_k^{(i)}$ and set $G_{k,j}^{(i)} = \min\{g_{k,j}^{(i)}, g^* - g_k^{(i)}\} > 0$, note that since $g_k^{(i)} = g^*$, the rule from Step 3 in Protocol 4.2 specifies that $G_{k,j}^{(i)} = 0$. □

Using the agreement properties we have now shown, we will show security properties of the committee elected via the process described in Section 4.2.1. These properties mirror those of the committee games from Section 4.1. Specifically, recall that in Section 4.1, we did not explicitly discuss different committee views for the honest parties. In the current section, since two different honest parties might hold different views on the pretags and public keys that define the committee, we explicitly consider each party's committee view separately. Recall that the properties we need

are that (i) the adversary cannot corrupt the entire committee and (ii) the adversary cannot elect more than B parties in the committee. For the moment, we only prove these properties for parties with a strictly positive grade assigned. Later, we will ensure that the views of different honest parties regarding the committee are aligned during the broadcast protocol for all grades' values.

4.2.3 Graded Committee Corruption Game

The graded committee corruption game is executed between n parties. We can think of it as an extension of the committee corruption game (see Section 4.1.3), differing in two major ways; 1) the previous use of the public bulletin-board for anything else than signing keys and the common view assumption at the challenger are replaced by different flavors of gradecast, which leads to: 2) each party has a different view of whether another party is in the committee or not. The current game aims to capture that all honest parties still have consistent views about the honest committee members, leading to the equivalent of Lemma 15 in this setting. The *graded committee corruption game* is defined as follows:

1. **Corruptions.** $\mathcal{A}$ can corrupt a party at the beginning of any round of the simulated protocols; before the game simulates a round, $\mathcal{A}$ can select additional parties to corrupt. Throughout, the game simulates all honest parties and queries $\mathcal{A}$ during the respective steps of the protocol for the actions of dishonest parties.
2. **Setup.** The game and $\mathcal{A}$ execute protocol $\mathsf{GCES.Gen}(\mathsf{par})$. As a result, for each honest party (at that time) P_i the game obtains $\left((\mathsf{pk}_i, \mathsf{sk}_i), (\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})_{j\in[n]\setminus\{i\}}\right)$.
3. **Toss phase.** The game and $\mathcal{A}$ execute protocol $\mathsf{GCES.Toss}(\mathsf{par})$. As a result, for each honest party (at that time) P_i the game obtains $((\mathsf{pretags}_{k,j}^{(i)})_{k\in[n]}, G_j^{(i)})_{j\in[n]}$, where $\mathsf{pretags}_j^{(i)} :=$

$\mathsf{pretags}_{j,i}^{(i)}$ for each $j \in [n]$.

4. **Winning Condition.** $\mathcal{A}$ wins the game if no honest party is *elected unanimously*:
   (a) Let Hon^* be the set of remaining honest parties.
   (b) The game runs $(\mathsf{res}_k, \mathsf{ticket}_k) \leftarrow \mathsf{CES.TryElect}(\mathsf{par}, \mathsf{sk}_k, (\mathsf{pretags}_i^{(k)})_{i\in[n]})$ for each honest party P_k.
   (c) The game defines the set of honest committee members as

$$\mathsf{HonComm} := \left\{ k \in \mathsf{Hon}^* \;\middle|\; \begin{array}{l} \mathsf{res}_k = 1 \wedge \forall i \in \mathsf{Hon}^* : \left(\tilde{g}_k^{(i)} = g^* \text{ and } G_k^{(i)} \geq g^* - 1\right) \wedge \\ \left(\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k^{(i)}, (\mathsf{pretags}_{j,k}^{(i)})_{j\in[n]}, \mathsf{ticket}_k) = 1\right) \end{array} \right\}.$$

   (d) The game outputs 1 if $\mathsf{HonComm} = \emptyset$. Otherwise, it outputs 0.

Intuitively, if the graded committee corruption game outputs 0, then it is guaranteed that at least one honest party will be in the committee and can convince any honest party that it is in the committee by sharing its ticket. We now show that this is the case for efficient adversaries.

Lemma 24. *Assume that the conditions for Lemmas 15 and 17 hold. Assume also that* GC *is a* g^**-gradecast protocol for* n *parties with corruption threshold* t *(see Definition 4). Then, the probability that the graded committee corruption game outputs* 1 *is at most* $\mathsf{negl}(\kappa) + \mathsf{negl}(\lambda)$.

Proof. Let $\mathcal{A}$ be a PPT adversary and $\varepsilon^{\mathcal{A}}$ denote the probability that the graded committee corruption game outputs 1. We aim to upper bound $\varepsilon^{\mathcal{A}}$. We do so by providing a sequence of hybrid games. For each hybrid $\mathbf{H}_i$, we denote the probability that it outputs 1 when run with adversary $\mathcal{A}$ by $\varepsilon_i^{\mathcal{A}}$.

Hybrid $\mathbf{H}_0$. With this we denote the graded committee corruption game. Therefore, by definition

$$\varepsilon^{\mathcal{A}} = \varepsilon_0^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_1$. We change *setup*, i.e., how the public keys are distributed. Specifically, recall that

until now, the game would run $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CES.Gen}(\mathsf{par})$ for each honest party P_i, and then run a gradecast, i.e., party P_i would call $\mathsf{GC}(\mathsf{pk}_i, g^*)$. Then, each honest party P_i would receive an output $(\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})$ from the gradecast with sender P_j. Now, in $\mathbf{H}_1$, we directly set $\mathsf{pk}_j^{(i)} := \mathsf{pk}_j$ and $\tilde{g}_j^{(i)} := g^*$ for honest sender P_j and honest receiver P_i. By gradecast's graded soundness property, we get

$$\varepsilon_0^{\mathcal{A}} = \varepsilon_1^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_2$. We change how $\mathsf{HonComm}$ is defined. Recall that so far, this set has been defined as

$$\mathsf{HonComm} := \left\{ k \in \mathsf{Hon}^* \;\middle|\; \begin{array}{l} \mathsf{res}_k = 1 \wedge \forall i \in \mathsf{Hon}^* : \left(\tilde{g}_k^{(i)} = g^* \text{ and } G_k^{(i)} \geq g^* - 1\right) \wedge \\ \left(\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k^{(i)}, (\mathsf{pretags}_{j,k}^{(i)})_{j\in[n]}, \mathsf{ticket}_k) = 1\right) \end{array} \right\}.$$

From $\mathbf{H}_2$ on, we instead define $\mathsf{HonComm}$ as

$$\mathsf{HonComm} := \left\{ k \in \mathsf{Hon}^* \;\middle|\; \mathsf{res}_k = 1 \wedge \mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k, (\mathsf{pretags}_j^{(k)})_{j\in[n]}, \mathsf{ticket}_k) = 1 \right\}.$$

We claim that the two definitions are equivalent for these two hybrids. Indeed, if $k \in \mathsf{HonComm}$ in $\mathbf{H}_2$, then trivially $k \in \mathsf{HonComm}$ in $\mathbf{H}_1$. Conversely, assume that $k \in \mathsf{HonComm}$ in $\mathbf{H}_1$, so, by definition, P_k is honest. Due to the change in $\mathbf{H}_1$, we know that for all honest parties P_i, it holds that $\tilde{g}_k^{(i)} = g^*$ and $\mathsf{pk}_k^{(i)} := \mathsf{pk}_k$. From the graded validity property (see Lemma 20), we get that for an honest moderator P_k, the game obtains with respect to each honest receiver P_i the values $(\mathsf{pretags}_{j,k}^{(i)})_{j\in[n]} = (\mathsf{pretags}_j^{(k)})_{j\in[n]}$ and $G_k^{(i)} \geq g^* - 1$. Thus, $k \in \mathsf{HonComm}$ in $\mathbf{H}_2$. So, we have

$$\varepsilon_1^{\mathcal{A}} = \varepsilon_2^{\mathcal{A}}.$$

Reduction to the Committee Corruption Game. Finally, one can easily bound $\varepsilon_2^{\mathcal{A}}$ via a reduction to the committee corruption game of Lemma 15. We sketch the reduction:

1. The reduction gets as input from the committee corruption game $\mathsf{par} \leftarrow \mathsf{CES.Setup}(1^\kappa)$. It

then starts to simulate hybrid $\mathbf{H}_2$ for $\mathcal{A}$. After $\mathcal{A}$ made its initial corruptions, the reduction declares the set of initially corrupted parties to the committee corruption game.

2. The reduction gets from the committee corruption game a public key pk_i for every honest party P_i and uses them to simulate the *setup* of $\mathbf{H}_2$. It also gets access to a corruption oracle, which it uses whenever $\mathcal{A}$ decides to corrupt a party.
3. Recall that in the tag phase of the committee corruption game, the reduction has to output n^2 pretags. To do so, the reduction runs the *toss phase* of the graded committee corruption game. Then, for each honest P_k, the reduction outputs $\mathsf{pretags}_i^{(k)}$ for each $j \in [n]$, and for each other $k \in [n]$ it outputs some default values. Note that these are not relevant for the committee corruption game.

The reduction perfectly simulates $\mathbf{H}_2$ for $\mathcal{A}$ and is efficient. Further, the committee corruption game outputs 1 whenever $\mathbf{H}_2$ does, which is primarily due to the change in $\mathbf{H}_2$. Then, via Lemmas 15 and 17, we obtain that

$$\varepsilon_2^{\mathcal{A}} \leq \mathsf{negl}(\kappa) + \mathsf{negl}(\lambda).$$

□

4.2.4 Graded Large Committee Game

The graded large committee game is executed between n parties. It extends the large committee game from Section 4.1.4, where, similarly to Section 4.2.3, honest parties might have different views on whether a party is in the committee. The current game captures that even so, the adversary cannot select—within an acceptable time frame depending on the difficulty parameter of the puzzles—pretags such that the committee exceeds size B in the view of any

honest party. The difficulty parameter of the puzzles should be determined by the first two steps of $\mathsf{GCES.Toss}$, which correspond to two sequential runs of GC with grade g^*. In particular, let $\Delta' = (4 \cdot g^* + 2) \cdot \Delta_r$ be the duration of these steps. Then, the difficulty of the puzzle is $\Delta(\Delta')$ as in Remark 1. The *graded large committee game* is formally written as follows:

1., 2., 3. The same steps as in the graded committee corruption game.

4. **Winning Condition.** $\mathcal{A}$ wins the game if there exists an honest party P_i for which too many parties are elected in its view:

   (a) Let Hon^* be the set of remaining honest parties.

   (b) The game runs $\mathcal{A}$ and gets, for every $i \in \mathsf{Hon}^*$, a set $\mathsf{Committee}^{(i)} \subseteq [n]$ and $\mathsf{ticket}_j^{(i)}$ for all $j \in \mathsf{Committee}^{(i)}$.

   (c) The game outputs 1 if the following two conditions hold for at least one $i \in \mathsf{Hon}^*$:

      i. We have $|\mathsf{Committee}^{(i)}| > B$.

      ii. For all $j \in \mathsf{Committee}^{(i)}$, it holds that:

$$\tilde{g}_j^{(i)} \geq 1, G_j^{(i)} \geq 1, \text{ and } \mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_j^{(i)}, (\mathsf{pretags}_{k,j}^{(i)})_{k\in[n]}, \mathsf{ticket}_j^{(i)}) = 1.$$

   Otherwise, the game outputs 0.

Lemma 25. *Assume that the conditions for Lemmas 17 and 18 hold. Assume also that* GC *is a* g^**-gradecast protocol for* n *parties with corruption threshold* t *(see Definition 4). Finally, assume that the conditions for Lemma 16 hold, where* TLP *is* $\Delta(\Delta')$*-CPA secure, for* $\Delta' := (4 \cdot g^* + 2) \cdot \Delta_r$*. Then, the probability that the graded large committee game outputs* 1 *is at most* $\mathsf{negl}(\kappa) + \mathsf{negl}(\lambda)$.

4.3 Our Broadcast Protocol

We now construct our sublinear broadcast protocol. Our protocol follows the committee-based protocol of Chan *et al.* [16]. The main difference is that we do not assume a trusted setup of keys and of public parameters. Instead, parties rely on a bulletin-PKI (only for the signature public keys), an unstructured common random string, and a graded protocol, which generates keys and tags, and can be seen as an online setup phase that issues graded random identifier strings and proofs of the validity of these identifiers. These identifiers are then used to correctly and verifiably elect bit-specific committees. We need bit-specific committees instead of a single committee to prevent the adaptive adversary from corrupting a party who voted for one bit and make it also vote for the other bit.

However, we do not exactly obtain the equivalent of a trusted setup from Section 4.2, as the only guarantees that parties have are related to the grades of these random strings, where the maximum grade is denoted as g^* and the minimum grade can be 0. This can be seen as a *graded mining functionality*, with the terminology of mining from Chan *et al.* [16], taken to mean consistent committee election and membership verification. Below, we describe how to use the grades to our advantage over a number of rounds, in order to obtain true agreement between parties.

4.3.1 Voting and Vote Distribution

Let us first review in more detail the broadcast protocol of Chan *et al.*: a party checks if it is in the committee for the bit b via an (ungraded) ideal mining functionality, which also allows other parties to validate this statement. This mining functionality is instantiated via an

adaptively secure VRF (in [16] implemented with non-interactive zero-knowledge proofs with trusted setup and commitment schemes). This enables parties to secretly but verifiably self-elect in a committee for a specific bit and only reveal their membership after they have performed their committee task, thus achieving security against an adaptive adversary. The protocol is composed of stages, each stage r having two rounds: distribution and voting. For a fixed number of rounds, each party observing a batch of r valid signatures from the committee members of b, which can be interpreted as votes, echoes this batch to all parties (distribution round). A party that is in the committee adds its vote if it observes a batch of r votes on the bit b for the first time, and multicasts the updated batch of $r+1$ signatures (voting round). Chan *et al.* show that it is possible to achieve consistency with overwhelming probability even if the number of rounds is constant and the committee size is also constant, against a constant corrupted fraction.

In our case, there is a key difference with respect to the mining functionality from Chan *et al.*, which is that the verification performed by other parties on the membership of one party will return a binary answer *and* a grade in $\{0, \ldots, g^*\}$. This mining functionality does not necessarily return the same grade to all parties, but the returned grades to two honest parties can differ by at most one. However, dishonest parties might try to convince honest parties to accept their membership despite having lower grades. To address this, we will interconnect the validity of the membership at a given round with the grade associated to the party that wants to prove is a committee member, such that the parties in which the honest parties do not have confidence can only vote in the very last round or not at all. Specifically, we set the maximum grade g^* that can be returned by the gradecast protocols to be equal to the number of rounds the Chan *et al.* protocol requires. We also say that a batch of r signatures is valid only if there are r signatures from parties on which the verification predicate returns 1 *and* which have a sufficiently large grade, greater than $g^* - 2r + 1$ (we will

define this more formally below). A symmetric way to view this is that at each Round ρ, the value of the grades have to be at least $g^* - \rho$, and the number of signatures has to be at least half of the round number. This ensures that parties that have a grade of 1 can only submit their signatures in the last possible round and parties that have grade of 0 cannot submit their signatures at all.

Since the grades obtained by the honest parties might differ by one (honest parties can be graded by g^* or $g^* - 1$), an honest party P_i might accept a batch with r signatures, i.e., all grades for the signers that P_i has are at least $g^* - 2r + 1$, but another honest party P_j might not accept it if it has a lower grade $g^* - 2r$ for one of the same signers. To avoid this, in Stage r, in the distribution Round $2r - 1$, where parties just echo what they received, honest parties accept r-batches with grade at least $g^* - 2r + 1$. At the end of voting Round $2r$ however, where committee parties multicast their votes after seeing a valid batch for that round (with grades at least $g^* - 2r$), committee parties are allowed to have a lower grade of at least $g^* - 2r$, in order to be picked up in the distribution round of Stage $r + 1$.

4.3.2 Broadcast Protocol Description

With these ideas in mind, we now describe our broadcast protocol in more detail.

Preparing Committee Election. There will be one committee for each value (0 or 1) that may be broadcasted. Therefore, the protocols GCES.Gen and GCES.Toss will produce strings for two bits, and the inputs of the algorithms CES.TryElect and CES.VerElect will be parametrized by the bit b. In the previous section, we preferred to describe GCES.Gen and GCES.Toss for a single committee for clarity purposes; in this section, we slightly abuse the previous notation to accommodate two committees. We emphasize that GCES.Gen and GCES.Toss are run a single time, not twice in

parallel.

Parties read the transparent system parameters par and first execute the key generation and distribution protocol GCES.Gen from Figure 4.1 to receive keys that correspond to each committee. In particular, each party P_i obtains the output

$$\left((\mathsf{pk}_i^b, \mathsf{sk}_i^b)_{b\in\{0,1\}}, \left((\mathsf{pk}_j^{(i),b})_{b\in\{0,1\}}, \tilde{g}_j^{(i)}\right)_{j\in[n]\setminus\{i\}}\right) \leftarrow \mathsf{GCES.Gen}(\mathsf{par}). \tag{4.3}$$

In the GCES.Toss protocol from Figure 4.2, parties will compute their pretags twice, namely, one set for each bit b. As such, each party P_i will obtain the output

$$\left(\left((\mathsf{pretags}_{k,j}^{(i),b})_{k=1}^n, G_j^{(i)}\right)_{j=1}^n\right) \leftarrow \mathsf{GCES.Toss}(\mathsf{pp}). \tag{4.4}$$

To recall, P_j is the moderator.

Note that in both GCES.Gen and GCES.Toss, although there are two (sets of) strings gradecast, one for every bit, the final grade is the same for both. In other words, if a party submits incorrect messages for one bit but not for the other, it will be penalized in both cases.

A party P_i can then run algorithm CES.TryElect algorithm for a specific bit $b \in \{0,1\}$, namely, it can run:

$$\left(\mathsf{res}^b, \mathsf{ticket}_i^b\right) \leftarrow \mathsf{CES.TryElect}\left(\mathsf{par}, \mathsf{sk}_i^b, (\mathsf{pretags}_j^{(i),b})_{j\in[n]}\right). \tag{4.5}$$

Finally, to verify that a party P_j is in the committee for bit b, P_i will run CES.VerElect:

$$\mathsf{res}_j^b := \mathsf{CES.VerElect}\left(\mathsf{par}, (\mathsf{pretags}_{k,j}^{(i),b})_{k\in[n]}, \mathsf{ticket}_j^{(i),b}\right). \tag{4.6}$$

That is, in verifying the committee claim of P_j, the verifying party P_i uses the values $(\mathsf{pretags}_{k,j}^{(i),b})_{k\in[n]}$ that it obtained as an output in Toss for P_j, but the values $\mathsf{ticket}_j^{(i),b}$ that party P_j provided to convince P_i that it is in the committee.

Tracking Committee Membership. Each party P_i maintains two variables, named res_i^0 and res_i^1.

The role of the variable res_i^b is to store the local view of whether P_i is in the committee for bit $b \in \{0,1\}$. Recall that as part of $\mathsf{CES.TryElect}$, for the bit b, the party P_i checks (once) whether it is in the committee for b and sets the variable res_i^b. In particular, at the beginning of the broadcast protocol, P_i checks the value of res_i^b for each bit b. During the later stages of broadcast, once they receive the first vote on b, P_i uses res_i^b to decide whether it will also send a vote on b or not. Once this check for b is done, P_i sets $\mathsf{res}_i^b = \perp$ to internally record this and not check (and vote) again.

The main part of the broadcast protocol can start after the amount of time it took for each honest party P_i to obtain the result of its committee election for each bit $(\mathsf{res}_i^0, \mathsf{res}_i^1)$. Recall that Δ_r denotes the duration of a round and that the time an honest party takes to solve a puzzle of difficulty $\Delta(\Delta')$ is $p_1(\kappa, \Delta) + p_2(\kappa, n)$ (Definition 19 and Remark 1). Based on Lemma 25, the difficulty parameter $\Delta(\Delta')$ is set based on the duration of two gradecast protocols, i.e., on $\Delta' := (4g^* + 2) \cdot \Delta_r$ rounds. We obtain this value of Δ' by instantiating the predicate Pred as in Section 4.1.5 and setting the maximum grade in the gradecast protocols g^* to be twice the committee bound, $g^* = 2 \cdot \lceil 3 \log(\delta^{-1})/\epsilon \rceil = O(\lambda/\epsilon)$. For convenience, we define $q_h := (p_1(\kappa, \Delta) + p_2(\kappa, n) - \Delta')/\Delta_r$, which the additional number of rounds an honest party takes to solve the batch puzzle compared to Δ' (in reality, this slow-down is very small and is definitely sublinear in the number of parties). We also set the number of the rest of the rounds in the broadcast protocol to also be g^*, which is sublinear in the number of parties. Intuitively, Lemma 17 and Lemma 18 guarantee that for this choice of parameters, the generated bit-specific committees contain at least one honest party, and at most $g^*/2$ dishonest parties with overwhelming probability.

Verifying Membership, Valid Batches and Certificates. Our protocol consists of stages, where

each stage is composed of two rounds. We denote the stage number by r for $r \in \{1, \ldots, g^*/2\}$. Then round 1 of stage r will be the $2r-1$'th round, and round 2 of stage r will be the $2r$'th round.

Each party collects *batches* of signatures. We will refer to a signature $\mathsf{Sig}_j(b)$ from party P_j as a j-signature. Parties (except for the sender) will send the previously collected signatures in a batch batch, as well as proofs of the committee elections for the bit b in a *certificate* called cert. In particular, a new batch batch'_b constructed from another batch batch_b, using a signature $\mathsf{Sig}_i(b)$ from a party P_i whose signature was not in batch_b, is denoted as $\mathsf{batch}'_b := \mathsf{batch}_b \| \mathsf{Sig}_i(b)$, where $\|$ means concatenation. Similarly, a new certificate constructed from another certificate cert_b, using a ticket ticket_i^b corresponding to a party P_i whose ticket was not in cert_b, is denoted as $\mathsf{cert}'_b := \mathsf{cert}_b \| \mathsf{ticket}_i^b$. To address the difference in grades in the two rounds, we define $(r,1)$-batches and $(r,2)$-batches. For notation purposes, we will use *ballot* to denote a batch and its associated certificate. For clarity, we prefer to define separately the validity of the batches and of the certificates, rather than lumped in a single definition of a valid ballot.

Definition 25 (Batches, Certificates, and Ballots)**.** *A* batch *consists of a number of votes, which are signatures associated to distinct parties. A* certificate *consists of a number of tickets coming from distinct parties. For a batch for bit b, batch_b, consisting of a number of signatures $(\mathsf{Sig}_j(b))_j$, we say certificate cert_b is associated to batch_b if it consists of tuples $(\mathsf{ticket}_j^b)_j$ for every j-signature present in batch_b, for parties $j \in [n]$. Finally, we call a pair of one batch and its associated certificate a* ballot.

