Optimal Load-Balanced Scalable Distributed Agreement *

Yuval Gelles† Ilan Komargodski‡

Abstract

We consider the fundamental problem of designing classical consensus-related distributed abstractions for large-scale networks, where the number of parties can be huge. Specifically, we consider tasks such as Byzantine Agreement, Broadcast, and Committee Election, and our goal is to design *scalable* protocols in the sense that each honest party processes[1] and sends a number of bits which is sub-linear in n, the total number of parties.

In this work, we construct the first such scalable protocols for all of the above tasks. In our protocols, each party processes and sends $\tilde{O}(\sqrt{n})$ bits throughout $\tilde{O}(1)$ rounds of communication, and correctness is guaranteed for at most $1/3 - \epsilon$ fraction of static byzantine corruptions for every constant $\epsilon > 0$ (in the full information model). All previous protocols for the considered agreement tasks were non-scalable, either because the communication complexity was linear or because the computational complexity was super polynomial.

We complement our result with a matching lower bound showing that any Byzantine Agreement protocol must have $\Omega(\sqrt{n})$ complexity in our model. Previously, the state of the art was the well-known $\tilde{\Omega}(\sqrt[3]{n})$ lower bound of Holtby, Kapron, and King (Distributed Computing, 2008).

*This research is supported in part by an Alon Young Faculty Fellowship, by a JPM Faculty Research Award, by a grant from the Israel Science Foundation (ISF Grant No. 1774/20), and by a grant from the US-Israel Binational Science Foundation and the US National Science Foundation (BSF-NSF Grant No. 2020643).

†School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel. Email: yuval.gelles@mail.huji.ac.il.

‡School of Computer Science and Engineering, Hebrew University of Jerusalem, Israel, and NTT Research. Email: ilank@cs.huji.ac.il. Incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University.

[1]For an incoming message to be "processed" its content needs to be read. Reading its metadata (sender's info and length) is done for free. This standard modeling is known as "static filtering" in the literature.

Contents

1 Introduction **1**
- 1.1 Our Main Results . . . 1
- 1.2 Prior Work and Relevant Background . . . 2
- 1.3 Applications of Scalable Agreement . . . 3

2 Overview of Our Techniques **5**
- 2.1 Restricting Polled Lists . . . 6
- 2.2 Enforcing Poll Lists with the Prescribed Structure . . . 8
- 2.3 Putting Things Together for Everywhere Agreements . . . 9
- 2.4 From Agreement to Byzantine Agreement, Broadcast, and More . . . 11
- 2.5 The Lower Bound . . . 11

3 The Model **12**

4 Restricting Poll Lists and Enforcing Consistency **14**
- 4.1 Generating a Quorum (the H Mapping) . . . 14
- 4.2 Choosing Poll Lists (the J Mapping) . . . 17

5 The Scalable Agreement Protocol **19**
- 5.1 Efficiency . . . 21
- 5.2 Correctness . . . 22
- 5.3 Distribute Poll Requests Sub-Protocol . . . 24
  - 5.3.1 Correctness . . . 25
  - 5.3.2 Efficiency . . . 26

6 Multi-Party Computation, Byzantine Agreement, Broadcast, and More **26**
- 6.1 Realizing the Quorum Model . . . 27
- 6.2 Applications in the Quorum Model . . . 27

7 An $\Omega(\sqrt{n})$ Lower Bound for Byzantine Agreement **29**
- 7.1 Proof of the Bottleneck Lemma (Lemma 4) . . . 30

References **37**

1 Introduction

We consider the basic problem of designing scalable protocols for classical agreement abstractions such as Byzantine Agreement, Broadcast, and Committee or Leader Election [PSL80, LSP82]. These abstractions are fundamental in many central distributed applications, including blockchain protocols (e.g., Algorand [GHM+17, CM19]) and secure multiparty computation (MPC) protocols [GMW87, BGW88, CCD87, RB89].

Understanding the complexity of the above abstractions is the subject of a rich line of research. In this work, we are interested in building protocols for the above abstractions that fit large-scale networks, where a very large number of participants needs to be supported. That is, we want protocols that are *scalable*: the complexity of each party should scale sub-linearly with the number of parties. Specifically, we want protocols with the following properties:

- **Correctness (agreement)**: all (honest) parties terminate and agree on the output of the protocol (except with negligible probability of error).
- **Efficiency (scalability)**: the next-message function for each (honest) party is given by a uniform