We define a valid certificate only with respect to a certain batch. In our protocols, the verification of signatures is performed implicitly.

Definition 26 (Validity of Certificates)**.** *We say that a certificate cert_b is a* valid certificate *for a*

batch batch_b *from the view of a verifying party* P_i *if for all valid* j*-signatures in* batch_b *not coming from the sender, it holds that:*

$$\mathsf{CES.VerElect}\left(\mathsf{par}, (\mathsf{pretags}_{k,j}^{(i),b})_{k\in[n]}, \mathsf{ticket}_j^{(i),b}\right) = 1,$$

where for party P_i *we used the notation* $(\mathsf{ticket}_j^{(i),b})_j = \mathsf{cert}_b$ *for all* j*-signatures in* batch_b*, and* $\left((\mathsf{pretags}_{k,j}^{(i),b})_{k\in[n]}\right)_j$ *are from* P_i*'s state.*

Definition 27 (Validity of Batches and Ballots)**.** *We say that a batch* batch_b*, consisting of a tuple of form* $(\mathsf{Sig}_j(b))_j$*,* $j \in [n]$*, in Stage* r *for a bit* b*, is a* valid $(r, 1)$-batch *from the perspective of a verifying party* P_i *if:*

(i) It contains at least r *valid distinct signatures and one is from the sender* P_s*;*

(ii) For every valid j*-signature* $\mathsf{Sig}_j(b)$*, it holds that* $G_j^{(i)} \geq g^* - 2r + 1$*, and* $\tilde{g}_j^{(i)} \geq g^* - 2r + 1$*, where* $G_j^{(i)}, \tilde{g}_j^{(i)}$ *are from* P_i*'s state;*

(iii) It has associated a valid certificate cert_b *(*P_i *receives both* batch_b *and* cert_b*).*

Similarly, a batch is considered a valid $(r, 2)$-batch *by party* P_i *if (i), (iii) hold as above and also*

(ii)' For every valid j*-signature* $\mathsf{Sig}_j(b)$*, it holds that* $G_j^{(i)} \geq g^* - 2r$*, and* $\tilde{g}_j^{(i)} \geq g^* - 2r$*, where* $G_j^{(i)}, \tilde{g}_j^{(i)}$ *are from* P_i*'s state.*

Moreover, a ballot is valid *if its batch is valid and its certificate is a valid certificate for that batch.*

Further, each party P_i will also maintain a set $\mathsf{Extracted}_i$, initialized to the empty set. A party P_i adds a bit b to their set $\mathsf{Extracted}_i$ in a round if it receives a valid batch and a valid certificate on b for that round. With these definitions, the full broadcast protocol is described in Figure 4.3.

Π_{BC}: Sublinear-round, plain PKI Broadcast

Let $g^* := 2\left\lceil 3\log\left(\delta^{-1}\right)/\epsilon\right\rceil = O(\lambda/\epsilon)$.

Rounds $-6g^* - 3 - q_h$[a] to -1 :

1. Each party P_i performs (4.3), (4.4), (4.5) and stores their outputs. In particular, it stores $(\mathsf{res}_i^0, \mathsf{ticket}_i^0, \mathsf{res}_i^1, \mathsf{ticket}_i^1)$ as its own election result for committees 0 and 1.

Stage 0:

1. (Round 0) Each party P_i initializes $\mathsf{Extracted}_i = \emptyset$.
   The designated sender P_s sends $(b_s, \mathsf{Sig}_s(b_s), \perp)$ to all parties, where b_s is the sender's input bit.

Stage $r = 1$ to $g^*/2 - 1$:

1. (Round $2r-1$) Each party P_i accepts a message $b \notin \mathsf{Extracted}_i$, i.e., sets $\mathsf{Extracted}_i \leftarrow \mathsf{Extracted}_i \cup \{b\}$, only if it is accompanied by some valid ballot $(\mathsf{batch}_b, \mathsf{cert}_b)$, where batch_b is a valid $(r,1)$-batch and cert_b is a valid certificate for batch_b.
   P_i then multicasts $(b, \mathsf{batch}_b, \mathsf{cert}_b)$ to all parties.
2. (Round $2r$) Each party $P_i \neq P_s$ checks all bits b that it received on whether they are accompanied by a valid ballot $(\mathsf{batch}_b, \mathsf{cert}_b)$, where batch_b is a valid $(r,2)$-batch and cert_b is a valid certificate for batch_b.
   For each bit b, P_i checks if $\mathsf{res}_i^b = 1$, and if so:
   - sets $\mathsf{Extracted}_i \leftarrow \mathsf{Extracted}_i \cup \{b\}$;
   - constructs a $(r+1,1)$-batch $\mathsf{batch}'_b := \mathsf{batch}_b \| \mathsf{Sig}_i(b)$, $\mathsf{cert}'_b := \mathsf{cert}_b \| \mathsf{ticket}_i^b$.

   P_i then sets $\mathsf{res}_i^b = \perp$ and sends $(b, \mathsf{batch}'_b, \mathsf{cert}'_b)$ to all n parties.

Stage $r = g^*/2$:

1. (Round $g^* - 1$) Each party P_i accepts each message $b \notin \mathsf{Extracted}_i$, i.e., sets $\mathsf{Extracted}_i \leftarrow \mathsf{Extracted}_i \cup \{b\}$, that is accompanied by a valid $(g^*/2, 1)$-batch and a valid certificate for that batch.
   P_i then outputs either the message $b' \in \mathsf{Extracted}_i$ if $|\mathsf{Extracted}_i| = 1$, or a canonical message otherwise.

[a] q_h is the extra number of rounds over $4g^* + 2$ that an honest party takes to solve a puzzle with difficulty $(4g^* + 2) \cdot \Delta_r$ (negligible in practice).

Figure 4.3: Broadcast protocol for designated sender P_s and parties $P_1, \ldots, P_n$.

4.3.3 Proof of Broadcast Properties

We now show that Π_{BC} is a broadcast protocol as defined in Definition 2. Our setup assumptions are the existence of a bulletin-PKI and a uniform common random string, and our computational assumptions are the hardness of discrete logarithm and the security of time-lock puzzles and zero-knowledge proofs.

Theorem 5. *Consider a PPT adversary who can adaptively corrupt $(1-\epsilon)n$ parties, for a constant $\epsilon \in (0,1)$. Then, under the assumptions above, the protocol Π_{BC} given in Figure 4.3 satisfies the*

properties of a broadcast protocol except with probability $\mathsf{negl}(\lambda) + \mathsf{negl}(\kappa)$.

The proof ties together all the results based on committee elections, graded consensus on the committees, and on the valid ballot creation and verification. We will write the proof of Theorem 5 as three separate lemmata, each proving a property of the broadcast protocol: Lemma 28 for validity, Lemma 29 for consistency, and Lemma 30 for termination. For validity and consistency, we first show some helper results.

Lemma 26. *Consider a Stage* $r < g^*/2$*, a bit* b *and a ballot* $(\mathsf{batch}_b, \mathsf{cert}_b)$ *that is* $(r, 1)$*-valid for honest party* P_i*. Then,* $(\mathsf{batch}_b, \mathsf{cert}_b)$ *will be* $(r, 2)$*-valid for any honest party* P_j*.*

Proof. We will prove that if an honest party P_i accepted a ballot in Round $2r - 1$ and forwarded it, then an honest party P_j will accept the same ballot in Round $2r$.

Since batch_b is considered valid by party P_i, it means it is accompanied by a valid certificate cert_b, and according to Definition 27, the following hold:

(i) It contains at least r valid distinct signatures and one of the signatures is from the sender P_s;

(ii) For every valid k-signature $\mathsf{Sig}_k(b) \in \mathsf{batch}_b$, it holds that $G_k^{(i)} \geq g^* - 2r + 1$; and $\tilde{g}_k^{(i)} \geq g^* - 2r + 1$, where $G_k^{(i)}, \tilde{g}_k^{(i)}$ are from P_i's state.

(iii) cert_b is valid.

Now assume honest party P_j receives $(b, \mathsf{batch}_b, \mathsf{cert}_b)$ during the second round of Stage r and checks whether batch_b is a valid $(r, 2)$-batch. We show that all three properties according to Definition 27 will hold for P_j.

(i) It holds trivially from the fact that batch_b is a $(r, 1)$-valid batch and the signing public keys are posted on the bulletin-PKI.

(ii)' For all signatures $\mathsf{Sig}_k(b) \in \mathsf{batch}_b$, from the graded agreement property of GC in GCES.Gen and GCES.Toss, it holds that $|\tilde{g}_k^{(i)} - \tilde{g}_k^{(j)}| \leq 1$ and $|G_k^{(i)} - G_k^{(j)}| \leq 1$. Since $G_k^{(i)} \geq g^* - 2r + 1$, it also holds that $G_k^{(j)} \geq g^* - 2r$, for any $r < g^*/2$, and similarly $\tilde{g}_k^{(j)} \geq g^* - 2r$.

(iii) From item (ii), it holds that $\tilde{g}_k^{(i)}, \tilde{g}_k^{(j)}, G_k^{(i)}, G_k^{(j)}$ are always strictly positive for $r \leq g^*/2 - 1$ (in particular, the minimum grade is at least 2). Then, from the graded agreement of GCES.Toss, it holds that $\mathsf{pretags}_{l,k}^{(i),b} = \mathsf{pretags}_{l,k}^{(j),b}$ for all $k, l \in [n] : \mathsf{Sig}_k(b) \in \mathsf{batch}_b$. The same argument holds for $\mathsf{pk}_k^{(i)} = \mathsf{pk}_k^{(j)}$ for all $k \in [n] : \mathsf{Sig}_k(b) \in \mathsf{batch}_b$. Therefore, it follows that since cert_b is valid for P_i, it is also valid for P_j (since P_j will perform the same checks on the same values of pretags and public keys, contained in batch_b, and tickets, contained in cert_b, as P_i did).

□

Lemma 27. *Consider a Stage $r < g^*/2$, a bit b and a batch batch_b (with certificate cert_b) that is $(r, 2)$-valid for honest party P_i for which $\mathsf{res}_i^b = 1$ and $\mathsf{Sig}_i(b) \notin \mathsf{batch}_b$. Then, the ballot $(\mathsf{batch}_b', \mathsf{cert}_b')$ with $\mathsf{batch}_b' := \mathsf{batch}_b \| \mathsf{Sig}_i(b)$ and $\mathsf{cert}_b' := \mathsf{cert}_b \| \mathsf{ticket}_i^b$ will be $(r+1, 1)$-valid for any honest party P_j.*

Proof. For a party P_i in the b-committee (i.e. $\mathsf{res}_i^b = 1$), that receives a valid $(r, 2)$-batch in Stage r, we prove that in Stage $r + 1$ its updated batch will be $(r + 1, 1)$-valid for all honest parties.

Since batch_b is considered $(r, 2)$-valid by party P_i, it means that the following hold for batch_b according to Definition 27:

(i) It contains at least r valid distinct signatures and one of them is from the sender P_s;

(ii)' For every valid k-signature $\mathsf{Sig}_k(b) \in \mathsf{batch}_b$, it holds that $G_k^{(i)} \geq g^* - 2r$; and $\tilde{g}_k^{(i)} \geq g^* - 2r$, where $G_k^{(i)}, \tilde{g}_k^{(i)}$ are from P_i's state.

(iii) cert_b is valid.

Now assume honest party P_j receives $(b, \mathsf{batch}'_b, \mathsf{cert}'_b)$ during the first round of Stage $r+1$ and checks whether batch'_b is a valid $(r+1, 1)$-batch. We show that all three properties according to Definition 27 will hold for P_j.

(i) First, since P_i is honest and by assumption $\mathsf{Sig}_i(b) \notin \mathsf{batch}_b$, then P_j will accept $\mathsf{Sig}_i(b)$ as a valid, distinct signature. Moreover, given that the signing public keys are posted on the bulletin-PKI, all signatures that were valid for P_i are valid for P_j. Therefore, it holds that $\mathsf{batch}'_b = \mathsf{batch}_b \| \mathsf{Sig}_i(b)$ contains at least $r+1$ distinct signatures and one of them is from the sender P_s from property (i) of batch_b above.

(ii) For all signatures $\mathsf{Sig}_k(b) \in \mathsf{batch}_b$, from the graded agreement property of GC in GCES.Gen and GCES.Toss, it holds that $|\tilde{g}_k^{(i)} - \tilde{g}_k^{(j)}| \leq 1$ and $|G_k^{(i)} - G_k^{(j)}| \leq 1$. Since $G_k^{(i)} \geq g^* - 2r$, it holds that $G_k^{(j)} \geq g^* - 2r - 1 = g^* - 2(r+1) + 1$. Furthermore, since P_i is honest, from graded validity of GCES.Toss it holds that for every honest party P_j: $G_i^{(j)} \geq g^* - 1 = g^* - 2(1+1) + 1$. Similarly, it holds that $g_k^{(j)} \geq g^* - 2r - 1 = g^* - 2(r+1) + 1$, and moreover, honest party P_j will have for honest party P_i that $\tilde{g}_i^{(j)} = g^*$. To conclude, for any honest party P_j it holds that for every valid k-signature $\mathsf{Sig}_k(b) \in \mathsf{batch}'_b$: $G_k^{(j)} \geq g^* - 2(r+1) + 1$, and $\tilde{g}_k^{(j)} \geq g^* - 2(r+1) + 1$.

(iii) From item (ii), for all $r \leq g^*/2 - 1$, it holds that $\tilde{g}_k^{(i)}, \tilde{g}_k^{(j)}, G_k^{(i)}, G_k^{(j)}$ are strictly positive (in particular, the minimum grade is at least 1). Then, from the graded agreement property of GCES.Toss, it holds that $\mathsf{pretags}_{l,k}^{(i),b} = \mathsf{pretags}_{l,k}^{(j),b}$ for all $k, l \in [n] : \mathsf{Sig}_k(b) \in \mathsf{batch}_b$. The same argument holds for $\mathsf{pk}_k^{(i)} = \mathsf{pk}_k^{(j)}$ for all $k \in [n] : \mathsf{Sig}_k(b) \in \mathsf{batch}_b$. Therefore, it follows that since cert_b is valid for P_i, it is also valid for P_j (since P_j will perform the same checks on the same values of pretags and public keys, contained in batch_b, and tickets,

contained in cert_b, as P_i did). Finally, from the graded validity of $\mathsf{GCES.Toss}$ it holds that $\mathsf{pretags}_{k,i}^{(j),b} = \mathsf{pretags}_k^{(i),b}$, for all $k \in [n]$. So, ticket_i^b is valid for P_j with respect to $\mathsf{pretags}_k^{(i),b}$, since $\mathsf{res}_i^b = 1$. Therefore cert' is a valid certificate for batch'.

□

Proof of Theorem 5. We now prove Theorem 5. To this end, we show soundness, consistency, and termination of our protocol in Figure 4.3.

Lemma 28. *Protocol* Π_{BC} *achieves soundness with probability* $1 - \mathsf{negl}(\kappa)$.

Proof. In Stage 0, the sender P_s sends $(b_s, \mathsf{Sig}_s(b_s), \perp)$ to all parties. By Round 1, all honest parties have a valid $(1,1)$-batch for b_s and add b_s to their extracted sets. Since P_s is honest, the security of the signature scheme ensures that no malicious party can inject another validly signed message, so parties only elect themselves in a single committee for b_s and will not accept as valid any batch $\mathsf{batch}_{\bar{b}_s}$. Thus, soundness holds, unless with negligible probability of the adversary forging the sender's signature. □

Lemma 29. *Protocol* Π_{BC} *achieves consistency with probability* $1 - \mathsf{negl}(\lambda) - \mathsf{negl}(\kappa)$.

Proof. Let P_i, P_j be two honest parties. Assume without loss of generality that P_i adds a value $b \in \mathsf{Extracted}_i$ before P_j. We show that by the end of the protocol $b \in \mathsf{Extracted}_j$. This is sufficient to prove consistency, since it implies that by the end of the protocol $\mathsf{Extracted}_i = \mathsf{Extracted}_j$ (set equality). Therefore the outputs of P_i, P_j for the broadcast protocol will match. We distinguish two cases.

Case 1. P_i adds b during Stage $r^* = g^*/2$: In that case, P_i must have received a valid $(g^*/2, 1)$-ballot for b. According to Definition 27, the batch in the ballot must contain valid signatures

from at least $g^*/2$ parties and for each such party P_k it must hold that $\tilde{g}_k^{(i)} \geq 1$ and $G_k^{(i)} \geq 1$. Furthermore, the batch is accompanied by a valid certificate cert_b for it, i.e. a set of values ticket_k^b, for each P_k whose signature is in the batch. To prove consistency, we need that at least one of the signatures must have come from an honest party P_h with probability $1 - \mathsf{negl}(\lambda) - \mathsf{negl}(\kappa)$. This is true because, first, by Lemma 25, the committee cannot be larger than $B = g^*/2$ except with negligible probability—else there exists a direct reduction that wins the graded honest committee game with non-negligible probability—therefore the ballot contains the signatures of all parties in the committee, and there cannot exist be a distinct valid ballot from $g^*/2$ parties. Secondly, by Lemma 24, the adversary cannot corrupt the entire committee before the honest committee members reveal themselves, except with negligible probability—else there exists a direct reduction that wins the graded committee game with non-negligible probability—so the ballot contains a signature from at least an honest party P_h.

We now show that the honest committee member P_h would have sent its ballot to all parties by Round 1 of Stage r^*. If P_h added its own signature to a batch at Round 2 of a Stage $r < r^* - 1$ (and thus sent the updated ballot to all parties), then according to Lemma 27, P_i would have added b during $r + 1$, which contradicts our assumption. So, P_h added its own signature to a batch at Round 2 of Stage $r^* - 1$ and sent the updated ballot to all parties, including honest party P_j. So, by Lemma 27, it must hold that $b \in \mathsf{Extracted}_j$.

Case 2. P_i adds b during Stage $r < g^*/2$: First note that if P_i has $\mathsf{res}_i^b = 1$ then, in the second round of Stage r, P_i would send an updated ballot to all parties. By Lemma 27, any honest party P_j would add $b \in \mathsf{Extracted}_j$ in Round 1 of Stage $r + 1$.

Now assume that P_i adds b during Round 1 of Stage $r < g^*/2$ and $\mathsf{res}_i^b = 0$. P_i would then send the valid $(r, 1)$-ballot for b to all parties. By Lemma 26, any honest party considers

that ballot to also be $(r, 2)$-valid. Then, by Lemma 24, at least one honest party P_h would have $\mathsf{res}_h^b = 1$ with probability $1 - \mathsf{negl}(\lambda) - \mathsf{negl}(\kappa)$; otherwise, the adversary would have to corrupt the entire committee before the honest committee members reveal themselves.

The honest committee member P_h however would have sent its ballot to all parties and all honest parties P_j would add $b \in \mathsf{Extracted}_j$ by Round 1 of Stage $r + 1$.

Therefore, in any case $b \in \mathsf{Extracted}_j$. □

Lemma 30. *Protocol* Π_{BC} *terminates.*

Proof. Termination holds by the termination of GCES.Gen, GCES.Toss and CES.TryElect, which act like an online setup, and by construction of the subsequent $g^*/2$ stages. □

Combining 28, 29 and 30 completes the proof of 5.

Round Complexity. Next, we give a bound on the round complexity of our protocol.

Theorem 6. *Consider a PPT adversary who can adaptively corrupt* $(1-\epsilon)n$ *parties, for a constant* $\epsilon \in (0, 1)$. *Then, the protocol* Π_{BC} *(Figure 4.3) has round complexity* $O\left(\frac{1}{\epsilon} \log\left(\frac{1}{\delta}\right)\right)$.

Proof. Recall that $q_h = (p_1(\kappa, \Delta) + p_2(\kappa, n) - \Delta')/\Delta_r$ is the additional number of rounds (the slowest) honest party takes to solve the batch puzzle compared to Δ'/Δ_r, for $\Delta' = (4g^* + 2) \cdot \Delta_r$, and is a small constant number. Then, the first step of the broadcast protocol, before Stage 0, takes $6g^* + 3 + q_h$ rounds, as follows. GCES.Gen has to be ran before GCES.Toss and takes $2g^* + 1$ rounds; GCES.Toss takes $4g^* + 2$ rounds, but after the first $2g^* + 1$ of GCES.Toss, honest parties can start CES.TryElect, which involves solving a batch puzzle with time complexity Δ', so the total number of rounds is $2g^* + 1 + \Delta'/\Delta_r + q_g = 6g^* + 3 + q_h$.

Stage 0 takes one round and the subsequent $g^*/2$ stages take a total of $g^* - 1$ rounds. As a result, the round complexity of broadcast in terms of g^* and q_h is $7g^* + 3 + q_h$. For $g^* =$

$2 \cdot \lceil 3 \log(\delta^{-1})/\epsilon \rceil = O(\lambda/\epsilon)$ as described above, the broadcast protocol has round complexity $O(\log(\delta^{-1})/\epsilon)$. For $\delta = \exp(-\omega(\log \lambda))$ negligible in the security parameter, we obtain a round complexity of $O(\lambda/\epsilon)$. □

Communication Complexity. The communication complexity of broadcast is dominated by the $\mathsf{GCES.Toss}$ protocol, which is executed once for every broadcast instance. Therefore, $O(\mathsf{CC}_{\mathsf{Gen}} + \mathsf{CC}_{\mathsf{Toss}} + g^* \cdot \kappa \cdot n^3) = O(\kappa \cdot n^5 \cdot \log(\delta^{-1})/\epsilon)$. For $\delta = \exp(-\omega(\log \lambda))$ negligible in the security parameter, we obtain a total communication complexity of $O(\kappa \cdot \lambda \cdot n^5/\epsilon)$. Using the version of $\mathsf{GCES.Toss}$ with gossiping, the total communication complexity reduces to $\tilde{O}(\kappa \cdot \lambda \cdot n^4/\epsilon)$.

Chapter 5: Deterministic BA with Adaptive $O(n \cdot f)$ Communication

In this chapter, we focus on the *communication complexity* of deterministic binary BA. In particular, we present the first protocol that achieves deterministic binary BA[1] with adaptive $O(n \cdot f)$ communication complexity[2] and near-optimal resilience $t < (\frac{1}{2} - \epsilon)n$, where $f \leq t$ is the *exact* number of faulty parties. Our results theoretically improves the quadratic $O(n^2)$ communication bound achieved by previous work [15]. For example, for executions where f is constant, our work achieves a linear communication bound. Our protocol does not require knowledge of the exact value of f. In Section 5.1 we describe how to simplify BA into two separate weaker problems; Reliable Voting and Weak BA. Then, once one has obtained efficient protocols for each of the two subproblems, we see how to combine the results to achieve BA. Furthermore, we give a brief description of the notion of *silent phases* and how to utilize them.

Then in Sections 5.2 and 5.3 we provide a deeper look in to the details of instantiating an efficient Reliable Voting and Weak BA protocol respectively.

[1]Note that for the binary case, BA and strong BA where parties must agree *always* on the initial input of some honest party, are exactly the same. This is not trivially the case in the multivalued setting.

[2]Communication complexity refers to the number of words, where a word is a constant-size object. This is consistent with all related work on the setting.

5.1 Overview of our BA Protocol.

Breaking BA down to weaker problems: Reliable Voting and Weak BA. To achieve deterministic BA with $O(n \cdot f)$ communication, we reduce the problem into two distinct, weaker versions, namely *Reliable Voting* (RV) and *Weak Byzantine Agreement* (WBA). We then provide protocols with $O(n \cdot f)$ communication for both RV and WBA.

On a very high level, RV (Definition 7) is a protocol that allows parties to tally their votes for which value is to be agreed upon. In RV, every party starts with some value in $\{0, 1\}$ and upon termination, every party outputs a value v alongside some auxiliary information aux. What we wish RV to achieve is *provable validity* with respect to a fixed Boolean predicate $\mathsf{validate}$; if all honest parties have the same input value v, then they all output v along with aux such that $(\mathsf{v}, \mathsf{aux})$ satisfy $\mathsf{validate}$. We must also guarantee that no byzantine party can output a different value $\mathsf{v}' \neq \mathsf{v}$ with auxiliary information aux' that satisfy $\mathsf{validate}$.

In our RV construction, aux represents a proof that there exist $n-t$ distinct digital signatures on v, r (for some phase number $r = 1, \ldots, n$), in which case we call the respective Boolean predicate a "threshold predicate"[3]. A keen reader might have already noticed that RV achieves the validity property of BA, however, it does not achieve agreement (Nothing is guaranteed for the case where honest parties start with different values.)

To achieve agreement, we utilize WBA (Definition 8); each party now starts with some input value v along with some auxiliary information aux and upon termination outputs a decision value. The protocol must satisfy *agreement* and *restricted validity*, again with respect to a fixed

[3] An implementation of this predicate could simply be a verification of $n-t$ distinct signatures. In our construction we will use a more communication-efficient implementation, i.e. through verification of succinct arguments.

Boolean predicate validate. Agreement is the same as for BA with respect to the output values. Restricted validity guarantees that, in case all honest parties start with the same input value v and auxiliary information aux, such that $(\mathsf{v}, \mathsf{aux})$ satisfy validate, and no party starts with a different value $\mathsf{v}' \neq \mathsf{v}$ and some auxiliary information aux' that satisfy validate, then all honest parties output v. Again, the validate predicate in our WBA construction will be the threshold predicate.

BA from RV and WBA: Combining the weaker primitives to achieve BA. Observe that the input requirements for WBA's restricted validity are satisfied by the output requirements guaranteed by RV's provable validity, assuming they both use the same predicate. This is not a coincidence. Our BA construction simply takes the input values of BA and runs them through an RV protocol that enhances them with aux so as to satisfy the threshold predicate whenever all honest parties start with the same input. Then, it runs the output pair (value and aux) through a WBA protocol to also guarantee agreement. The simple pseudocode of our proposed BA is given in Figure 5.1.

Algorithm $\Pi_{\mathsf{BA}}(\mathsf{v}_i)$:
1: $\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i \leftarrow \Pi_{\mathsf{RV}}(\mathsf{v}_i)$
2: $\mathsf{y}_{\mathsf{i}} \leftarrow \Pi_{\mathsf{WBA}}(\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i)$
return y_{i}

Figure 5.1: Code of Π_{BA} for party P_i.

It is fairly straightforward to prove the following theorem.

Theorem 7 (Byzantine Agreement from Reliable Voting and Weak Byzantine Agreement)**.** *Let* Π_{RV} *be an RV protocol, as per Definition 7, with respect to a predicate* validate*. Let* Π_{WBA} *be a WBA protocol, as per Definition 8, with respect to the same predicate* validate*. Then* Π_{BA} *from Figure 5.1 is a BA protocol, as per Definition 6.*

Proof. For validity, assume each honest party P_i starts with value $\mathsf{v}_i = \mathsf{v}$. The execution of $\Pi_{\mathsf{RV}}(\mathsf{v}_i)$ ensures that all honest parties output $(\mathsf{v}, \mathsf{aux}_i)$ such that $\mathsf{validate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$. At

the same time, no party generates an output of $(\mathsf{v}' \neq \mathsf{v}, \mathsf{aux}_i)$ where $\mathsf{validate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$, as guaranteed by RV. In Line 2, all honest parties invoke Π_{WBA} with the output $(\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i)$ of Π_{RV}. Thus, based on the restricted validity property of WBA, all honest parties decide v. For agreement, Π_{WBA} achieves agreement independently of the input with which it is initialized. Since Π_{WBA} is the last call of every honest party, the agreement property of Π_{BA} is guaranteed from the agreement property of Π_{WBA}. Termination follows from the termination of Π_{RV} and Π_{WBA}. □

Communication complexity and the use of silent phases. Let us estimate the communication complexity of the above protocol. For one thing, naive implementations of RV or WBA can lead to $O(n^2)$ communication complexity. For example, one can implement RV by having each party sign its input and send its signature along with its input to all other parties, which would lead to quadratic communication complexity. Since our goal is to achieve $O(n \cdot f)$ communication, both RV and WBA must maintain a communication of $O(n \cdot f)$. To this end, both our proposals for RV and WBA will follow the paradigm of *silent phases*, introduced by Spiegelman [45].

In this paradigm, protocols execute over n round-robin phases, where each party is the leader of its respective phase. Communication takes place only (1) from leader to parties and (2) from parties to leader. In our work we also require that communication is restricted to a fixed connectivity structure—expander graphs (Malicious parties can communicate with each other as they choose, but this communication is not counted in the total communication.) Throughout each phase, the goal is for the leader to build and send out certificates, which will ensure that the leader made all honest parties agree on the same output, which satisfies some additional, protocol-specific properties. Once an *honest* leader has sent such a certificate, all subsequent honest leaders will not invoke any communication during their respective phases, i.e., they remain

silent. Since it typically takes f phases for an honest leader to be picked (and nobody speaks after that) and each phase typically causes the communication of $O(n)$ words (due to expander graphs), protocols in this paradigm achieve $O(n \cdot f)$ communication.

In the next sections, we utilize SNARKs (succinct NIZKs satisfy), threshold signatures, and collision-resistant hash functions to reduce the communication of our required proofs and certificates during each phase. This, in conjunction with the use of expanders, facilitates the efficient dissemination of certificates between parties, achieving $O(n \cdot f)$ word complexity. While describing the protocols, we use several variables and predicates, which we understand might be difficult for a reader to follow. To facilitate that, we provide Table 5.1, hoping to clarify notation.

5.2 The Reliable Voting Problem

In this section we propose protocol Π_{RV} for RV—see Figure 5.2. Recall that in RV, each party starts with initial value $\mathsf{v}_i \in \mathsf{V}$, where $\mathsf{V} = \{0, 1\}$, and generates output value $\mathsf{y}_i \in \mathsf{V}$, and auxiliary information aux_i. As mentioned earlier, RV plays a crucial role in ensuring validity for BA through its provable validity property. Our Π_{RV} consists of two subroutines: *Polling* (Section 5.2.1) and *Rebuttal* (Section 5.2.2), as depicted in Figure 5.2. Our Π_{RV} protocol is instantiated with respect to the Boolean predicate denoted by $\mathsf{ThresholdPredicate}$. Specifically, $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i)$ is true if and only if aux_i contains proof of existence of $n-t$ distinct signatures (votes) on v, r for some $r \in [n]$, where the proof here is constructed using SNARKs.

Variable	Definition
N	Set of public keys belonging to the n parties.
$\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$	r: phase number in which ballot was created; count_0: number of parties that have signed 0; count_1: number of parties that have signed 1; H_A: a hash of the list of public keys of accused parties; π_{ballot}: proof of consistency of $(\mathsf{count}_0, \mathsf{count}_1, H_A)$.
accList	list of accused parties committed to by H_A.
Ψ_i	Holds all accList that P_i has received wrt ballot. $\Psi_i[\mathsf{ballot}]$ retrieves accList, or $\perp$.
QC_{commit}	Aggr. sig. of $n-t$ sigs on $\langle \mathsf{ballot}, r \rangle$ for Π_{Polling}. Aggr. sig. of $n-t$ sigs on $\langle \mathsf{v}, r \rangle$ for Π_{WBA}.
Commit Certificate for Π_{Polling}/Π_{WBA}	$\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$/$\langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle$
QC_{decide}	Aggr. sig. of $n-t$ sigs on $\langle decide, \mathsf{ballot}, r \rangle$ for Π_{Polling}. Aggr. sig. of $n-t$ sigs on $\langle decide, \mathsf{v}, r \rangle$ for Π_{WBA}.
Decide Certificate for Π_{Polling}/Π_{WBA}	$\langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle$/$\langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle$.
$\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux})$	$\mathsf{SNRK}_3.\mathsf{Verify}(\mathsf{crs}_3, (\mathsf{v}), \mathsf{aux})$ (see Section 5.2.2).
$\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$	true if $\mathsf{v}_{\mathsf{commit}} = \perp$ **and** $\big(\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ **or** $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}\big)$.

Table 5.1: Variables and predicates used throughout all our protocols used by party P_i.

5.2.1 The Polling Protocol

The Polling protocol (Figures 5.4, 5.3) operates in two separate layers. First, in the leader-based layer, which utilizes the silent phases framework, communication is between parties and a leader, who may opt to remain silent. Then, in the processing layer, which occurs at the end of all n silent phases, parties that have not decided on a valid output can ask for and receive help. In

Algorithm $\Pi_{\mathsf{RV}}(\mathsf{v}_i)$:

1: $\mathsf{ballot}_i, \mathsf{accList}_i \leftarrow \Pi_{\mathsf{Polling}}(\mathsf{v}_i)$
2: $\mathsf{y}_i, \mathsf{aux}_i \leftarrow \Pi_{\mathsf{Rebuttal}}(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$

return $(\mathsf{y}_i, \mathsf{aux}_i)$

Figure 5.2: Code of Π_{RV} for party P_i.

essence, the Polling protocol aims to tally parties' initial values of the form $\langle \mathsf{v}_i, r \rangle$, where r is the current phase number, create a list, accList, of parties whose votes were not counted, then provide proof of correct tallying, and share the result with everyone in the form of a ballot. By the end of Polling, it is guaranteed that all honest parties agree on the same ballot; however, parties reach weak agreement on accList; An honest party either outputs $\mathsf{accList}_i = \mathsf{accused}$ or $\perp$, but no other honest party output $\mathsf{accList}_j \neq \mathsf{accused}$.

Protocol Overview. The protocol consists of a leader-based layer and a final processing layer. The former runs for n phases, each phase involving four different parts:

1. ***Prepare***: In each phase, the leader remains silent if decided ($\mathsf{ballot}_i \neq \perp$). Otherwise, the leader seeks help by sending $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ to all parties. Upon receiving $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ from the leader, if a party P_j is committed to a certain ballot, $\mathsf{ballot}_{\mathsf{commit}} = \mathsf{ballot}$, in a previous phase, P_j forwards ballot to the leader along with $\mathsf{rank}_{\mathsf{commit}}$ (the phase in which commit certificate was created), and π_{commit} (proof of the validity of the commit certificate). The π_{commit} is $n - t$ signatures on $\langle \mathsf{ballot}, \mathsf{rank}_{\mathsf{commit}} \rangle$ (QC_{commit}). Otherwise, P_j sends to the leader the initial value $\langle \mathsf{v}_j, r \rangle_\sigma$, where r is the current phase number. Note, that including the phase number in the vote is important to prevent malicious leaders from using votes from older phases to create a valid ballot.

2. ***Pre-commit***: In case the leader receives multiple valid commit certificates $\langle \mathsf{ballot}, \mathsf{rank}_{\mathsf{commit}}, QC_{\mathsf{commit}} \rangle$ from parties, where a commit certificate is considered valid if $\mathsf{SS.Verify}(pk_{ss}, \langle \mathsf{ballot},$

$\mathsf{rank}_{\mathsf{commit}}\rangle, QC_{\mathsf{commit}}) = 1$, the leader selects the one with the highest $\mathsf{rank}_{\mathsf{commit}}$. Subsequently, the leader proposes the corresponding ballot in the commit certificate with the highest rank to the parties, accompanied by π_{commit} and $\mathsf{rank}_{\mathsf{commit}}$. Otherwise, the leader uses the initial values received to form ballot. To create a ballot, the leader uses a collision-resistant hash function (Definition 10) to form a hash H_A on the list of parties that the leader accuses of not submitting any votes. The leader also includes the count of the set of parties voted on $0, r$ and the set of parties voted on $1, r$. Furthermore, the leader provides evidence to validate the legitimacy of the generated $\langle \mathsf{count}_0, \mathsf{count}_1, H_A \rangle$ by including a proof π_{ballot}. Let SNRK_1 be a SNARK (see Section 17. The leader P_r constructs π_{ballot} for the public statement:

$$\mathsf{x} = (\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A \rangle) ,$$

The above public NP statement is verified using the following witness:

$$\mathsf{w} = (P_0, P_1, S_0, S_1) ,$$

where P_0 is a vector of public keys, P_1 is another vector of public keys, S_0 is a set of signatures and S_1 is another set of signatures. The verification computation $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, \mathsf{x}, \pi_{\mathsf{ballot}})$ outputs "1" if and only if the following checks are true:

1. $\mathsf{count}_0 = |P_0|$ and $\mathsf{count}_1 = |P_1|$;
2. $P_0 \subseteq N$, $P_1 \subseteq N$ and $P_0 \cap P_1 = \emptyset$;
3. $H_A = \mathsf{Hash}(N - P_0 - P_1)$;
4. S_0 are valid signatures on $\langle 0, r \rangle$ by public keys in P_0;
5. S_1 are valid signatures on $\langle 1, r \rangle$ by public keys in P_1;

Finally, the leader sends $(\langle \mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r} \rangle)$ to all.

If a party receives a ballot proposal from the leader associated with a commit certificate with rank greater than $\mathsf{rank}_{\mathsf{commit}}$, the party propagates the proposal ballot though the expander. Otherwise, if a party receives ballot, the party verifies whether $\mathsf{Hash}(\mathsf{accList}) = H_A$, $|\mathsf{accList}| \leq \min\{r-1, t\}$, $P_i \notin \mathsf{accList}$, and $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, (r, \mathsf{count}_0, \mathsf{count}_1, H_A\rangle), \pi_{\mathsf{ballot}}) = 1$. If the verification is successful, the party propagates ballot through the expander. Since the leader might send a conflicting ballot$'$, we use an $(n, 2\epsilon, 1-2\epsilon)$-expander with constant degree. After propagation, each party checks if received a conflicting ballot$'$ before sending vote $\langle \mathsf{ballot}, r\rangle_{\sigma_i}$ to the leader.

3. *Commit*: If the leader receives at least $n-t$ precommit messages of the form $\langle \mathsf{ballot}, r\rangle_{\sigma_j}$, the leader creates a valid commit certificate $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}}\rangle$ by aggregating the messages received forming QC_{commit} and sends the commit certificate to all parties. If a commit certificate exists, then $n - t > t + 2\epsilon$ parties, of which at least 2ϵ honest parties, must have propagated the leader's ballot proposal to their neighbors. The expansion property implies more than $(1-2\epsilon)n > 2t$, out of which at least $t+1$ honest parties would receive ballot. They would never vote for ballot$' \neq$ ballot, so a conflicting commit certificate cannot exist. Note that accList is not propagated or signed by the parties, nor is it included in commit certificate as it is of size $O(f)$. Had it been included in commit certificate, f the malicious leader could influence honest parties to send the commit certificate to which they are committed in *(pre-commit)*, incurring $O(nf^2)$ communication complexity for $O(f)$ phases. However, the parties store $(\mathsf{ballot}, \mathsf{accList})$ for future purposes. Upon receiving a valid commit certificate from the leader, the party updates commit variables; $\mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}}$, and propagates the commit certificate received through the expander. Consequently, the party sends a matching decide message of $\langle decide, \mathsf{ballot}, r\rangle_{\sigma_i}$ to the leader, where ballot $\in$ commit certificate received from the leader in the previous round. As

the adversary can potentially generate two conflicting commitment certificates in the run using two consecutive malicious leaders, it is not safe to decide on the ballot that the party is committed to. Thus, we introduce an additional layer of protection in the form of another round of voting.

4. ***Decide***: If the leader receives at least $n - t$ decide messages on the same ballot, the leader creates a valid decide certificate by aggregating the messages received to form QC_{decide}, and multicasts $\langle \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r}$ to parties. Once received a valid decide certificate, the party sets $\mathsf{ballot}_i = \mathsf{ballot}$. If stored accList for ballot_i in *(pre-commit)*, $\Psi_i[\mathsf{ballot}_i] \neq \perp$, the party sets $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i]$. Consequently, the party is decided and remains silent.

By the end of the n phases, it is not guaranteed that all parties are decided due to the decreasing threshold. This is why we require the additional ***Processing*** layer. For example, in scenarios where $f = O(t)$ and the initial leaders are honest, it is possible that parties cannot gather sufficient initial values to meet the threshold required for the creation of the ballot. Consequently, the following malicious leader will cause the remaining honest parties, who are designated as leaders for the remaining n phases, to be decided, resulting in their silence during their respective phases. Thus, after the n phases, if an honest party is still undecided, the party sends a help message $\langle help \rangle_{\sigma_j}$ to all parties. Upon receiving a help message, an honest party sends back the ballot decided during Polling, and proof of decision π_{decide} to all parties who sent the help message. As a result, we guarantee that all honest parties are decided before starting the rebuttal protocol. In Lemma 32, we prove that the number of undecided honest parties is upper-bounded by $O(f)$. Thus, the overall communication complexity of this step is $O(n \cdot f)$.

Protocol Π_{Polling} achieves the properties of Theorem 8.

Theorem 8 (Polling)**.** *Assume an execution of protocol* Π_{Polling} *(Figure 5.4), where each party* P_i

inputs value $\mathsf{v}_i \in \{0,1\}$*, and outputs values* $\mathsf{ballot}_i = \langle r, \mathit{count}_0, \mathit{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$ *and* $\mathsf{accList}_i$*, where* $\mathsf{accList}_i$ *is either a subset of* $[n]$ *or* $\perp$*. Then,* Π_{Polling} *satisfies the following properties, except with negligible probability* $\mathsf{negl}(\kappa)$*:*

- ***Agreement****: If an honest party outputs* ballot_i*, no honest party outputs* $\mathsf{ballot}_j \neq \mathsf{ballot}_i$*;*
- ***Correctness****: (i) There exists a set of parties* P_0*, with* $|P_0| = \mathit{count}_0$*, that have signed* $\langle 0, r \rangle$*; (ii) There exists a set of parties* P_1*, with* $|P_1| = \mathit{count}_1$*, that have signed* $\langle 1, r \rangle$*; (iii)* P_0 *and* P_1 *are disjoint; (iv)* H_A *is the collision-resistant hash of the set* $[n] - P_0 - P_1$*; (v)* π_{ballot} *is a SNARK proof that* $\langle r, \mathit{count}_0, \mathit{count}_1, H_A \rangle$ *satisfy properties (i) to (iv);*
- ***Count Validity****: Let* h *be the number of honest parties in the set of parties committed to by* H_A*. If honest parties start with the same value* $\mathsf{v}_i = \mathsf{v}$*, then* $\mathsf{count}_{\mathsf{v}} + h \geq n - t$*;*
- ***Integrity****: If an honest party outputs* $\mathsf{accList}_i \neq \perp$*, then* $\mathsf{accList}_i = [n] - P_0 - P_1$*;*
- ***Reliability****: At least one honest party outputs* $\mathsf{accList}_i \neq \perp$*;*
- ***Termination****: All honest parties terminate.*

Subsequently, we prove that Π_{Polling} incurs overall $O(n \cdot f)$ communication complexity. To do so, we begin by demonstrating that if a leader is honest and undecided in phase $r > f$, all honest parties will reach a decision by the end of this phase. This observation leads us to the conclusion that there are at most $2f + 1$ non-silent phases throughout the run.

Lemma 31. *If an honest leader is undecided in phase* $r > f$*, then all honest parties are decided by the end of phase* r*.*

Proof. Assume some phase $r > f$ and let leader P_r for that phase be honest. In Round 1, P_r, if undecided, sends $\langle \mathsf{value\_req} \rangle$ to all parties. To propose a ballot in *(precommit)*, P_r can either propose a ballot included in a commit message received from some party at the beginning of

the round or create a new ballot and π_{ballot} from the initial values received from parties. If the leader receives some commit message, P_r will propose the respective valid ballot with the greatest respective phase number.

Conversely, by construction, if the leader does not receive any commit message, then no honest party is committed. Therefore, the leader needs to create a new ballot by collecting enough initial values to meet the threshold for the creation of the ballot. Since $r > f$, the leader needs to collect at least $n - r + 1 \leq n - (f + 1) + 1 \leq n - f$ initial values from parties. Since no honest party is committed, then every honest party sends the initial value to the leader. So, the leader will collect at least $n - f$ initial values and will meet the threshold for the creation of the ballot. Consequently, the leader will send a valid ballot proposal to all parties.

Once the honest leader proposes a valid ballot to the parties, all honest parties will vote for it in *(round 3)*. Thus, the leader will collect at least $n - t$ votes to create QC_{commit} and a valid commit certificate $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$, which the leader will broadcast to all parties. Upon receiving a valid commit certificate, every honest party will send to the leader a matching decide message $\langle decide, \mathsf{ballot}, r \rangle$, allowing the leader to collect at least $n - t$ decide messages to form QC_{decide} and a valid decide certificate that the leader broadcasts to all parties. Finally, once an honest party receives a valid decide certificate, sets ballot_i to the received ballot, which is the one the honest leader proposed for the respective commit and the decide certificate. Thus, all honest parties are decided at the end of phase r. □

From Lemma 31, we prove that the number of undecided honest parties before the processing step, and the number of non-silent phases are bounded by $O(f)$.

Lemma 32. *Before processing (Line 28 of Π_{Polling}), at least $n - 2f$ honest parties are decided,*

and at most f honest parties are undecided.

Proof. Given that the leading-based step spans n phases, every honest party has the opportunity to act as a leader during its respective phase, provided that it has not already decided. From Lemma 31, if a leader is honest and undecided in phase $r > f$, every honest party is decided at the end of their phase. Thus, if not all honest parties are decided at the end of Polling, the adversary must have made every honest party that is supposed to be a leader in phases $r > f$ decided by phase f. As there are n phases in the run and the number of malicious parties that could potentially be leaders after phase f is f, at least $n - f - f$ honest parties are decided at the end of Polling. Since there are f malicious parties, at most $n - (n - 2f) - f = f$ honest parties are undecided. □

Lemma 33. *There are at most $2f + 1$ non-silent phases in Π_{Polling}.*

Proof. In Polling, each honest party P_i is the leader during phase i. From Lemma 31, if an honest leader P_r is undecided at the beginning of phase $r > f$, then after phase r all honest parties are decided and every honest party $P_{r'}$ remains silent for phases $r' > r$. Since there are n phases and at most f malicious parties that could be leaders after phase f, there are at most $f + f + 1$ non-silent phases during Π_{Polling}. □

The minimum threshold for the number of initial values required to create a ballot is determined by the phase number, which, in turn, affects the maximum size of the accused list for that phase. This relationship is derived from the fact that the size of the accused list is calculated as the difference between the total number of parties n and the minimum threshold for that phase. By previously proving that there are at most $2f + 1$ non-silent phases, we proceed to prove Lemma 34, which bounds the size of the accused list to $O(f)$.

Lemma 34. *There are at most $2f - 1 = O(f)$ accused parties in* accList.

Proof. The size of accList depends on the phase at which the ballot ballot, which all parties decide, is formed. Let r be the first phase in which ballot is considered valid. Then, by construction, ballot cannot already be accompanied by a commit certificate. Therefore, it must include a valid SNRK_1 proof and must be accompanied by an accused list accList. According to Line 11, an honest party accepts ballot only if the size of accList is at most $n - \max\{n - r + 1, n - t\} = \min\{r - 1, t\}$. If $r \leq f$, $|\mathsf{accList}| < f = O(f)$. Otherwise, if $r > f$, we distinguish two cases based on whether ballot is created by an honest or malicious leader. In phase $r > f$, an honest leader collects at least $n - f \geq \max\{n - r + 1, n - t\}$ initial values, resulting in $|\mathsf{accList}| \leq f$. On the other hand, a malicious leader can have ballot accompanied by accList which includes parties who actually sent their values to the leader. Despite this, due to the constraint of at most $2f + 1$ speaking phases from Lemma 33, a malicious leader must construct the ballot agreed by the parties at most by phase $2f$. Thus, the maximum size of the accList a malicious leader of phase $r \in \{f + 1, \ldots, 2f\}$ can send is limited to at most $\min\{2f - 1, t\}$, else honest parties do not accept accList in Line 11. □

Finally, we use Lemmas 32,33 and 34 to prove that Π_{Polling} incurs $O(n \cdot f)$ communication.

Theorem 9. Π_{Polling} *has a communication complexity of* $O(n \cdot f)$.

5.2.2 The Rebuttal Protocol

The objective of Π_{Rebuttal} (Figure 5.5) is to validate the correctness of the initial tallying process and that no initial values belonging to some parties were unjustly omitted. In Π_{Rebuttal}, parties invoke it with input $(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$, where ballot_i and $\mathsf{accList}_i$ represent the output of Π_{Polling}. If, in the Polling protocol, the malicious ballot creator fails to include the initial values of all honest parties, the honest parties possessing non-empty accusation lists ($\mathsf{accList} \neq \perp$) will send

Algorithm $\Pi_{\mathsf{Polling}}(\mathsf{v}_i)$:

$\mathsf{ballot}_i, \mathsf{accList}_i, \mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}}, \pi_{\mathsf{decide}} = \perp$, $\Psi_i = \emptyset$

Round 1:

1: **if** $\mathsf{ballot}_i = \perp$ **then** $\textsc{Send}([n], \langle \mathsf{value\_req} \rangle_{\sigma_r})$

Round 3:

2: **if** received messages $\langle c_j \rangle_{\sigma_j}$, s.t $\left\{ \begin{array}{c} c_j = \langle \mathsf{ballot}_j, k, QC_{\mathsf{commit}} \rangle \textbf{ and} \\ \mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}_j, k \rangle, QC_{\mathsf{commit}}) = 1 \end{array} \right\}$ **then**

3: $\quad \mathbf{c} = \arg\max_{c_j} \{k\}$

4: $\quad \textsc{Send}([N], (\langle \mathbf{c}.\mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathbf{c}.k, \mathbf{c}.QC_{\mathsf{commit}} \rangle_{\sigma_r}))$

5: **else if** received at least $\max\{n - r + 1, n - t\}$ messages $\langle m_j, r \rangle_{\sigma_j}$, $m_j \in \{0, 1\}$ **then**

6: $\quad P_0 = \{\mathsf{pk}_j\}_{j:m_j=0}, P_1 = \{\mathsf{pk}_j\}_{j:m_j=1}, \mathsf{accList} = N - (P_0 \cup P_1))$

7: $\quad \mathsf{w} = (P_0, P_1, \{\langle 0, r \rangle_{\sigma_j}\}_{j \in P_0}, \{\langle 1, r \rangle_{\sigma_j}\}_{j \in P_1}), \mathsf{x} = (r, |P_0|, |P_1|, \mathsf{Hash}(\mathsf{accList}))$

8: $\quad \pi_{\mathsf{ballot}} = \mathsf{SNRK}_1.\mathsf{Prove}(\mathsf{crs}_1, \mathsf{x}, \mathsf{w})$, $\mathsf{ballot} = \langle r, |P_0|, |P_1|, \mathsf{Hash}(\mathsf{accList}), \pi_{\mathsf{ballot}} \rangle$

9: $\quad \textsc{Send}([N], (\langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r}))$

Round 6:

10: **if** received set S of size $s \geq n - t$ many signatures on $\langle \mathsf{ballot}, r \rangle$ **then**

11: $\quad QC_{\mathsf{commit}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle \mathsf{ballot}, r \rangle, S)$

12: $\quad \textsc{Send}([N], \langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r})$

Round 9:

13: **if** received set DS of size $s \geq n - t$ many signatures on $\langle decide, \mathsf{ballot}, r \rangle$ **then**

14: $\quad QC_{\mathsf{decide}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle decide, \mathsf{ballot}, r \rangle, DS)$

15: $\quad \textsc{Send}([N], \langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r})$

Figure 5.3: Our Π_{Polling} protocol from the view of leader P_r.

accusing messages to each party within accList. In response, upon receiving an accusing message, the honest parties send their initial values to all the other parties. Consequently, honest parties can observe whether the total number of initial values $\mathsf{v}_i = (\mathsf{v}, r)$, where $r \in \mathsf{ballot}$, received during the Rebuttal phase, in addition to $\mathsf{count}_{\mathsf{v}}$ in the agreed ballot, adds to $n - t$ for a $\mathsf{v} \in \{0, 1\}$. If this condition is met, the parties output $\mathsf{y_i} = \mathsf{v}$, and aux_i, which is a SNARK proof π_{v} confirming the existence of $n - t$ votes on v, r during the run. The creation of aux_i involves the use of π_{ballot}, which confirms the existence of $\mathsf{count}_{\mathsf{v}}$ valid signatures, and the additional signatures received during rebuttal. Consequently, we define $\mathsf{ThresholdPredicate}(y_i, \mathsf{aux}_i) = \mathsf{true}$, if aux_i is a valid proof of existence of $n - t$ signatures on v, r. By ensuring the prevention of double counting, we can guarantee that even if accused malicious parties act maliciously and send contradicting initial values to parties, it will not compromise validity. If an honest party receives $n - t$ distinct

Algorithm $\Pi_{\mathsf{Polling}}(\mathsf{v}_i)$:

$\mathsf{ballot}_i, \mathsf{accList}_i, \mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}}, \pi_{\mathsf{decide}} = \perp$, $\Psi_i = \emptyset$

for phase $r = 1$ to n **do**
  leader $\leftarrow P_r \mod n$

Round 2:

  if received $\langle \text{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{ballot}_{\mathsf{commit}} = \perp$ **then** $\textsc{Send}(P_r, \langle \mathsf{v}_i, r \rangle_{\sigma_i})$
  else if received $\langle \text{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{ballot}_{\mathsf{commit}} \neq \perp$ **then**
    $\textsc{Send}(P_r, \langle \mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \rangle_{\sigma_i})$

Round 4:

  if received exactly one message M from the leader **then**
    if $M = \langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle k, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **then**
      if $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, k \rangle, QC_{\mathsf{commit}}) = 1$ **and** $k \geq \mathsf{rank}_{\mathsf{commit}}$ **then**
        $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{ballot}, r \rangle_{\sigma_r})$
    else if $M = \langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r} \rangle$, where $\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$ **then**
      if $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, (r, \mathsf{count}_0, \mathsf{count}_1, H_A), \pi_{\mathsf{ballot}}) = 1$ **and** $pk_i \notin \mathsf{accList}$ **and** $\mathsf{Hash}(\mathsf{accList}) = H_A$ **and** $|\mathsf{accList}| \leq \min\{r-1, t\}$ **then**
        $\Psi_i[\mathsf{ballot}] = \langle \mathsf{accList} \rangle_{\sigma_r}$
        $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{ballot}, r \rangle_{\sigma_r})$

Round 5:

  if sent in (Round 4) **and** not received $\langle \mathsf{ballot}', r \rangle_{\sigma_r}$, s.t. $\mathsf{ballot}' \neq \mathsf{ballot}$ **then**
    $\textsc{Send}(P_r, \langle \mathsf{ballot}, r \rangle_{\sigma_i})$

Round 7:

  if received $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
    $\texttt{temp\_com\_cert} = \langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$
    $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \texttt{temp\_com\_cert} \rangle_{\sigma_r})$

Round 8:

  if $\texttt{temp\_com\_cert} \neq \perp$ **then**
    $\mathsf{ballot}_{\mathsf{commit}} = \texttt{temp\_com\_cert}.\mathsf{ballot}$, $\mathsf{rank}_{\mathsf{commit}} = \texttt{temp\_com\_cert}.r$,
    $\pi_{\mathsf{commit}} = \texttt{temp\_com\_cert}.QC_{\mathsf{commit}}$
    $\textsc{Send}(P_r, \langle decide, \mathsf{ballot}_{\mathsf{commit}}, r \rangle_{\sigma_i})$
  else if received $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
    $\mathsf{ballot}_{\mathsf{commit}} = \mathsf{ballot}$, $\mathsf{rank}_{\mathsf{commit}} = r$, $\pi_{\mathsf{commit}} = QC_{\mathsf{commit}}$

Round 10:

  if received $\langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle decide, \mathsf{ballot}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**
    $\mathsf{ballot}_i = \mathsf{ballot}$, $\pi_{\mathsf{decide}} = r, QC_{\mathsf{decide}}$
    if $\Psi_i[\mathsf{ballot}_i] \neq \emptyset$ **then** $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i].\mathsf{accList}$

Processing (after n phases):

if $\mathsf{ballot}_i = \perp$ **then**
  $\textsc{Send}([N], \langle help \rangle_{\sigma_i})$
for all $\langle help \rangle_{\sigma_j}$ messages received **do**
  $\textsc{Send}(P_j, \langle \mathsf{ballot}_i, \pi_{\mathsf{decide}} \rangle_{\sigma_i})$
if $\mathsf{ballot}_i = \perp$ and received $\langle decide, \mathsf{ballot}_j, r, QC_{\mathsf{decide}} \rangle_{\sigma_j}$ s.t $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**
  $\mathsf{ballot}_i = \mathsf{ballot}_j$
  if $\Psi_i[\mathsf{ballot}_i] \neq \emptyset$ **then** $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i]$

return $(\mathsf{ballot}_i, \mathsf{accList}_i)$

Figure 5.4: Our Π_{Polling} protocol for party P_i.

signatures on the same value, then the signature of at least one honest party is included. If all honest parties start with the same value v, the adversary cannot generate $n - t > t$ signatures on value v', r except with negligible probability for forging signatures. This achievement aligns with the primary goal of Reliable voting, ensuring validity for BA.

Protocol Overview. The algorithm is comprised of three rounds. During the first round, all honest parties possessing a non-null $\mathsf{accList}$ send an accusing message $\langle pk_j, \pi_{j,A} \rangle_{\sigma_r}$ to the accused parties in $\mathsf{accList}$. Here, $\pi_{j,A}$ denotes a SNARK proof constructed by the accusing parties, proving that $pk_j \in \mathsf{accList}$ to parties that might not have knowledge of $\mathsf{accList}$. Let SNRK_2 be a SNARK system as defined in Definition 17. An accusing party constructs $\pi_{j,A}$ for the public statement:

$$\mathsf{x} = (pk_j, H_A)\,,$$

where pk_j is the public key for some party P_j and H_A is the hash of some set. The above public NP statement is verified using the following witness:

$$\mathsf{w} = (\mathsf{accList})\,,$$

where $\mathsf{accList}$ is a set of public keys. The verification computation $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, \mathsf{x}, \pi_{j,A})$ outputs "1" if and only if $H_A = \mathsf{Hash}(\mathsf{accList})$ and $pk_j \in \mathsf{accList}$.

In round 2, parties who receive an accusing message that contains a valid $\pi_{i,A}$, forward their initial value $\langle \mathsf{v}_i, r \rangle_{\sigma_j}$ and the accusing message to all parties. Finally, during round 3, each honest party verifies that each message $\langle \mathsf{v}, r \rangle_{\sigma_j}$, where $r \in \mathsf{ballot}$, it received is accompanied by a valid SNARK proof $\pi_{j,A}$, proving that $P_j \in \mathsf{accList}$. If the messages are deemed valid, the honest party tallies the votes of all accused parties. Specifically, if the sum of (1) the total count of received signatures on v, r and (2) the $\mathsf{count}_\mathsf{v} \in \mathsf{ballot}$ is at least $n - t$ for $\mathsf{v} \in \{0, 1\}$, then the party constructs a proof π_v using SNRK_3, a SNARK system as per Definition 17, indicating the

existence of at least $n-t$ votes on v, r. The party constructs the proof π_{v} for the public statement $\mathsf{x} = (\mathsf{v})$. This public NP statement is verified using the following witness $\mathsf{w} = (\mathsf{votes}_{\mathsf{v}}, \mathsf{ballot})$, where $\mathsf{votes}_{\mathsf{v}}$ is a set of messages. The verification computation $\mathsf{SNRK}_3.\mathsf{Verify}(\mathsf{crs}_3, \mathsf{x}, \pi_{\mathsf{v}})$ outputs "1" if and only if the following checks are true:

- $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A\rangle, \pi_{\mathsf{ballot}}) = 1$ and

  ballot is of the form $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}}\rangle$,
- each $m \in \mathsf{votes}_{\mathsf{v}}$ is of the form $\langle\langle \mathsf{v}, r\rangle_{\sigma_j}, \langle pk_j, \pi_{j,A}\rangle\rangle$,
- for each $m \in \mathsf{votes}_{\mathsf{v}}$, the signature for $\langle \mathsf{v}, r\rangle$ is valid with respect to pk_j,
- for each $m \in \mathsf{votes}_{\mathsf{v}}$: $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, \langle pk_j, H_A\rangle, \pi_{j,A}) = 1$
- $\mathsf{count}_{\mathsf{v}} + |\mathsf{votes}_{\mathsf{v}}| \geq n - t$

The honest party last sets $\mathsf{y_i} = \mathsf{v}$ and $\mathsf{aux}_i = \pi_{\mathsf{v}}$. By the end of Rebuttal, all honest parties are assured of receiving the votes of every honest party in accList that were not included in ballot.

Protocol Π_{Rebuttal} achieves the properties stated in the following theorem.

Theorem 10 (Rebuttal)**.** *Assume an execution of protocol* Π_{Rebuttal} *(Figure 5.5). Each party* P_i *starts with input* $(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$*, where* v_i *is the input to* Π_{Polling}*,* ballot_i *is equal to* $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}}\rangle$ *and* $(\mathsf{ballot}_i, \mathsf{accList}_i)$ *are the outputs of* Π_{Polling}*. Each party outputs value* $\mathsf{y}_i \in \{0,1\}$ *and auxiliary information* aux_i*. Then,* Π_{Rebuttal} *satisfies the following properties, except with negligible probability* $\mathsf{negl}(\kappa)$*:*

- *Provable validity: If all honest parties start with* $\mathsf{v}_i = \mathsf{v}$*, then, all honest parties output* $\mathsf{y}_i = \mathsf{v}$*, and* aux_i *such that* $\mathsf{ThresholdPredicate}(y_i, \mathsf{aux}_i)$ *is true. Moreover, no party outputs* $\mathsf{y}_j \neq \mathsf{y}_i$ *and auxiliary information* aux_j *such that* $\mathsf{ThresholdPredicate}(\mathsf{y}_j, \mathsf{aux}_j)$ *is true.*
- *Termination: All honest parties terminate.*

Algorithm $\Pi_{\mathsf{Rebuttal}}(\mathsf{v}_i, \mathsf{ballot}_i = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}}\rangle, \mathsf{accList}_i)$:

Round 1:

$\mathsf{y_i} \leftarrow \mathsf{v}_i, \mathsf{aux}_i \leftarrow \perp$

1: **for** each $\mathsf{pk}_j \in \mathsf{accList}_i$ **do**

2: $\quad \pi_{j,A} = \mathsf{SNRK}_2.\mathsf{Prove}(\mathsf{crs}_2, (\mathsf{pk}_j, H_A), \mathsf{accList}_i))$

3: $\quad \textsc{Send}(P_j, \langle \pi_{j,A} \rangle_{\sigma_i})$

Round 2:

4: **if** received message $\langle \pi_{i,A} \rangle_{\sigma_j}$ **and** $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, (\mathsf{pk}_i, H_A), \pi_{i,A}) = 1$ **then**

5: $\quad \textsc{Send}([N], (\langle \mathsf{v}_i, r \rangle_{\sigma_i}, \langle \pi_{i,A} \rangle_{\sigma_i}))$

Round 3:

6: **for** each $\mathsf{v} \in \{0, 1\}$ **do**

7: $\quad$ **for** all $(\langle \mathsf{v}, r \rangle_{\sigma_j}, \langle \pi_{j,A} \rangle_{\sigma_j})$ received s.t. $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, (\mathsf{pk}_j, H_A), \pi_{j,A}) = 1$ **do**

8: $\quad\quad \mathsf{votes_v} = \mathsf{votes_v} \cup \{(\langle \mathsf{v}, r \rangle_{\sigma_j}, \langle \pi_{j,A} \rangle_{\sigma_j})\}$

9: $\quad$ **if** $|\mathsf{votes_v}| + \mathsf{count_v} \geq n - t$ **then**

10: $\quad\quad \pi_{\mathsf{v}} = \mathsf{SNRK}_3.\mathsf{Prove}(\mathsf{crs}_3, (\mathsf{v}), (\mathsf{votes_v}, \mathsf{ballot}_i))$

11: $\quad\quad \mathsf{y_i} = \mathsf{v}, \mathsf{aux}_i = \pi_{\mathsf{v}}$

return $\mathsf{y_i}, \mathsf{aux}_i$

Figure 5.5: Code of Π_{Rebuttal} for party P_i.

Further, we prove that Π_{Rebuttal} has $O(n \cdot f)$ communication complexity.

Theorem 11. Π_{Rebuttal} *has* $O(n \cdot f)$ *communication complexity.*

Proof. The protocol involves three communication rounds, with messages exchanged between all parties and the accused parties. By Lemma 34, the number of accused parties is less than $2f$. Consequently, the communication complexity of the rebuttal stage is $O(n \cdot f)$. □

Finally, we demonstrate that the security properties of Π_{Polling} and Π_{Rebuttal} when combined, result in a secure Reliable Voting protocol, Π_{RV}.

Theorem 12 (Reliable Voting from Polling and Rebuttal)**.** *Let* Π_{Polling} *be the protocol of Figure 5.4 and* Π_{Rebuttal} *be the protocol of Figure 5.5.* Π_{RV} *of Figure 5.2 achieves Reliable Voting per Definition 7, for* $t < (1/2 - \epsilon)n$ *with respect to* ThresholdPredicate*, using* $O(n \cdot f)$ *communication.*

Proof. First, from assumption, Π_{Polling} and Π_{Rebuttal} terminate as per Theorem 8 and Theorem 10, thereby ensuring the termination of Π_{RV}.

Now, assume that all honest parties start with the same value v. Then, Π_{Rebuttal} achieves Provable Validity on v and aux_i per Theorem 10, and by result, Π_{RV} achieves Provable Validity as well.

Finally, our proposed approach involves invoking each protocol, namely Π_{Polling} and Π_{Rebuttal}, only once in Π_{RV}. Given our assumptions, Π_{RV} incurs a communication complexity of $O(n \cdot f)$. □

5.3 Weak Byzantine Agreement (WBA)

Our WBA protocol, Π_{WBA} (Figure 5.7), is leader-based and built upon the silent phases paradigm. Our starting point is the WBA protocol in [21], which achieves agreement, termination, and unique validity. However, we achieve a different validity property, the so-called restricted validity. In Π_{WBA}, each party begins with input $(\mathsf{v}_i, \mathsf{aux}_i)$, where the auxiliary information aux_i assists the party in deciding whether to support a leader's proposed value or not. Similar to RV, Our Π_{WBA} protocol is instantiated with respect to the Boolean predicate $\mathsf{ThresholdPredicate}$. Thus, if an honest party starts with $(\mathsf{v}_i, \mathsf{aux}_i)$ where $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$, they are more likely to refrain from voting if the leader proposes a conflicting value v_r unless the leader can prove that $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$. For modularity, we define a predicate $\mathsf{SafeVal}$ that checks if the value proposed by the leader P_r is safe to accept with respect to $\mathsf{v}_i, \mathsf{aux}_i$ with which a party P_i starts. In other words, $\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ is true if $\mathsf{v}_{\mathsf{commit}} = \perp$, where $\mathsf{v}_{\mathsf{commit}}$ is a value that the party is committed to, and either $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ or, $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$.

$\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ is true if $\mathsf{v}_{\mathsf{commit}} = \perp$ **and**

- $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ **or**,
- $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$

Protocol Overview. Π_{WBA} runs for n phases. Each phase consists of four different parts, similar in structure to the parts comprising our Polling protocol.

1. ***Prepare***: If a leader is undecided, they ask for all parties' values through a request message $\langle \mathsf{value\_req} \rangle$. If a committed honest party receives the request message, they forward their commit value $\mathsf{v}_{\mathsf{commit}}$ along with a proof π_{commit} and a commit rank to the leader. If they are not committed, they forward their input $(\mathsf{v}_i, \mathsf{aux}_i)$ to the leader, but only if $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$.

2. ***Pre-commit***: The leader checks all messages sent during the *prepare* round and selects an optimal message that all honest parties will vote for. If the leader receives commit messages, it selects the one with the highest commit rank and proposes it along with the commit proof and rank to the parties. If the leader does not receive any commit message, it chooses a value that satisfies the $\mathsf{SafeVal}$ conditions. If none of these conditions is met, the leader proposes initial value $(\mathsf{v}_i, \mathsf{aux}_i)$. Honest parties sign the leader's proposal if they receive a commit message with a higher rank than the one to which they committed or if the value proposed is safe, as determined by the $\mathsf{SafeVal}$ function. Before sending the vote to the leader, the party propagates the leader's proposal through the expander $\mathcal{G}_{n,\epsilon}$ to ensure that the leader did not send a conflicting value.

3. ***Commit***: If the leader receives $n-t$ votes (signatures) on the leader's proposal, it combines them to form a valid commit proof QC_{commit}. It then forwards the commit certificate $\langle \mathsf{commit\_value}, r, QC_{\mathsf{commit}} \rangle$ to all parties. Upon receiving a valid commit certificate, a party sends a matching decide message to the leader and propagates the commit message through the expander.

4. ***Decide***: Once the leader collects enough ($\geq n - t$) decide messages, the leader forms a valid

decide proof QC_{decide} and decide certificate for v_r, which is then sent to all parties, to decide on.

Lemma 35. *All honest parties are decided within $f+1$ phases.*

Proof. Each undecided honest leader P_r initiates communication during phase r, by sending $\langle \mathsf{value\_req} \rangle$ to all. If an honest party has $\mathsf{v}_{\mathsf{commit}} \neq \perp$, they send their $\mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}$, and π_{commit} to the leader. Otherwise, if an honest party P_j has $\mathsf{ThresholdPredicate}(\mathsf{v}_j, \mathsf{aux}_j) = \mathsf{true}$ and $\mathsf{v}_{\mathsf{commit}} = \perp$, it sends $(\mathsf{v}_j, \mathsf{aux}_j)$ to P_r. As a result, P_r proposes either the highest-rank commit certificate if they receive one, ensuring that all committed parties vote for it (Line 8), or a value v, where $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$, ensuring that all honest parties vote for it if they are not committed. If the leader does not receive either of the above, it proposes its v_i. After receiving a proposed value, each party propagates it to their neighbors to confirm that there are no conflicting values proposed. Since the leader is honest and does not send conflicting values, all honest parties send a matching vote message $\langle \mathsf{v}, r \rangle_{\sigma_i}$, allowing P_r to collect enough $n-t$ votes to form a valid commit certificate, which they forward to all. Then, parties send a matching decide message once they receive a valid commit certificate. This allows P_r to collect enough $n-t$ decide messages to form a valid decide certificate, which they again forward to all parties. Once an honest party sees a valid decide certificate on v, they set $\mathsf{y_i} = \mathsf{v}$ and is decided henceforth. In subsequent phases, honest leaders remain silent and no communication occurs. Since there are f malicious leaders on the run, all honest parties are decided within $f+1$ phases. □

Theorem 13. *Our protocol Π_{WBA} (Figure 5.7), instantiated with $\mathsf{ThresholdPredicate}$, achieves Weak Byzantine Agreement per Definition 8 for $t < (1/2 - \epsilon)n$.*

Finally, we prove the communication complexity of Π_{WBA}.

Algorithm $\Pi_{\mathsf{WBA}}(\mathsf{v}_i, \mathsf{aux}_i)$:

$\mathsf{y}_i, \mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \leftarrow \perp$

(prepare)

Round 1:

1: **if** $\mathsf{y}_i = \perp$ **then**

2: $\quad$ SEND$([n], \langle \mathsf{value_req} \rangle_{\sigma_r})$

(pre-commit)

Round 3:

3: **if** received messages $\langle c_j \rangle_{\sigma_j}$, s.t $c_j = \langle \mathsf{v}_j, k, QC_{\mathsf{commit}} \rangle$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}_j, k \rangle, QC_{\mathsf{commit}}) = 1$ **then**

4: $\quad \mathbf{c} = \arg\max_{c_j} \{k\}$

5: $\quad$ SEND$([n], \langle \mathbf{c}.\mathsf{v}, r \rangle_{\sigma_r}, \langle \mathbf{c}.k, \mathbf{c}.QC_{\mathsf{commit}} \rangle_{\sigma_r}))$

6: **else if** $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$ **or** received $\langle \mathsf{v}_j, \mathsf{aux}_j \rangle_{\sigma_j} : \mathsf{ThresholdPredicate}(\mathsf{v}_j, \mathsf{aux}_j) = \mathsf{true}$ **then**

7: $\quad$ pick any and SEND$([n], (\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle \mathsf{aux}, r \rangle_{\sigma_r}))$

8: **else** SEND$([n], (\langle \mathsf{v}_i, r \rangle_{\sigma_r}), \langle \mathsf{aux}_i, r \rangle_{\sigma_r})$

(commit)

Round 6:

9: **if** received set S of size $s \geq n - t$ many signatures on $\langle \mathsf{v}, r \rangle$ **then**

10: $\quad QC_{\mathsf{commit}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle \mathsf{v}, r \rangle, S)$

11: $\quad$ SEND$([n], \langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_i})$

(decide)

Round 9:

12: **if** received set DS of size $s \geq n - t$ many signatures on $\langle decide, \mathsf{v}, r \rangle$ **then**

13: $\quad QC_{\mathsf{decide}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle decide, \mathsf{v}, r \rangle, DS)$

14: $\quad$ SEND$([n], \langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle_{\sigma_i})$

Figure 5.6: Our Π_{WBA} protocol for leader P_r.

Theorem 14. Π_{WBA} *has* $O(n \cdot f)$ *communication complexity.*

Proof. Each phase requires constantly many steps of communication between the leader and all parties back and forth. Each message transmitted during this process is of size $O(1)$, meaning that each phase incurs $O(n)$ communication. According to Lemma 35, every honest party is decided within $f + 1$ phases. Once an honest party is decided, it stays silent in its leader phase and does not incur any communication. Thus, the protocol has $O(n \cdot f)$ communication. □

Algorithm $\Pi_{\mathsf{WBA}}(\mathsf{v}_i, \mathsf{aux}_i)$:

$\mathsf{y}_i, \mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \leftarrow \perp$

1: **for** phase $r = 1$ to n **do**
2: leader $\leftarrow P_r \mod n$

(prepare)

Round 2:

3: **if** received $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{v}_{\mathsf{commit}} \neq \perp$ **then**
4: $\textsc{Send}(P_r, \langle \mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \rangle_{\sigma_i})$
5: **else if** received $\langle \mathsf{value\_req} \rangle_{\sigma_i}$ **and** $\mathsf{v}_{\mathsf{commit}} = \perp$ **and** $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$ **then**
6: $\textsc{Send}(P_r, \langle \mathsf{v}_i, \mathsf{aux}_i \rangle)$

(pre-commit)

Round 4:

7: **if** received exactly one message M **then**
8: **if** M is of the form $\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle k, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **then**
9: **if** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, k \rangle, QC_{\mathsf{commit}}) = 1$ **and** $k \geq \mathsf{rank}_{\mathsf{commit}}$ **then**
10: $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{v}, r \rangle_{\sigma_r})$
11: **else if** M is of the form $\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle \mathsf{aux}, r \rangle_{\sigma_r}$ **and** $\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ **then**
12: $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{v}, r \rangle_{\sigma_r})$

(Round 5):

13: **if** sent in (Round 4) and not received $\langle \mathsf{v}', r \rangle_{\sigma_r}$ s.t. $\mathsf{v}' \neq \mathsf{v}$ **then**
14: $\textsc{Send}(P_r, \langle \mathsf{v}, r \rangle_{\sigma_i})$

(commit)

Round 7:

15: **if** received $\langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
16: $\texttt{temp\_com\_cert} = \langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle$
17: $\textsc{Send}(\mathcal{G}_{n,\epsilon}, \langle \texttt{temp\_com\_cert} \rangle_{\sigma_r})$

Round 8:

18: **if** $\texttt{temp\_com\_cert} \neq \perp$ **then**
19: $\mathsf{v}_{\mathsf{commit}} = \texttt{temp\_com\_cert}.\mathsf{v}$,$\mathsf{rank}_{\mathsf{commit}} = \texttt{temp\_com\_cert}.r$, $\pi_{\mathsf{commit}} = \texttt{temp\_com\_cert}.QC_{\mathsf{commit}}$
20: $\textsc{Send}(P_r, \langle decide, \mathsf{v}, r \rangle_{\sigma_i})$
21: **if** received $\langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
22: $\mathsf{v}_{\mathsf{commit}} = \mathsf{v}$,$\mathsf{rank}_{\mathsf{commit}i} = r$, $\pi_{\mathsf{commit}} = QC_{\mathsf{commit}}$

(decide)

Round 10:

23: **if** received $\langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle decide, \mathsf{v}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**
24: $\mathsf{y_i} = \mathsf{v}$

return $\mathsf{y_i}$

Figure 5.7: Our Π_{WBA} protocol for party P_i.

Appendix A: Deferred Proofs

A.1 Useful Inequalities

First, we provide the following inequalities, which are standard and of extensive use, and can be easily proven via standard bounds.

Lemma 36 (Bernoulli's inequality)**.** *For every $x \in \mathbb{R}$ and any positive exponent $r > 0$, it holds:*

$$(1+x)^r \leq e^{xr}. \tag{A.1}$$

Lemma 37 (Chernoff Bound)**.** *Assume that $X_1, \ldots, X_n$ are independent $\{0,1\}$ random variables, with $\Pr[X_i = 1] = p_i$. Let $X := \sum_{i=1}^{n} X_i$ and $\mu := \mathbb{E}[X]$. Then, for any $\zeta > 0$, it holds that:*

1. *(Upper tail Chernoff Bound)* $\Pr[X > (1+\zeta)\mu] < \left(\frac{e^{\zeta}}{(1+\zeta)^{1+\zeta}}\right)^{\mu}$,
2. *(Lower tail Chernoff Bound)* $\Pr[X < (1-\zeta)\mu] < \left(\frac{e^{-\zeta}}{(1-\zeta)^{1-\zeta}}\right)^{\mu}$.

Additionally, for any $\zeta \geq 0$, we have

$$\Pr[X \geq (1+\zeta)\mu] \leq e^{-\frac{\zeta^2 \mu}{\zeta+2}}.$$

A.2 Deferred proofs of Chapter 3

Proof of Lemma 5:

Proof. For any set H^*, define E_{H^*} to be the event that $\textsc{AddRandomEdgesAdaptive}_{\mathcal{A}}(V, S, H, m)$

outputs $(\cdot, H^*)$. Also, $\text{supp}(\mathsf{E}) = \{H' : \Pr[\mathsf{E}_{H'}] > 0\}$. Fix $H^* \in \text{supp}(\mathsf{E})$ such that

$$\Pr\left[\sum_{u\in S_2} Z_u \geq 2|H - S_2| \;\middle|\; \mathsf{E}_{H'}\right] \geq \Pr\left[\sum_{u\in S_2} Z_u \geq 2|H - S_2| \;\middle|\; \mathsf{E}_{H^*}\right],$$

for all $H' \in \text{supp}(\mathsf{E})$. Therefore

$$\Pr\left[\sum_{u\in S_2} Z_u \geq 2|H - S_2|\right] = \sum_{H'\in\text{supp}(\mathsf{E})} \Pr\left[\sum_{u\in S_2} Z_u \geq 2|H - S_2| \;\middle|\; \mathsf{E}_{H'}\right] \Pr[\mathsf{E}_{H'}]$$

$$\geq \sum_{H'\in\text{supp}(\mathsf{E})} \Pr\left[\sum_{u\in S_2} Z_u \geq 2|H - S_2| \;\middle|\; \mathsf{E}_{H^*}\right] \Pr[\mathsf{E}_{H'}] \geq \Pr\left[\sum_{u\in S_2\cap H^*} Z_u \geq 2|H - S_2| \;\middle|\; \mathsf{E}_{H^*}\right],$$

since $S_2 \cap H^* \subseteq S_2$. The remaining proof lower-bounds the probability $\Pr[\sum_{u\in S_2\cap H^*} Z_u \geq 2|H - S_2| \mid \mathsf{E}_{H^*}]$. This is done in three steps.

Step 1: Computing the probabilities $\Pr[Z_u = 1 \mid \mathsf{E}_{H^*}]$ *for all* $u \in S_2 \cap H^*$.

For $u \in S_2 \cap H^*$ let Y_u be a random variable such that $Y_u = 1$ iff u has nonzero degree in G. By definition of Z_u ($Z_u = (Y_u = 1) \cap (u \in H')$, for $u \in S_2$) we have that, for all $u \in S_2 \cap H^*$: $Pr[Z_u = 1 \mid \mathsf{E}_{H^*}] = \Pr[Y_u = 1 \mid \mathsf{E}_{H^*}]$.

Suppose now $H^* = H - \{v_1, \ldots, v_\ell\}$, where $v_1, \ldots, v_\ell$ are nodes chosen by the adversary in Line 8 of the experiment. Because the adversary, in his picking $v_1, \ldots, v_\ell$, never accesses information related to nodes in H^* (his access is confined to information in $V - H^*$ through the RevealedEdges list), it follows that E_{H^*} (the event of the adversary picking $v_1, \ldots, v_\ell$) and Y_u (for every $u \in S_2 \cap H^*$) are independent events. Therefore for all $u \in S_2 \cap H^*$ it is

$$\Pr[Z_u = 1 \mid \mathsf{E}_{H^*}] = \Pr[Y_u = 1 \mid \mathsf{E}_{H^*}] = \Pr[Y_u = 1] = 1 - (1 - m/n)^{|S|}.$$

Step 2: Showing $\{Z_u\}_{u\in S_2\cap H^*}$*, conditioned on* E_{H^*}*, are independent.* By using the above findings

and by the independence of Y_u we have that for all $b_u \in \{0,1\}$

$$\Pr\left[\left(\bigcap_{u\in S_2\cap H^*} Z_u = b_u\right) \mid \mathsf{E}_{H^*}\right] = \Pr\left[\left(\bigcap_{u\in S_2\cap H^*} Y_u = b_u\right) \mid \mathsf{E}_{H^*}\right]$$

$$= \Pr\left[\bigcap_{u\in S_2\cap H^*} Y_u = b_u\right] = \prod_{u\in S_2\cap H^*} \Pr[Y_u = b_u] = \prod_{u\in S_2\cap H^*} \Pr[Z_u = b_u \mid \mathsf{E}_{H^*}]$$

and therefore $\{Z_u\}_{u\in S_2\cap H^*}$ are independent conditioned on E_{H^*}.

Step 3: Applying a Chernoff bound. We consider two cases, one for $|S| > 4\epsilon \cdot n/45$ and one for $|S| \leq 4\epsilon \cdot n/45$. For $|S| > 4\epsilon \cdot n/45$, we have that for $u \in S_2 \cap H^*$

$$\Pr[Z_u = 0 \mid \mathsf{E}_{H^*}] = (1 - m/n)^{|S|} < (1 - m/n)^{4\epsilon\cdot n/45} \leq e^{-4\epsilon\cdot m/45},$$

where the last step is derived from the inequality $(1 - x) \leq e^{-x}, \forall x \in \mathbb{R}$ and the fact that $(1 - m/n) > 0$ and $4\epsilon \cdot n/45 > 0$. Note that we have $|H^* - S_2| \leq |H - S_2| \leq \epsilon n/3$ and $|H^*| \geq \epsilon n$. So, by using the set equality $A = (A - B) \cup (A \cap B)$, for $A = H^*$ and $B = S_2$, we get that $|S_2 \cap H^*| \geq 2\epsilon n/3$. Also, $|H - S_2| \leq \epsilon \cdot n/3$, so it is $|S_2 \cap H^*| \geq 2|H - S_2|$. Therefore

$$\begin{aligned}
& \Pr\left[\sum_{u\in S_2\cap H^*} Z_u \geq 2 \cdot |H - S_2| \mid \mathsf{E}_{H^*}\right] \geq \Pr\left[\sum_{u\in S_2\cap H^*} Z_u = |S_2 \cap H^*| \mid \mathsf{E}_{H^*}\right] \\
= \;& \Pr\left[\bigcap_{u\in S_2\cap H^*} Z_u = 1 \mid \mathsf{E}_{H^*}\right] = 1 - \Pr\left[\bigcup_{u\in S_2\cap H^*} Z_u = 0 \mid \mathsf{E}_{H^*}\right] \\
\geq \;& 1 - \sum_{u\in S_2\cap H^*} \Pr\left[Z_u = 0 \mid \mathsf{E}_{H^*}\right] > 1 - |S_2 \cap H^*| \cdot e^{-4\epsilon\cdot m/45} \\
> \;& 1 - n \cdot e^{-4\epsilon\cdot m/45}, \text{ since } |S_2 \cap H^*| < n\,.
\end{aligned}$$

For the case $|S| \leq 4\epsilon\cdot n/45$, since Z_u ($u \in S_2 \cap H^*$) are independent random variables conditioned on E_{H^*} we can use the lower tail Chernoff bound for $Z = \sum_{u\in S_2\cap H^*} Z_u$, i.e.,

$$\Pr[Z < (1 - \delta)\mu \mid \mathsf{E}_{H^*}] < \left(\frac{e^{-\delta}}{(1-\delta)^{1-\delta}}\right)^{\mu}.$$

Now $\mu = \mathbb{E}\left[Z \mid \mathsf{E}_{H^*}\right] = \sum_{u\in S_2\cap H^*} \Pr[Z_u = 1 \mid \mathsf{E}_{H^*}] \geq 2\epsilon \cdot n \cdot (1 - (1 - m/n)^{|S|})/3$, since

$|S_2 \cap H^*| \geq 2\epsilon \cdot n/3$. Also, since $|S| \geq 1$ we have

$$\mu \geq 2\epsilon \cdot n \cdot (1 - (1 - m/n)^{|S|})/3 \geq 2\epsilon \cdot n \cdot (1 - (1 - m/n))/3 = 2\epsilon \cdot m/3\,.$$

For $\delta = 1/2$ this yields $\Pr[Z < \mu/2 \mid \mathsf{E}_{H^*}] < \left(\frac{e^{-0.5}}{(0.5)^{0.5}}\right)^{2\epsilon \cdot m/3} = \left(\frac{e}{2}\right)^{-\epsilon \cdot m/3}$.

Recall however that we must bound the probability $\Pr[Z < 2|H - S_2| \mid \mathsf{E}_{H^*}]$. Therefore it is enough to also show that $\mu \geq 4 \cdot |H - S_2|$. Recall that $|S| \geq 2|H - S_2|/3$ and since $|S| \leq 4\epsilon n/45$, then $|H - S_2| \leq 2\epsilon n/15$. From $\mu \geq 2\epsilon \cdot n \cdot \left(1 - (1 - m/n)^{|S|}\right)/3$ and *by* $1 + x \leq e^x$ *and* $m \leq n$ we have that

$$\begin{aligned}
\mu \quad &\geq \frac{2}{3}\epsilon \cdot n \cdot (1 - e^{-\frac{m}{n}|S|}) \geq 5 \cdot |H - S_2| \cdot \left(1 - e^{\frac{-2m \cdot \epsilon |S|}{15|H - S_2|}}\right), \\
&\textit{since } (n \geq 15|H - S_2|/2\epsilon \textit{ and for } \alpha > 0,\ f(n) = n\left(1 - e^{-\frac{\alpha}{n}}\right) \nearrow \textit{ in } n) \\
&\geq 5 \cdot |H - S_2| \cdot \left(1 - e^{\frac{-4m \cdot \epsilon}{45}}\right), \ \textit{(since } |S|/|H - S_2| \geq 2/3\textit{)} \\
&> 4 \cdot |H - S_2|, \ \ \textit{(since } m \geq 19/\epsilon > \frac{45 \ln 5}{4\epsilon}\textit{)}\,. \quad \square
\end{aligned}$$

Proof of Lemma 9:

Proof. Let $h_\beta \geq \epsilon \cdot n$ be the set of parties that remain honest after the adversary has performed corruptions for round β of $\mathcal{M}$-ConvergeRandom. Thus, for all $\beta \geq 0$, $h_{\beta+1} \subseteq h_\beta$. Also, let R_β denote the set of parties that *i.* remain honest after the adversary has performed corruptions for round β; and *ii.* receive m^* for the first time during round β. Let

$$\begin{aligned}
\omega_\beta \quad &= \left(\bigcup_{i=1}^{\beta} R_i\right) \cap h_\beta = (R_\beta \cap h_\beta) \cup \left(\left(\bigcup_{i=1}^{\beta-1} R_i\right) \cap h_\beta\right) \\
&= R_\beta \cup \left(\left(\bigcup_{i=1}^{\beta-1} R_i\right) \cap h_{\beta-1} \cap h_\beta\right) = R_\beta \cup (\omega_{\beta-1} \cap h_\beta)
\end{aligned}$$

We shall use the above notation for sets interchangeably with the notation for their cardinality, which will be clear depending on the context of the operations. We first prove that with probability $1 - \mathsf{negl}(\kappa)$, for all rounds $\beta \leq \lceil \log(\epsilon \cdot n) \rceil$:

1. If $\omega_{\beta-1} \leq \epsilon \cdot n/3$, then $R_\beta \geq (2/3) \cdot \omega_\beta$ and $\omega_\beta \geq 2^\beta$
2. If $\omega_{\beta-1} > \epsilon \cdot n/3$, then $\omega_\beta = h_\beta \geq \epsilon \cdot n$

(*1.*) In $\mathcal{M}$-ConvergeRandom, a message m^* at round β is propagated in a randomized fashion using $\mathcal{F}_{\mathsf{prop}}$. So, the adversary does not see which party receives the message unless this party is already corrupted—this is exactly what is modeled by AddRandomEdgesAdaptive with the use of RevealedEdges. Thus, since $h_\beta \geq \epsilon \cdot n$, we can compute the number of "new" honest receivers R_β of message m in the end of round β by (almost) directly applying Lemma 5, with S_2 being $h_\beta - \omega_{\beta-1}$ (the set of honest parties that have never received m^* by that round of the protocol) and $R_{\beta-1}$ being S, i.e. the set of honest senders of m^* for round β. Also $R_0 = \omega_0 \geq 1$, since there is at least one honest sender for m^* (namely p) for round 1. We use induction. For $\beta = 1$, if $R_0 \leq \epsilon n/3$, by applying Lemma 5:

$$\Pr[R_1 \geq 2\omega_0] = \Pr[R_1 \geq 2R_0] \geq 1 - p = 1 - \mathsf{negl}(\kappa)\,,$$

where $p = \max\{n \cdot e^{-4\epsilon \cdot m/45}, (\frac{e}{2})^{-\epsilon \cdot m/2}\}$ and since $m = \Theta(\kappa)$ and $\kappa = \mathsf{poly}(n)$. Then, we have that $\omega_1 = R_1 + \omega_0 \geq 3\omega_0 \geq 2$, with probability $1 - \mathsf{negl}(\kappa)$. Also,

$$\frac{\omega_1}{R_1} = \frac{R_1 + \omega_0 \cap h_0}{R_1} = \frac{R_1 + \omega_0}{R_1} \leq \frac{R_1 + R_1/2}{R_1} = 3/2,$$

thus $R_1 \geq (2/3)\omega_1$ with probability $1 - \mathsf{negl}(\kappa)$. Therefore, the base case holds. Let us assume the claim holds for round $\beta - 1 \leq \lceil \log(\epsilon \cdot n) \rceil - 1$. Then, if $\omega_{\beta-1} \leq \epsilon n/3$, we prove that $R_\beta \geq (2/3) \cdot \omega_\beta$ and $\omega_\beta \geq 2^\beta$. By $\mathcal{M}$-ConvergeRandom, all honest receivers in round $\beta - 1$ become senders in round β. So, for applying Lemma 5,we match the sets as follows: $h_{\beta-1} - \omega_{\beta-1} := S_2, h_{\beta-1} := H, h_\beta := H', R_{\beta-1} := S$. Notice that the $u \in S_2$ s.t. $Z_u^{(\beta)} = 1$ are

exactly the $u \in R_\beta$ and thus $\sum_{u \in S_2} Z_u^{(\beta)} = R_\beta$. Also, notice that

$$H - S_2 = h_{\beta-1} - (h_{\beta-1} - \omega_{\beta-1}) = h_{\beta-1} \cap \omega_{\beta-1} = \omega_{\beta-1},$$

$$\text{meaning: } |H - S_2| = \omega_{\beta-1} \leq \epsilon n/3 \text{ and } S = R_{\beta-1} \geq 2\omega_{\beta-1}/3 \geq 2|H - S_2|/3.$$

Finally, we have $m = \Theta(\kappa) \geq 19/\epsilon = \Theta(1)$. Therefore, we can use Lemma 5 for round β: Thus, $\Pr[R_\beta \geq 2|H - S_2|] = \Pr[R_\beta \geq 2\omega_{\beta-1}] \geq 1 - p = 1 - \mathsf{negl}(\kappa)$. Therefore, $\omega_\beta \geq R_\beta \geq 2\omega_{\beta-1} \geq 2 \cdot 2^{\beta-1} = 2^\beta$, with probability $1 - \mathsf{negl}(\kappa)$. Also,

$$\frac{\omega_\beta}{R_\beta} = \frac{R_\beta + \omega_{\beta-1} \cap h_\beta}{R_\beta} \leq \frac{R_\beta + \omega_{\beta-1}}{R_\beta} \leq \frac{R_\beta + R_\beta/2}{R_\beta} = 3/2,$$

thus $R_\beta \geq (2/3)\omega_\beta$, with probability $1 - \mathsf{negl}(\kappa)$, which completes the proof.

(*2.*) For every party $p_i : i = 1, \ldots, n$ we define Bernoulli R.V.s $Z_i^{(\beta)} = 1$ if and only if party p_i received m^* at round β. If $\beta = 1$ and $\omega_0 > \epsilon n/3$, then from the first round there are $\geq \epsilon n/3$ honest senders for m^*. We bound the probability

$$\Pr[\bigcup_{i=1}^{n} \{Z_i^{(1)} = 0\}] \leq \sum_{i=1}^{n} \Pr\left[Z_i^{(1)} = 0\right] = n \cdot \left(1 - \frac{m}{n}\right)^{R_0} \leq n \cdot \left(1 - \frac{m}{n}\right)^{\epsilon n/3}$$
$$= n \cdot \left[\left(1 - \frac{m}{n}\right)^{n/m}\right]^{\epsilon m/3} \leq n \cdot e^{-\epsilon m/3} = \mathsf{negl}(\kappa).$$

Now, if $\beta > 1$, then there was some round $\beta^* \leq \beta$ s.t. $\beta^* \stackrel{\text{def}}{=} \arg\min_{i \geq 0}\{\omega_i > \epsilon n/3\}$. If $\beta^* = 0$, we already showed that all parties receive m^* in the first round. Thus, for all subsequent rounds i, $\omega_i = h_i$. Else, if $\beta^* > 0$, by definition $\omega_{\beta^*-1} \leq \epsilon n/3$. So, we can apply (*1.*), meaning that $R_{\beta^*} \geq 2\omega_{\beta^*}/3 \geq 2\epsilon n/9$. Accordingly, for round β^*, we can again bound the probability

$$\Pr[\bigcup_{i=1}^{n} \{Z_i^{(\beta^*)} = 0\}] \leq n \cdot \left[\left(1 - \frac{m}{n}\right)^{n/m}\right]^{2\epsilon m/9} \leq n \cdot e^{-2\epsilon m/9} = \mathsf{negl}(\kappa),$$

So, for all subsequent rounds $i \geq \beta^*$, $\omega_i = h_i$.

The results from (*1.*),(*2.*) combined mean that, if the protocol runs for more than $\lceil \log \epsilon n/3 \rceil$ rounds without reaching case (*2.*), it is definite that, after applying (*1.*) for round $\beta = \lceil \log(\epsilon n/3) \rceil$, case

(2.) will hold for the next round. Therefore, with probability $1 - \mathsf{negl}(\kappa)$, all remaining honest parties after $\mathcal{M}$-ConvergeRandom terminates have received the message m^*. □

Proof of Theorem 3:

Proof. The communication complexity (CC) of BulletinPBB is dominated by the CC of $\mathcal{M}$-ConvergeRandom. By Theorem 2 each such call invokes $O(n \log n \cdot ms + \sum_{i=1}^{\lceil \log \epsilon n \rceil} |M_p^{(i)} - C_p^{(i)}| \cdot ms)$ communication for each party, where $M_p^{(i)} - C_p^{(i)}$ is the corresponding set of messages input to Propagate. The sets input to Propagate by the protocol are sets containing signatures of size κ each. During each super-round j of the protocol, $\mathcal{M}$-ConvergeRandom is called once, with input set $M_p^{(j)} - C_p^{(j)}$. Due to $\mathcal{M}$-ConvergeRandom, when a message is input by party p to Propagate at some subround, it is subsequently added to C_p and thus won't be input to Propagate again during that super-round. Let for a fixed party p, $I_p^{(j,i)}$ denote the input set $M_p - C_p$ for its call of $\mathcal{M}$-ConvergeRandom at subround i of round j. In BulletinPBB, parties only propagate signatures they have propagated less than twice. Thus, each message can be propagated by the same party at most twice meaning that, for each party p, during all sub-rounds of all super-rounds of the protocol, it holds that $\sum_{j=1}^{(t+1)} \sum_{i=1}^{\lceil \log \epsilon n \rceil} |I_p^{(j,i)}| \leq 2|\mathcal{M}| = O(n^2)$.

We can count the overall CC of each party during the protocol, so we have

$$\mathsf{CC}(p) = \sum_{j=1}^{(t+1)} O(n \log n \cdot m\kappa + \sum_{i=1}^{\lceil \log \epsilon n \rceil} |I_p^{(j,i)}| \cdot m\kappa)$$

$$= O(\sum_{j=1}^{(t+1)} \sum_{i=1}^{\lceil \log \epsilon n \rceil} |I_p^{(j,i)}| \cdot m \cdot \kappa) + O(n^2 \log n \cdot m \cdot \kappa) = \tilde{O}(n^2 \cdot \kappa^2),$$

where, we used the fact that $\sum_{j=1}^{(t+1)} \sum_{i=1}^{\lceil \log \epsilon n \rceil} |I_p^{(j,i)}| = O(n^2)$.Thus, the CC of $\mathcal{M}$-ConvergeRandom for all parties is $\tilde{O}(n^3 \cdot \kappa^2)$. □

A.3 Deferred proofs of Chapter 4

Proof of Lemma 15:

Proof. Fix a PPT adversary $\mathcal{A}$ and let $\varepsilon^{\mathcal{A}}$ be the probability that the committee corruption game outputs 1. Our goal is to upper bound $\varepsilon^{\mathcal{A}}$. We do so by providing a sequence of hybrid games $\mathbf{H}_0, \ldots, \mathbf{H}_4$. For each hybrid $\mathbf{H}_i$, we denote the probability that it outputs 1 when run with adversary $\mathcal{A}$ by $\varepsilon_i^{\mathcal{A}}$.

Hybrid $\mathbf{H}_0$. This is the committee corruption game. To fix notation, we recall it here. The game first generates parameters par via algorithm $\mathsf{CES.Setup}$. Recall that par include the group $\mathbb{G}$ with generator g, a group element $h \leftarrow\!\!\$\, \mathbb{G}$, time-lock parameters tlpar and common reference strings crs and $\tilde{\mathsf{crs}}$ for PS and $\tilde{\mathsf{PS}}$, respectively. Next, the adversary gets par as input and declares its initial set $\mathsf{Corr}_0 \subseteq [n]$ of corrupted parties (with $|\mathsf{Corr}_0| \leq t$. The set of initially honest parties is $\mathsf{Hon}_0 := [n] \setminus \mathsf{Corr}_0$. For each such party $i \in \mathsf{Hon}_0$, the game computes a key pair $(\mathsf{pk}_i, \mathsf{sk}_i)$ via algorithm $\mathsf{CES.Gen}$. To recall, we have $\mathsf{pk}_i = (X_i, \pi_i)$ and $\mathsf{sk}_i = (x_i, r_i)$, where x_i, r_i are uniformly chosen in $\mathbb{Z}_p$, $X_i = g^{x_i} h^{r_i}$, and π_i is a proof for this equality computed using $\mathsf{PS.Prove}$. At any time during the game, $\mathcal{A}$ can corrupt a party i, thereby learning $\mathsf{sk}_i = (x_i, r_i)$. The set Corr gets updated accordingly by inserting i. In the *tag phase*, $\mathcal{A}$ outputs $\mathsf{pretags}_i^{(k)} = (Z_i^{(k)}, (c_{i,j}^{(k)})_{j=1}^n, \tilde{\pi}_i^{(k)})$ for all $i \in [n]$ and all $k \in [n]$. The game continues until $\mathcal{A}$ has corrupted exactly t parties. Let $\mathsf{Hon}^* := [n] \setminus \mathsf{Corr}$ be the set of $n - t$ remaining honest parties. For each $k \in \mathsf{Hon}^*$, the game runs algorithm $\mathsf{CES.TryElect}$. As in this algorithm, let

$$\mathsf{ValidPre}^{(k)} := \{i \in [n] \mid \tilde{\mathsf{PS}}.\mathsf{Ver}(\tilde{\mathsf{crs}}, (Z_i^{(k)}, (c_{i,j}^{(k)})_{j=1}^n), \tilde{\pi}_i^{(k)}) = 1\}$$

and

$$((T_{i,j}^{(k)}, \gamma_{i,j}^{(k)})_{j=1}^{n})_{i \in \mathsf{ValidPre}^{(k)}} := \mathsf{TLP.Solve}(\mathsf{tlpar}, (Z_i^{(k)})_{i \in \mathsf{ValidPre}^{(k)}}).$$

Then, to check the *winning condition*, the game computes

$$\mathsf{id}_k := x_k + \sum_{i \in \mathsf{ValidPre}^{(k)}} T_{i,k}^{(k)}$$

for each $k \in \mathsf{Hon}^*$ as in CES.TryElect. Then, it defines the honest committee members as the remaining honest parties k for which $\mathsf{Pred}(\mathsf{id}_k) = 1$ and CES.VerElect outputs 1, concretely

$$\begin{aligned}\mathsf{HonComm} &= \left\{ k \in \mathsf{Hon}^* \;\middle|\; \mathsf{Pred}(\mathsf{id}_k) = 1 \wedge X_k = g^{x_k} h^{r_k} \wedge \forall i \in \mathsf{ValidPre}^{(k)} :\ g^{T_{i,k}^{(k)}} h^{\gamma_{i,k}^{(k)}} = c_{i,k}^{(k)} \right\} \\ &= \left\{ k \in \mathsf{Hon}^* \;\middle|\; \mathsf{Pred}(\mathsf{id}_k) = 1 \wedge \forall i \in \mathsf{ValidPre}^{(k)} :\ g^{T_{i,k}^{(k)}} h^{\gamma_{i,k}^{(k)}} = c_{i,k}^{(k)} \right\}.\end{aligned}$$

Here, we have used the definition of CES.VerElect, that CES.VerElect and CES.TryElect define the same set $\mathsf{ValidPre}^{(k)}$ when they get the same list $(\mathsf{pretags}_i^{(k)})_{i=1}^n$ as input, and the fact that the X_k have been computed honestly by the game. Finally, the game outputs 1 if $\mathsf{HonComm} = \emptyset$. By definition, we get

$$\varepsilon^{\mathcal{A}} = \varepsilon_0^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_1$. We change how HonComm is defined. Namely, we now define it as

$$\mathsf{HonComm} = \left\{ k \in \mathsf{Hon}^* \mid \mathsf{Pred}(\mathsf{id}_k) = 1 \right\}.$$

Observe that if the definition differs from the definition in the previous hybrid, then there must be an $i \in [n]$ and a $k \in \mathsf{Hon}^*$ such that $i \in \mathsf{ValidPre}^{(k)} \wedge g^{T_{i,k}^{(k)}} h^{\gamma_{i,k}^{(k)}} \neq c_{i,k}^{(k)}$. In particular, for at least one $i^* \in \mathsf{ValidPre}^{(k)}$, it must hold that

$$\exists j \in [n] : g^{\hat{T}_{i^*,j}} h^{\hat{\gamma}_{i^*,j}} \neq c_{i^*,j}^{(j)}$$

where $\hat{T}_{i^*,j}$ and $\hat{\gamma}_{i^*,j}$ are such that $Z_{i^*}^{(k)} \in \mathsf{TLP.Gen}(\mathsf{tlpar}, (\hat{T}_{i^*,j}, \hat{\gamma}_{i^*,j})_{j=1}^n)$. Indeed, if there was no

such i^*, then by completeness of TLP there could never have been such an i. Hence, we can build a PPT reduction that breaks soundness of $\tilde{\mathsf{PS}}$ by outputting the statement $(Z_{i^*}^{(k)}, (c_{i^*,j}^{(k)})_{j=1}^{n})$ and the proof $\tilde{\pi}_{i^*}^{(k)}$ it received from $\mathcal{A}$ in the tag phase. The proof verifies because $i^* \in \mathsf{ValidPre}^{(k)}$. By our assumption that $\tilde{\mathsf{PS}}$ is sound, we get

$$\left|\varepsilon_0^{\mathcal{A}} - \varepsilon_1^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_2$. This hybrid is the same as $\mathbf{H}_1$, but we change the way the common reference string crs and the proofs π_i are generated. Namely, recall that they have been generated in $\mathbf{H}_1$ as

$$\mathsf{crs} \leftarrow \mathsf{PS.Setup}(1^\kappa), \;\; \forall i \in \mathsf{Hon}_0 : \; \pi_i \leftarrow \mathsf{PS.Prove}(\mathsf{crs}, X_i, (x_i, r_i)).$$

Now, in $\mathbf{H}_2$, we generate them as

$$(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{PS.TrapSetup}(1^\kappa), \;\; \forall i \in \mathsf{Hon}_0 : \; \pi_i \leftarrow \mathsf{PS.Sim}(\mathsf{trap}, X_i),$$

where $\mathsf{TrapSetup}$ and Sim are the trapdoor setup algorithm and zero-knowledge simulator guaranteed to exist by the zero-knowledge property. With this, we no longer need the witness (x_i, r_i) to generate the proofs. We can easily bound the difference between using a reduction to the zero-knowledge property of PS. The reduction gets crs from the game as input and simulates $\mathbf{H}_1$ until the proofs π_i have to be generated. Instead of generating them itself, the reduction outputs the statement-witness pairs $(X_i, (x_i, r_i))$ for $i \in \mathsf{Hon}_0$ to the zero-knowledge game and obtains the proofs π_i in return. It continues simulating $\mathbf{H}_1$ and outputs whatever the hybrid would output. Clearly, the reduction is PPT. Further, if $b = 0$ in the zero-knowledge game, then the reduction perfectly simulates $\mathbf{H}_1$, and if $b = 1$ it perfectly simulates $\mathbf{H}_2$. By our assumption that PS is zero-knowledge, we get

$$\left|\varepsilon_1^{\mathcal{A}} - \varepsilon_2^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_3$. We change how to generate element $h \in \mathbb{G}$. Namely, from now on the game samples $\vartheta \leftarrow\!\!\$\, \mathbb{Z}_p^*$ and sets $h := g^{\vartheta}$. Note that $\mathbf{H}_2$ and $\mathbf{H}_3$ only differ if $h = g^0$ in $\mathbf{H}_2$. So we get

$$\left|\varepsilon_2^{\mathcal{A}} - \varepsilon_3^{\mathcal{A}}\right| \leq \frac{1}{p} \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_4$. In this hybrid, we change how the elements X_i are computed. Recall that before, they were computed as $X_i = g^{x_i} h^{r_i}$ for all $i \in \mathsf{Hon}_0$. Also, observe that by the change in $\mathbf{H}_2$, the game only needs (x_i, r_i) when party i is corrupted or to check the winning condition. In this hybrid, we sample $d_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and set $X_i := g^{d_i}$ for all $i \in \mathsf{Hon}_0$. When $\mathcal{A}$ calls the corruption oracle on input i and the has to return (x_i, r_i), it samples $x_i \leftarrow\!\!\$\, \mathbb{Z}_p$ and computes $r_i := (d_i - x_i)/\vartheta$. Note that $\vartheta \neq 0$. Further, let Hon^* be the final set of honest parties when the winning condition is checked, i.e., $\mathsf{Hon}^* := [n] \setminus \mathsf{Corr}$ at the end of the game. For each $i \in \mathsf{Hon}^*$, sample $x_i \leftarrow\!\!\$\, \mathbb{Z}_p$ before checking the winning condition. Note that the joint distribution of the (X_i, x_i, r_i) does not change and so we get

$$\varepsilon_3^{\mathcal{A}} = \varepsilon_4^{\mathcal{A}}.$$

Let us make explicit what we have achieved so far: for every party $k \in \mathsf{Hon}^*$ that is not corrupted when the winning condition is checked, the game first samples $x_k \leftarrow\!\!\$\, \mathbb{Z}_p$, then defines

$$\mathsf{id}_k := x_k + \sum_{i \in \mathsf{ValidPre}^{(k)}} T_{i,k}^{(k)}$$

and then checks if $\mathsf{Pred}(\mathsf{id}_k) = 1$ to see if the party is in the committee. The game outputs 1 if no such party is in the committee. Note that all id_k for $k \in \mathsf{Hon}^*$ are uniform over $\mathbb{Z}_p$ and independent, because the x_k are. Further, we know that $|\mathsf{Hon}^*| = n - t$. We can therefore finish the proof with

$$\varepsilon_4^{\mathcal{A}} \leq \Pr[\mathsf{HonComm} = \emptyset] = \Pr\left[\forall k \in \mathsf{Hon}^* : \mathsf{Pred}(\mathsf{id}_k) = 0\right] = \Pr_{\nu}[\mathsf{Pred}(\nu) = 0]^{n-t} = \mathsf{bias}(\mathsf{Pred}, 0)^{n-t}.$$

□

Proof of Lemma 16:

Proof. Consider a PPT adversary $\mathcal{A}$ running in the large committee game. Let $\varepsilon^{\mathcal{A}}$ be the probability that the large committee game outputs 1. We want to upper bound $\varepsilon^{\mathcal{A}}$. To do so, we provide a sequence of hybrid games $\mathbf{H}_0, \ldots, \mathbf{H}_{10}$. For each hybrid $\mathbf{H}_i$, we denote the probability that it outputs 1 when run with adversary $\mathcal{A}$ by $\varepsilon_i^{\mathcal{A}}$.

Hybrid $\mathbf{H}_0$. This is the large committee game. We recall the game to fix notation. Initially, the game sets up parameters par via algorithm $\mathsf{CES.Setup}$. To recall, par include the group $\mathbb{G}$ with generator g, an element $h \leftarrow\!\!\$\, \mathbb{G}$, time-lock parameters tlpar and common reference strings crs and $\tilde{\mathsf{crs}}$ for PS and $\tilde{\mathsf{PS}}$, respectively. In the *initialization of parties phase*, the adversary gets par as input and declares all public keys $\mathsf{pk}_k = (X_k, \pi_k)$ for all $k \in [n]$, where $X_k \in \mathbb{G}$ and π_k is a proof. In the *tag phase*, the game generates a pretag $\mathsf{pretags}_1 = (Z_1, (c_{1,j})_{j=1}^n, \tilde{\pi}_1)$ via algorithm $\mathsf{CES.Prepare}$. Precisely, by definition of $\mathsf{CES.Prepare}$, these values are computed by sampling $T_{1,j} \leftarrow\!\!\$\, \mathbb{Z}_p$, $\gamma_{1,j} \leftarrow\!\!\$\, \mathbb{Z}_p$, setting $c_{1,j} := g^{T_{i,j}} h^{\gamma_{i,j}}$ for every $j \in [n]$, and computing the puzzle $Z_1 := \mathsf{TLP.Gen}(\mathsf{tlpar}, (T_{i,j}, \gamma_{i,j})_{j=1}^n; \rho_i)$ and the proof $\tilde{\pi}_i \leftarrow \tilde{\mathsf{PS}}.\mathsf{Prove}(\tilde{\mathsf{crs}}, (Z_1, (c_{1,j})_{j=1}^n), ((T_{1,j}, \gamma_{1,j})_{j=1}^n, \rho_1))$ for some random coins ρ_1. The game also sets $\mathsf{pretags}_{1,k} := \mathsf{pretags}_1$ for all $k \in [n]$, and we set $(Z_{1,k}, (c_{1,j,k})_{j=1}^n, \tilde{\pi}_{1,k}) := (Z_1, (c_{1,j})_{j=1}^n, \tilde{\pi}_1)$ accordingly. The adversary $\mathcal{A}$ then gets $\mathsf{pretags}_1$ and within time Δ', it has to output $\mathsf{pretags}_{i,k}$ for all $i \in [n] \setminus \{1\}$ and all $k \in [n]$, where we write $\mathsf{pretags}_{i,k} = (Z_{i,k}, (c_{i,j,k})_{j=1}^n, \tilde{\pi}_{i,k})$. To evaluate the *winning condition*, the game continues running $\mathcal{A}$ until it outputs a set $\mathsf{Committee} \subseteq [n]$ and tickets ticket_k for all $k \in \mathsf{Committee}$. The game then outputs 1 if and only if $|\mathsf{Committee}| > B$ and $\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_k, (\mathsf{pretags}_{i,k})_{i=1}^n, \mathsf{ticket}_k) = 1$ for all $k \in \mathsf{Committee}$. Let us make

explicit how $\mathsf{CES.VerElect}$ works here for such a $k \in \mathsf{Committee}$: The algorithm defines the set

$$\mathsf{ValidPre}_k := \left\{ i \in [n] \;\middle|\; \tilde{\mathsf{PS}}.\mathsf{Ver}(\tilde{\mathsf{crs}}, (Z_{i,k}, (c_{i,j,k})_{j=1}^{n}), \tilde{\pi}_{i,k}) = 1 \right\}.$$

It parses $\mathsf{ticket}_k = (x_k, r_k, (T_{i,k}, \gamma_{i,k})_{i \in \mathsf{ValidPre}_k})$. Then, it outputs 1 if and only if the following four conditions hold:

1. We have $\mathsf{PS.Ver}(\mathsf{crs}, X_k, \pi_k) = 1$.
2. We have $g^{x_k} h^{r_k} = X_k$.
3. For every $i \in \mathsf{ValidPre}_k$, we have $g^{T_{i,k}} h^{\gamma_{i,k}} = c_{i,k,k}$.
4. We have $\mathsf{Pred}(\mathsf{id}_k) = 1$ for $\mathsf{id}_k = x_k + \sum_{i \in \mathsf{ValidPre}_k} T_{i,k}$.

By definition, we get

$$\varepsilon^{\mathcal{A}} = \varepsilon_0^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_1$. We change how the game generates crs. Namely, while it has been generated as $\mathsf{crs} \leftarrow \mathsf{PS.Setup}(1^\kappa)$ in the previous hybrid, from now on it is generated with a trapdoor as $(\mathsf{crs}, \mathsf{trap}) \leftarrow \mathsf{PS.TrapSetup}(1^\kappa)$. We can easily bound the difference between $\mathbf{H}_0$ and $\mathbf{H}_1$ using the zero-knowledge property of PS, see definition 14. We omit giving the reduction formally. We get

$$\left|\varepsilon_0^{\mathcal{A}} - \varepsilon_1^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_2$. We add a step to the game after the *initialization of parties phase*. To understand this additional step, first recall that in the *initialization of parties phase*, the adversary $\mathcal{A}$ outputs keys $\mathsf{pk}_k = (X_k, \pi_k)$ where $X_k \in \mathbb{G}$ and π_k is a proof for all $k \in [n]$. Here is the additional step

that we add in $\mathbf{H}_2$: the game first defines the set[1]

$$\mathsf{Active} := \{k \in [n] \mid \mathsf{PS.Ver}(\mathsf{crs}, X_k, \pi_k) = 1\}\,.$$

It then extracts witnesses from the proofs π_k for $k \in \mathsf{Active}$, i.e.,

$$(\hat{x}_k, \hat{r}_k) \leftarrow \mathsf{PS.Ext}(\mathsf{trap}, X_k, \pi_k) \text{ for all } k \in \mathsf{Active},$$

where trap is the trapdoor introduced in $\mathbf{H}_1$. Further, the game aborts if $X_k \neq g^{\hat{x}_k} h^{\hat{r}_k}$ for some $k \in \mathsf{Active}$. The rest of the game remains unchanged. Note that the two hybrids only differ if the game aborts, and if the game aborts a straightforward reduction can break simulation-extractability (see definition 15) of PS. Therefore, we get

$$\left|\varepsilon_1^{\mathcal{A}} - \varepsilon_2^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_3$. We change how $\tilde{\mathsf{crs}}$ and the proof $\tilde{\pi}_1$ contained in $\mathsf{pretags}_1 = (Z_1, (c_{1,j})_{j=1}^n, \tilde{\pi}_1)$ (as computed in the *tag phase*) are generated. Recall that until now they have been generated as

$$\tilde{\mathsf{crs}} \leftarrow \tilde{\mathsf{PS}}.\mathsf{Setup}(1^\kappa), \;\; \tilde{\pi}_1 \leftarrow \tilde{\mathsf{PS}}.\mathsf{Prove}(\tilde{\mathsf{crs}}, (Z_1, (c_{1,j})_{j=1}^n), ((T_{1,j}, \gamma_{1,j})_{j=1}^n, \rho_1)).$$

From now on, we generate them using the trapdoor setup algorithm $\tilde{\mathsf{PS}}.\mathsf{TrapSetup}$ and the zero-knowledge simulator $\tilde{\mathsf{PS}}.\mathsf{Sim}$ as

$$(\tilde{\mathsf{crs}}, \tilde{\mathsf{trap}}) \leftarrow \tilde{\mathsf{PS}}.\mathsf{TrapSetup}(1^\kappa), \;\; \tilde{\pi}_1 \leftarrow \tilde{\mathsf{PS}}.\mathsf{Sim}(\tilde{\mathsf{trap}}, (Z_1, (c_{1,j})_{j=1}^n)).$$

We can bound the distinguishing advantage of $\mathcal{A}$ between this hybrid and the previous hybrid using the zero-knowledge property of $\tilde{\mathsf{PS}}$, see definition 14. The formal reduction is trivial and omitted. We get

$$\left|\varepsilon_2^{\mathcal{A}} - \varepsilon_3^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

[1] Note that every party that is in the committee has to be in Active, by definition of algorithm CES.VerElect.

Hybrid $\mathbf{H}_4$. In this hybrid, we use the knowledge extractor $\tilde{\mathsf{PS}}.\mathsf{Ext}$ of $\tilde{\mathsf{PS}}$ to extract the tags provided by the adversary. We now make this change more precise: recall that in the *tag phase*, the game obtains pretags $\mathsf{pretags}_{i,k} = (Z_{i,k}, (c_{i,j,k})_{j=1}^n, \tilde{\pi}_{i,k})$ for all $i \in [n] \setminus \{1\}$ and all $k \in [n]$. Here are the additional step that we add in $\mathbf{H}_4$, after receiving these pretags:

For each $k \in [n]$, do the following:

1. Define the set $\mathsf{ValidPre}_k := \{i \in [n] \mid \tilde{\mathsf{PS}}.\mathsf{Ver}(\tilde{\mathsf{crs}}, (Z_{i,k}, (c_{i,j,k})_{j=1}^n), \tilde{\pi}_{i,k}) = 1\}$. Note that this is exactly as algorithm $\mathsf{CES.VerElect}$ will define this set when the game will evaluate the winning condition, see $\mathbf{H}_0$.
2. Define the set $\mathsf{Copied}_k := \{i \in \mathsf{ValidPre}_k \mid (Z_{i,k}, (c_{i,j,k})_{j=1}^n) = (Z_1, (c_{1,j})_{j=1}^n)\}$ and note that $1 \in \mathsf{Copied}_k$.
3. For all $i \in \mathsf{Copied}_k$, set $\hat{T}_{i,k,k} = T_{1,k}$.
4. For all $i \in \mathsf{ValidPre}_k \setminus \mathsf{Copied}_k$, run
$$((\hat{T}_{i,j,k}, \hat{\gamma}_{i,j,k})_{j=1}^n, \hat{\rho}_{i,k}) \leftarrow \mathsf{PS.Ext}(\tilde{\mathsf{trap}}, (Z_{i,k}, (c_{i,j,k})_{j=1}^n), \tilde{\pi}_{i,k}).$$
5. If there is an $i \in \mathsf{ValidPre}_k$ such that $g^{\hat{T}_{i,k,k}} h^{\hat{\gamma}_{i,k,k}} \neq c_{i,k,k}$, abort the game.

Intuitively, this means the game extracts all commitment preimages (i.e., tags) for pretags that have valid proofs and have not been copied by the adversary. The rest of the game remains unchanged for now. Clearly, the output of this hybrid only differs from the previous one if the abort happens. For each fixed $k^* \in [n]$ and $i^* \in [n] \setminus \{1\}$, we bound the probability that $i^* \in \mathsf{ValidPre}_{k^*} \setminus \mathsf{Copied}_{k^*}$ and the abort happens for this pair (k^*, i^*). To this end, we using the simulation-extractability of $\tilde{\mathsf{PS}}$ via the following reduction:

1. The reduction gets $\tilde{\mathsf{crs}}$ as input. It runs the game in $\mathbf{H}_3$ until it has to provide the proof $\tilde{\pi}_1$ to the adversary.

2. To provide $\tilde{\pi}_1$, it outputs the statement $(Z_1, (c_{1,j})_{j=1}^n)$ to the simulation-extractability game. It obtains a simulated proof $\tilde{\pi}_1$ in return.
3. The reduction continues simulating $\mathbf{H}_3$ until in the *tag phase* $\mathcal{A}$ outputs $\mathsf{pretags}_{i,k}$ for all $i \in [n] \setminus \{1\}$ and all $k \in [n]$. It defines $\mathsf{ValidPre}_{k^*}$ and Copied_{k^*} as above and aborts if $i^* \notin \mathsf{ValidPre}_{k^*}$ or $i^* \in \mathsf{Copied}_{k^*}$. Otherwise, it outputs $(Z_{i^*,k^*}, (c_{i^*,j,k^*})_{j=1}^n)$ and $\tilde{\pi}_{i^*,k^*}$ and terminates.

If the abort happens for (k^*, i^*), the reduction breaks simulation-extractability. Further, the reduction is PPT. With a union bound over all pairs pair (k^*, i^*), we get

$$\left|\varepsilon_3^{\mathcal{A}} - \varepsilon_4^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_5$. We change how the winning condition is evaluated: concretely, recall that to evaluate the winning condition, the game obtains a set $\mathsf{Committee} \subseteq [n]$ and tickets of the form $\mathsf{ticket}_k = (x_k, r_k, (T_{i,k}, \gamma_{i,k})_{i \in \mathsf{ValidPre}_k})$ for all $k \in \mathsf{Committee}$ from the adversary. Then, it checks the winning condition as explained in $\mathbf{H}_0$. Especially, it defines

$$\mathsf{id}_k = x_k + \sum_{i \in \mathsf{ValidPre}_k} T_{i,k} \quad \text{for every } k \in \mathsf{Committee}.$$

In $\mathbf{H}_5$, we instead define them as

$$\mathsf{id}_k = \hat{x}_k + \sum_{i \in \mathsf{ValidPre}_k} \hat{T}_{i,k,k} \quad \text{for every } k \in \mathsf{Committee}.$$

To recall, the $\hat{x}_k$ have been extracted in hybrid $\mathbf{H}_2$ and the $\hat{T}_{i,k,k}$ have been extracted in hybrid $\mathbf{H}_4$. We argue that if the outputs of $\mathbf{H}_4$ and $\mathbf{H}_5$ differ, then we can break the discrete logarithm assumption via an efficient reduction. To see this, note that if the outputs differ, then there has to be some $k \in \mathsf{Committee}$ such that:

1. $\hat{x}_k \neq x_k$ or $\hat{T}_{i,k,k} \neq T_{i,k}$ for some $i \in \mathsf{ValidPre}_k$, because the outputs differ.

2. $g^{x_k} h^{r_k} = X_k$ and $g^{T_{i,k}} h^{\gamma_{i,k}} = c_{i,k,k}$ for all $i \in \mathsf{ValidPre}_k$, due to the winning condition, see $\mathbf{H}_0$.

3. $g^{\hat{x}_k} h^{\hat{r}_k} = X_k$ and $g^{\hat{T}_{i,k,k}} h^{\hat{\gamma}_{i,k,k}} = c_{i,k,k}$ for all $i \in \mathsf{ValidPre}_k$, due to the aborts in $\mathbf{H}_2$ and $\mathbf{H}_4$.

From this, it is easy to see that an efficient reduction can find the discrete logarithm of h with respect to basis g. By the discrete logarithm assumption, we get

$$\left|\varepsilon_4^{\mathcal{A}} - \varepsilon_5^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_6$. We change the winning condition making it easier for $\mathcal{A}$ to win. Concretely, as part of algorithm CES.VerElect, the game now no longer checks that $g^{x_k} h^{r_k} = X_k$ and $g^{T_{i,k}} h^{\gamma_{i,k}} = c_{i,k,k}$ for every $i \in \mathsf{ValidPre}_k$. What remains are the checks that $k \in \mathsf{Active}$ and that $\mathsf{Pred}(\mathsf{id}_k) = 1$, where id_k is defined as in the previous hybrid. Note that $\mathbf{H}_5$ and $\mathbf{H}_6$ are *not indistinguishable*, but one can easily see that if $\mathcal{A}$ wins in $\mathbf{H}_5$, then it also wins in $\mathbf{H}_6$ with at least the same probability. Thus, we get

$$\varepsilon_5^{\mathcal{A}} \leq \varepsilon_6^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_7$. We change the winning condition again. Recall that until now, game outputs one if $|\mathsf{Committee}| > B$, $\mathsf{Committee} \subseteq \mathsf{Active}$, and $\mathsf{Pred}(\mathsf{id}_k) = 1$ for all $k \in \mathsf{Committee}$. We now make it easier for the adversary: the game outputs 1 if $|\mathsf{Committee}^*| > B$ for

$$\mathsf{Committee}^* := \left\{k \in \mathsf{Active} \mid \mathsf{Pred}(\mathsf{id}_k) = 1\right\},$$

Again, these two hybrids are *not indistinguishable*, but it is easy to see that

$$\varepsilon_6^{\mathcal{A}} \leq \varepsilon_7^{\mathcal{A}}.$$

Note what we have now achieved: the winning condition of the game can be evaluated as soon as the adversary has sent its puzzles and the game extracted the tags $\hat{T}_{i,j,k}$ as described in hybrid

$\mathbf{H}_4$. This is because the winning condition does not depend on the final output of the adversary any more. Also, recall that the adversary has to send its puzzles in time Δ' after receiving puzzle Z_1. Thus, hybrid $\mathbf{H}_7$ can be run in time

$$\mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Sim}) + \Delta' + n \cdot \mathbf{T}(\tilde{\mathsf{PS}}.\mathsf{Ext}) + t \leq \Delta$$

after puzzle Z_1 is defined, where t is the negligible amount of time it takes to define the sets $\mathsf{ValidPre}_k$ and Copied_k, and to compute the size of $\mathsf{Committee}^*$ given all relevant $\hat{x}_k$ and $\hat{T}_{i,k,k}$. Our next goal is to remove all information the adversary gets about the tags $T_{1,j}$.

Hybrid $\mathbf{H}_8$. We change how the game computes the puzzle Z_1. While so far the game has generated

$$Z_1 := \mathsf{TLP}.\mathsf{Gen}(\mathsf{tlpar}, (T_{1,j}, \gamma_{1,j})_{j=1}^n; \rho_1),$$

we now generate it as

$$Z_1 \leftarrow \mathsf{TLP}.\mathsf{Gen}(\mathsf{tlpar}, \mathbf{0}),$$

were $\mathbf{0}$ is an all-zero string of appropriate length. We can bound the difference between hybrids $\mathbf{H}_7$ and $\mathbf{H}_8$ using the Δ-CPA security of TLP: a reduction would get tlpar from the Δ-CPA game and simulate $\mathbf{H}_7$ for the adversary. To define Z_1, the reduction would output $(T_{1,j}, \gamma_{1,j})_{j=1}^n$ and $\mathbf{0}$ to the Δ-CPA game and get Z_1 in return. Then, it would evaluate the winning condition and return the result to the Δ-CPA game. This works because (1) we do not use ρ_1 elsewhere in $\mathbf{H}_7$, due to the change introduced in hybrid $\mathbf{H}_3$, and (2) the online phase of the reduction, i.e., the time between receiving Z_1 and outputting the output of the game takes time at most Δ, as explained above. By the Δ-CPA security of TLP, we get

$$\left|\varepsilon_7^{\mathcal{A}} - \varepsilon_8^{\mathcal{A}}\right| \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_9$. We change the commitments $c_{1,j}$ for $j \in [n]$ that the game sends in the *tag phase*. So far, these commitments have been generated as

$$\gamma_{1,j} \leftarrow\!\!\$\, \mathbb{Z}_p,\ c_{1,j} := g^{T_{1,j}} h^{\gamma_{1,j}} \text{ for all } j \in [n].$$

From now on, the game samples them at random, i.e., $c_{1,j} \leftarrow\!\!\$\, \mathbb{G}$ for all $j \in [n]$. Note that due to the change in $\mathbf{H}_3$ and the previous hybrid, the values $\gamma_{1,j}$ are only used to generate these commitments and not elsewhere. Hence, assuming h is a generator of $\mathbb{G}$, the distribution does not change. We get

$$\left|\varepsilon_8^{\mathcal{A}} - \varepsilon_9^{\mathcal{A}}\right| \leq \frac{1}{p} \leq \mathsf{negl}(\kappa).$$

Hybrid $\mathbf{H}_{10}$. We change id_k to random. Namely, while id_k has been computed as $\mathsf{id}_k = \hat{x}_k + \sum_{i \in \mathsf{ValidPre}_k} \hat{T}_{i,k,k}$ for every $k \in \mathsf{Active}$ until now, this hybrid instead samples $\mathsf{id}_k \leftarrow\!\!\$\, \mathbb{Z}_p$ for all $k \in \mathsf{Active}$. We claim that this is a purely conceptual change, i.e., the values id_k are already uniform and independent in $\mathbf{H}_9$. To see this, recall the definition of the sets $\mathsf{Copied}_k \subseteq \mathsf{ValidPre}_k$ from $\mathbf{H}_4$ and let $\Gamma_k := |\mathsf{Copied}_k|$ for every $k \in \mathsf{Active}$. Then, in $\mathbf{H}_9$, we get

$$\mathsf{id}_k = \hat{x}_k + \sum_{i \in \mathsf{ValidPre}_k} \hat{T}_{i,k,k} = \hat{x}_k + \Gamma_k \cdot T_{1,k} + \sum_{i \in \mathsf{ValidPre}_k \setminus \mathsf{Copied}_k} \hat{T}_{i,k,k}.$$

By the changes we have made in hybrids $\mathbf{H}_3, \mathbf{H}_8$, and $\mathbf{H}_9$, the adversary obtains no information about the $T_{1,k}$'s and therefore $\hat{x}_k$, Γ_k, and all $\hat{T}_{i,k,k}$ for $i \in \mathsf{ValidPre}_k \setminus \mathsf{Copied}_k$ are independent of $T_{1,k}$. In $\mathbb{Z}_p$, $\Gamma^{(k)}$ is invertible as $1 \leq \Gamma_k < p$, so all id_k's are uniform and independent.

Finally, we bound the probability $\varepsilon_{10}^{\mathcal{A}}$: observe that the game in hybrid $\mathbf{H}_{10}$ outputs 1 if $|\mathsf{Committee}^*| > B$ (see $\mathbf{H}_7$), i.e., there are more than B parties k in Active such that $\mathsf{Pred}(\mathsf{id}_k) = 1$. Due to the changes in the previous hybrids, all id_k are uniform and independent. Hence, we can

conclude the proof with

$$\varepsilon_{10}^{\mathcal{A}} \leq \mathsf{tail}(P, n, B).$$

□

Proof of Lemma 25:

Proof. Let $\mathcal{A}$ be a PPT adversary and $\varepsilon^{\mathcal{A}}$ denote the probability that the graded large committee game outputs 1. We aim to upper bound $\varepsilon^{\mathcal{A}}$. We do so by providing a sequence of hybrid games. For each hybrid $\mathbf{H}_i$, we denote the probability that it outputs 1 when run with adversary $\mathcal{A}$ by $\varepsilon_i^{\mathcal{A}}$.

Hybrid $\mathbf{H}_0$. With this we denote the graded large committee game. Therefore, by definition

$$\varepsilon^{\mathcal{A}} = \varepsilon_0^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_1$. We change *setup*, i.e. how the public keys are distributed. Specifically, recall that in the previous hybrid, for each honest party P_i, the game runs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{CES.Gen}(\mathsf{par})$ and then a gradecast in which party P_i calls $\mathsf{GC}(\mathsf{pk}_i, g^*)$. Then, each honest party P_i would receive an output $(\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})$ from the gradecast with sender P_j. Now, in $\mathbf{H}_1$, the game directly sets $\mathsf{pk}_j^{(i)} := \mathsf{pk}_j$ and $\tilde{g}_j^{(i)} := g^*$ for honest P_i and P_j. Further, for each dishonest sender P_j at that point P_j, recall that honest party P_i obtains $(\mathsf{pk}_j^{(i)}, \tilde{g}_j^{(i)})$ from the gradecast. The game outputs 0 if the values and grades do not satisfy the soundness property of definition 4. Therefore, assuming hybrid $\mathbf{H}_1$ outputs 1, we know that for each such j, we are in one of the following three cases:

- For every honest pair P_k, P_l and dishonest party P_j it holds that $\mathsf{pk}_j^{(k)} = \mathsf{pk}_j^{(l)} =: \mathsf{pk}_j$, or
- there is at least one honest party P_k for which $\tilde{g}_j^{(k)} = 1$. In this case, fix the minimal such k and set $\mathsf{pk}_j := \mathsf{pk}_j^{(k)}$. We know that for every honest party P_l for which $\mathsf{pk}_j^{(l)} \neq \mathsf{pk}_j^{(k)}$ it holds that $\tilde{g}_j^{(l)} = 0$; or

- we have $\tilde{g}_j^{(k)} = 0$ for every honest party P_k. In this case, note that P_j cannot be part of any winning committee. For convenience, let pk_j be a default public key in this case.

For the first and second cases, for every honest party P_i and dishonest party P_j we set $\mathsf{pk}_j^{(i)} = \mathsf{pk}_j$ and $\tilde{g}_j^{(i)} = 1$. The new game is not indistinguishable from $\mathbf{H}_0$ but is at least as likely to output 1, since it defines a superset of cases that satisfy the winning condition. Therefore,

$$\varepsilon_0^{\mathcal{A}} \leq \varepsilon_1^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_2$. We now equalize the honest parties' view on honestly generated pretags: Assuming that hybrid $\mathbf{H}_1$ outputs 1, then for any honest party P_k, we can observe (via lemma 23) that for any moderator P_l (honest or dishonest) one of the following holds, where $\mathsf{pretags}_k$ denotes the value that the game generates when executing Step 1 of $\mathsf{GCES.Toss}$ with respect to honest party P_k:

- For any pair of honest parties P_i, P_j, we have $\mathsf{pretags}_{k,l}^{(i)} = \mathsf{pretags}_{k,l}^{(j)} = \mathsf{pretags}_k$. In that case, set $\mathsf{pretags}_{k,l} := \mathsf{pretags}_k$; or
- For any honest party P_i for which $\mathsf{pretags}_{k,l}^{(i)} \neq \mathsf{pretags}_k$, it holds that $G_l^{(i)} = 0$. In this case, note that P_l cannot be part of winning committee $\mathsf{Committee}^{(i)}$.

We set for all honest parties P_i, P_k and all moderators j: $\mathsf{pretags}_{k,j}^{(i)} = \mathsf{pretags}_k$ and reset $G_j^{(i)} = 1$, if $G_j^{(i)}$ was 0 before. This new game is not necessarily indistinguishable from the previous, but is at least as likely to output 1 (previous winning pretags are still winning now). So, we have

$$\varepsilon_1^{\mathcal{A}} \leq \varepsilon_2^{\mathcal{A}}.$$

Hybrid $\mathbf{H}_3$. We now equalize the honest parties' view on pretags coming from dishonest parties. Therefore, note that assuming hybrid $\mathbf{H}_2$ outputs 1, we know that for any dishonest party P_k and moderator P_l one of the following holds:

- For any pair of honest parties P_i, P_j, we have $\mathsf{pretags}_{k,l}^{(i)} = \mathsf{pretags}_{k,l}^{(j)}$. In that case, set

$\mathsf{pretags}_{k,l} := \mathsf{pretags}^{(i)}_{k,l}$; or

- There is at least one honest party P_i for which $G^{(i)}_{k,l} = 1$. In this case, fix the minimal such i and set $\mathsf{pretags}_{k,l} := \mathsf{pretags}^{(i)}_{k,l}$. We know from the soundness property in lemma 19 that for every honest party P_j for which $\mathsf{pretags}^{(j)}_{k,l} \neq \mathsf{pretags}_{k,l}$ it holds that $G^{(j)}_{k,l} = 0$ and thus $G^{(j)}_{l} = 0$; or
- We have $G^{(i)}_{l} = 0$ for every honest party P_i. In this case, note that P_l cannot be part of any winning committee.

For the first and second cases, for all honest parties P_i and all moderator j, we set $\mathsf{pretags}^{(i)}_{k,j} = \mathsf{pretags}_{k,j}$ and $G^{(i)}_{j} = 1$. This new game is not indistinguishable from the previous one, but is at least as likely to output 1 (previous winning pretags are still winning now). So, we have

$$\varepsilon^{\mathcal{A}}_2 \leq \varepsilon^{\mathcal{A}}_3.$$

Hybrid $\mathbf{H}_4$. We change the winning condition. Recall that the winning condition before was the existence of $i \in \mathsf{Hon}^*$ such that (1) $\forall j \in \mathsf{Committee}^{(i)}$: $\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}^{(i)}_j, (\mathsf{pretags}^{(i)}_{k,j})_{k\in[n]}, \mathsf{ticket}^{(i)}_j) = 1$ and $\tilde{g}^{(i)}_j \geq 1$, $G^{(i)}_j \geq 1$, and (2) $|\mathsf{Committee}^{(i)}| > B$. Now we set the game to output 1 if there exists $i \in \mathsf{Hon}^*$: (1) $\forall j \in \mathsf{Committee}^{(i)}$: $\mathsf{CES.VerElect}(\mathsf{par}, \mathsf{pk}_j, (\mathsf{pretags}_{k,j})_{k\in[n]}, \mathsf{ticket}^{(i)}_j) = 1$, and (2) $|\mathsf{Committee}^{(i)}| > B$. By our previous changes, all grades are set to 1 in $\mathbf{H}_3$ and all pretags are equalized for the honest parties, the two hybrids are equivalent. Thus, we have

$$\varepsilon^{\mathcal{A}}_3 = \varepsilon^{\mathcal{A}}_4.$$

Reduction to the Large Committee Game. One can clearly bound $\varepsilon^{\mathcal{A}}_4$ via a reduction to the large committee game of lemma 16. The reduction internally simulates hybrid $\mathbf{H}_4$ for the adversary, and its goal is to win in the large committee game. We sketch the reduction below.

1. The reduction samples a random party and starts simulating hybrid $\mathbf{H}_4$ for the adversary as

explained below. If at point this party is corrupted, the reduction aborts. For convenience, we denote this party as P_1, to match the notation of the large committee game.

2. The reduction gets as input from the large committee game $\mathsf{par} \leftarrow \mathsf{CES.Setup}(1^\kappa)$. Recall that it has to output public keys pk_i for every $i \in [n]$. To do so, the reduction simulates the *setup* of hybrid $\mathbf{H}_4$ for the adversary. Note that due the change in hybrid $\mathbf{H}_1$, pk_i is defined for every $i \in [n]$. The reduction outputs these pk_i.
3. In the *tag phase* of the large committee game, the reduction gets as input $\mathsf{pretags}_1$ and has to output $n \cdot (n-1)$ pretags, within bounded time $\Delta' = (4 \cdot g^* + 2) \cdot \Delta_r$. To do so, the reduction runs the *toss phase* of the graded large committee game, rigged with $\mathsf{pretags}_1$ specifically for P_1. Recall that the *toss phase* incurs exactly the execution of $\mathsf{GCES.Toss}(\mathsf{par})$, which takes $4 \cdot g^* + 2$ rounds, where each round consists of Δ_r timesteps. Therefore, the reduction receives the outcome of $\mathsf{GCES.Toss}(\mathsf{par})$ within $(4 \cdot g^* + 2) \cdot \Delta_r = \Delta'$ time. Then, for each party P_k, for all $k \in [n]$, the reduction outputs
   (a) $\mathsf{pretags}_{i,k} = \mathsf{pretags}_i$ for each $i \in [n] \setminus \{1\}$ that is honest (per hybrid $\mathbf{H}_2$) and
   (b) $\mathsf{pretags}_{j,k}$ according to hybrid $\mathbf{H}_3$ for each dishonest party P_j.
4. Finally, the reduction has to output $\mathsf{Committee}$ and a ticket ticket_k for all $k \in \mathsf{Committee}$. To do so, the reduction continues running the adversary until it outputs $\mathsf{Committee}^{(1)}$ and tickets $\mathsf{ticket}_k^{(1)}$ for all $k \in \mathsf{Committee}^{(1)}$. The reduction simply sets and outputs $\mathsf{Committee} := \mathsf{Committee}^{(1)}$ and $\mathsf{ticket}_k := \mathsf{ticket}_k^{(1)}$.

Clearly, the reduction is PPT, and as already explained, it satisfies the time constraints of the large committee game. Further, perfectly simulates $\mathbf{H}_4$ for $\mathcal{A}$ assuming it guessed correctly and does not abort. If $\mathbf{H}_4$ outputs 1, then there is at least one honest party for which the winning condition of the large committee game holds. As the view of the adversary does not depend on the guessed

party, we get $\varepsilon_4^{\mathcal{A}} \leq n\varepsilon'$, where ε' is the probability that the reduction wins the large committee game. By lemmas 16 and 18, we get

$$\varepsilon_4^{\mathcal{A}} \leq n\varepsilon' \leq \mathsf{negl}(\kappa) + \mathsf{negl}(\lambda).$$

□

A.4 Deferred proofs of Chapter 5

Proof of Theorem 9:

Proof. To assess the communication complexity of the polling consensus protocol, we analyze the communication for each phase. Each phase consists of four progressions, namely prepare, precommit, commit, and decide.

(prepare) In the first round, the leader for the phase sends a message of size $O(1)$, denoted as $\langle \mathsf{value\_req} \rangle$, to all parties if it is undecided. Honest parties reply (second round) by sending either a commit certificate consisting of their committed ballot $\mathsf{ballot}_{\mathsf{commit}}$ and π_{commit}, both of size $O(1)$, or their initial variable $O(1)$. The total communication complexity for (prepare) is $O(n)$.

(precommit) In the third round, if the leader receives a valid commit certificate from a party, it selects the highest-ranked one and proposes the ballot contained in it, which is of size $O(1)$, to all parties. Otherwise, if the leader gathers enough initial values from the parties to reach the threshold for the creation of a ballot, it creates a ballot (that includes π_{ballot}) and an accussed list and sends them to all parties. The accused list contains its members, which are $(O(f))$ from Lemma 34, resulting to $O(n \cdot f)$ communication for this round. If the leader does not receive a commit certificate or enough initial values to create a new ballot, it remains silent for the rest of

the phase. In the fourth round, if a party receives a valid ballot proposal, it propagates it to its $O(1)$ neighbors using the expander, without transmitting the accused list to avoid increasing the communication complexity. Finally, the honest party sends its vote on the ballot proposed to the leader. In total, the communication complexity is $O(n \cdot f)$ if the leader creates a ballot, and $O(n)$ if the leader proposes a ballot in a received commit certificate.

(commit) In this round, after receiving enough votes to create a valid commit certificate, the leader sends it to all parties, incurring a total communication complexity of $O(n)$. If a party receives a valid commit certificate, it sends a matching decide message $\langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle$ to the leader. Thus, the total communication complexity of this round is $O(n)$.

(decide) Similarly, upon receiving enough votes to create a valid decide certificate, the leader sends the certificate to all parties, incurring a total communication complexity of $O(n)$. Once a leader sends a valid decide certificate, all honest parties are decided, and they will be silent as leaders.

It should be noted that $O(n \cdot f)$ communication complexity in the precommit round is incurred by at most one honest leader in the run. This is because once a valid proposal (ballot) is sent, the leader can collect enough $n - t$ votes to create valid QC_{commit} and QC_{decide}, given that $t < (\frac{1}{2} - \epsilon)n$. Therefore, all honest parties will decide by the end of this honest leader's phase and will not speak in their phases. From Lemma 33, there are at most $2f + 1$ non-silent phases. Thus, the overall communication complexity of the first n phases is $O(n \cdot f) + O(n) \cdot (2f) = O(n(2f + 1)) = O(n \cdot f)$.

(processing) In the three rounds of processing, undecided parties send help messages to all parties. Since the number of undecided honest parties is upper-bounded by f (Lemma 32), at most $2f$

parties send help messages to everyone, whereas the other f could come from malicious parties. In the next round, the decided honest parties send ballot_i and π_{decide} to the parties who sent the help messages. Since both messages are of size $O(1)$, the communication complexity of these three rounds is $O(n \cdot f)$.

Thus, the overall communication complexity of Polling is $O(n \cdot f)$. □

Bibliography

[1] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[2] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[3] Carsten Baum, Emmanuela Orsini, Peter Scholl, and Eduardo Soria-Vazquez. Efficient constant-round MPC with identifiable abort and public verifiability. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 562–592. Springer, Cham, August 2020.

[4] Benoît Libert, Marc Joye, and Moti Yung. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. In Magnús M. Halldórsson and Shlomi Dolev, editors, *33rd ACM PODC*, pages 303–312. ACM, July 2014.

[5] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of bft protocols. In *CCS*, pages 31–42. ACM, 2016.

[6] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, 2019.

[7] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[8] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, jan 1985.

[9] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 1985.

[10] Oded Goldreich. *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge, 2004.

[11] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, pages 990–999, 2006.

[12] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *20th ACM STOC*, pages 148–161. ACM Press, May 1988.

[13] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

[14] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. 2019.

[15] Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In *35th International Symposium on Distributed Computing (DISC 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2021.

[16] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Sublinear-round byzantine agreement under corrupt majority. In Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas, editors, *PKC 2020, Part II*, volume 12111 of *LNCS*, pages 246–265. Springer, Cham, May 2020.

[17] Daniel Collins, Sisi Duan, Julian Loss, Charalampos Papamanthou, Giorgos Tsimos, and Haochen Wang. Stowards optimal parallel broadcast under a dishonest majority. Cryptology ePrint Archive, Paper 2024/974, 2024. `https://eprint.iacr.org/2024/974`.

[18] Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 412–456. Springer, Cham, November 2020.

[19] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. Expected constant round byzantine broadcast under dishonest majority. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 381–411. Springer, Cham, November 2020.

[20] Atsuki Momose, Ling Ren, Elaine Shi, Jun Wan, and Zhuolun Xiang. On the amortized communication complexity of byzantine broadcast. Cryptology ePrint Archive, Paper 2023/038, 2023. `https://eprint.iacr.org/2023/038`.

[21] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Make Every Word Count: Adaptive Byzantine Agreement with Fewer Words. *Leibniz International Proceedings in Informatics (LIPIcs)*, 253, 2023.

[22] Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the o $(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, pages 319–330, 2021.

[23] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In Andréa W. Richa and Rachid Guerraoui, editors, *29th ACM PODC*, pages 420–429. ACM, July 2010.

[24] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *17th SODA*, pages 990–999. ACM-SIAM, January 2006.

[25] Silvio Micali. Very simple and efficient byzantine agreement. In Christos H. Papadimitriou, editor, *ITCS 2017*, volume 4266, pages 6:1–6:1, 67, January 2017. LIPIcs.

[26] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. Technical report, MIT, 2017.

[27] Vitalik Buterin. A guide to 99% fault tolerant consensus. Blog Post, 2018.

[28] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380. Springer, Cham, November 2020.

[29] Matthias Fitzi and Jesper Buus Nielsen. On the number of synchronous rounds sufficient for authenticated byzantine agreement. In *International Symposium on Distributed Computing*, pages 449–463. Springer, 2009.

[30] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th FOCS*, pages 658–668. IEEE Computer Society Press, October 2007.

[31] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In *PODC*, pages 27–30, 1983.

[32] Michael O Rabin. Randomized byzantine generals. In *SFCS*, pages 403–409. IEEE, 1983.

[33] Matthias Fitzi and Juan A Garay. Efficient player-optimal protocols for strong and differential consensus. In *PODC*, pages 211–220, 2003.

[34] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Berlin, Heidelberg, August 2006.

[35] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. Gossiping for communication-efficient broadcast. In Yevgeniy Dodis and Thomas Shrimpton, editors, *CRYPTO 2022, Part III*, volume 13509 of *LNCS*, pages 439–469. Springer, Cham, August 2022.

[36] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Communication and round efficient parallel broadcast protocols. Cryptology ePrint Archive, Paper 2023/1172, 2023. `https://eprint.iacr.org/2023/1172`.

[37] Gilad Asharov and Anirudh Chandramouli. Perfect (parallel) broadcast in constant expected rounds via statistical vss. Cryptology ePrint Archive, Paper 2024/376, 2024. https://eprint.iacr.org/2024/376.

[38] Zygmunt J Haas, Joseph Y Halpern, and Li Li. Gossip-based ad hoc routing. *IEEE/ACM Transactions on networking*, pages 479–491, 2006.

[39] Alan J. Demers, Daniel H. Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard E. Sturgis, Daniel C. Swinehart, and Douglas B. Terry. Epidemic algorithms for replicated database maintenance. In Fred B. Schneider, editor, *6th ACM PODC*, pages 1–12. ACM, August 1987.

[40] Edward Bortnikov, Maxim Gurevich, Idit Keidar, Gabriel Kliot, and Alexander Shraer. Brahms: byzantine resilient random membership sampling. In Rida A. Bazzi and Boaz Patt-Shamir, editors, *27th ACM PODC*, pages 145–154. ACM, August 2008.

[41] Dahlia Malkhi, Yishay Mansour, and Michael K Reiter. On diffusing updates in a byzantine environment. In *Proc. of the IEEE Symposium on Reliable Distributed Systems*, pages 134–143, 1999.

[42] Christian Matt, Jesper Buus Nielsen, and Søren Eller Thomsen. Formalizing delayed adaptive corruptions and the security of flooding networks. *Cryptology ePrint Archive*, 2022.

[43] Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, and Dragos-Adrian Seredinschi. Scalable Byzantine Reliable Broadcast. In *Proc. of DISC*, volume 146, pages 22:1–22:16, 2019.

[44] Ran Cohen, Juan Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In *CRYPTO 2023*, 2023.

[45] Alexander Spiegelman. In search for a linear byzantine agreement. *CoRR*, abs/2002.06993, 2020.

[46] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International Conference on Financial Cryptography and Data Security*, pages 296–315. Springer, 2022.

[47] Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, and Alberto Sonnino. State machine replication in the libra blockchain. In *The Diem Association*, 2019.

[48] Danny Dolev, Ruediger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, oct 1990.

[49] Danny Dolev and Christoph Lenzen. Early-deciding consensus is expensive. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, page 270–279, New York, NY, USA, 2013. Association for Computing Machinery.

[50] Ittai Abraham and Danny Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity, 2015.

[51] Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science*, pages 313–321. Springer, 1992.

[52] Juan A Garay and Yoram Moses. Fully polynomial byzantine agreement in t+ 1 rounds. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 31–41, 1993.

[53] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous Byzantine agreement with expected $o(1)$ rounds, expected $o(n^2)$ communication, and optimal resilience. In *FC*, pages 320–334. Springer, 2019.

[54] Jeffrey Considine, Matthias Fitzi, Matthew Franklin, Leonid A Levin, Ueli Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *Journal of Cryptology*, 18(3):191–217, 2005.

[55] Andrea Flamini, Riccardo Longo, and Alessio Meneghetti. Multidimensional byzantine agreement in a synchronous setting. *Applicable Algebra in Engineering, Communication and Computing*, pages 1–19, 2022.

[56] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical report, Massachusetts Institute of Technology, 1996.

[57] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *ITCS 2016*, pages 345–356. ACM, January 2016.

[58] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 620–649. Springer, Cham, August 2019.

[59] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabien Laguillaumie, and Giulio Malavolta. Efficient CCA timed commitments in class groups. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021*, pages 2663–2684. ACM Press, November 2021.

[60] Aydin Abadi and Aggelos Kiayias. Multi-instance publicly verifiable time-lock puzzle and its applications. In Nikita Borisov and Claudia Díaz, editors, *FC 2021, Part II*, volume 12675 of *LNCS*, pages 541–559. Springer, Berlin, Heidelberg, March 2021.

[61] Yi Liu, Qi Wang, and Siu-Ming Yiu. Towards practical homomorphic time-lock puzzles: Applicability and verifiability. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022, Part I*, volume 13554 of *LNCS*, pages 424–443. Springer, Cham, September 2022.

[62] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. Cryptology ePrint Archive, Paper 2023/1582, 2023. `https://eprint.iacr.org/2023/1582`.

[63] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri Aravinda Krishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 554–584. Springer, Cham, May 2023.

[64] Poulami Das, Lisa Eckey, Sebastian Faust, Julian Loss, and Monosij Maitra. Round efficient byzantine agreement from VDFs. Cryptology ePrint Archive, Report 2022/823, 2022.

[65] Poulami Das, Lisa Eckey, Sebastian Faust, Julian Loss, and Monosij Maitra. Round efficient byzantine agreement from vdfs. In *International Conference on Security and Cryptography for Networks*, pages 139–160. Springer, 2024.

[66] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *The Computer Journal*, 49(1):82–96, 2006.

[67] Achour Mostéfaoui and Michel Raynal. Signature-free asynchronous byzantine systems: from multivalued to binary consensus with t< n/3, o(n^2) messages, and constant time. *Acta Informatica*, 54(5):501–520, 2017.

[68] Russell Turpin and Brian A Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[69] Chaya Ganesh and Arpita Patra. Broadcast extensions with optimal communication and round complexity. In *PODC*, pages 371–380, 2016.

[70] Guanfeng Liang and Nitin Vaidya. Error-free multi-valued consensus with byzantine failures. In *PODC*, pages 11–20, 2011.

[71] Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. Efficient adaptively-secure byzantine agreement for long messages. In *ASIACRYPT*, 2022.

[72] Xueqing Gong and Chi Wan Sung. Zigzag decodable codes: Linear-time erasure codes with applications to data storage. *Journal of Computer and System Sciences*, 89:190–208, 2017.

[73] James S Plank and Lihao Xu. Optimizing cauchy reed-solomon codes for fault-tolerant network storage applications. In *NCA*, pages 173–180. IEEE, 2006.

[74] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. *DISC*, 2020.

[75] Shir Cohen, Rati Gelashvili, Lefteris Kokoris Kogias, Zekun Li, Dahlia Malkhi, Alberto Sonnino, and Alexander Spiegelman. Be aware of your leaders. In *International Conference on Financial Cryptography and Data Security*, pages 279–295. Springer, 2022.

[76] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[77] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, page 165–175, 2021.

[78] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 2705–2718, 2022.

[79] Alexander Spiegelman, Balaji Aurn, Rati Gelashvili, and Zekun Li. Shoal: Improving dag-bft latency and robustness. *arXiv preprint arXiv:2306.03058*, 2023.

[80] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1187–1201, 2022.

[81] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. {DispersedLedger}:{High-Throughput} byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

[82] Sam Blackshear, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Xun Li, Mark Logan, Ashok Menon, Todd Nowacki, Alberto Sonnino, et al. Sui lutris: A blockchain combining broadcast and consensus. Technical report, Technical Report. Mysten Labs. https://sonnino. com/papers/sui-lutris. pdf, 2023.

[83] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag. *arXiv preprint arXiv:2208.00940*, 2022.

[84] The Aptos team. `https://aptoslabs.com`, 2023.

[85] The Sui team. `http://sui.io`, 2023.

[86] Giorgos Tsimos, Anastasios Kichidis, Alberto Sonnino, and Lefteris Kokoris-Kogias. Hammerhead: Leader reputation for dynamic scheduling. In *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, pages 1377–1387. IEEE, 2024.

[87] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[88] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, pages 191–201. IEEE, 2005.

[89] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In Alessia Milani and Philipp Woelfel, editors, *PODC*, pages 399–417. ACM, 2022.

[90] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, 1999.

[91] Andreea B. Alexandru, Julian Loss, Charalampos Papamanthou, Giorgos Tsimos, and Benedikt Wagner. Sublinear-round broadcast without trusted setup. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4132–4171.

[92] Fatima Elsheimy, Giorgos Tsimos, and Charalampos Papamanthou. Deterministic byzantine agreement with adaptive $O(n \cdot f)$ communication. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1120–1146, 2024.

[93] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. *J. Cryptol.*, 17(4):297–319, sep 2004.

[94] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *EUROCRYPT 2000*, volume 1807 of *LNCS*, pages 207–220. Springer, Berlin, Heidelberg, May 2000.

[95] J. Katz and Y. Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC Cryptography and Network Security Series. Taylor & Francis, 2014.

[96] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, abhi shelat, and Elaine Shi. C∅c∅: A framework for building composable zero-knowledge proofs. Cryptology ePrint Archive, Report 2015/1093, 2015.

[97] Jens Groth. On the size of pairing-based non-interactive arguments. pages 305–326, 05 2016.

[98] Johannes Buchmann and Safuat Hamdy. A survey on iq cryptography. In *Public-Key Cryptography and Computational Number Theory*, pages 1–15, 2011.

[99] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.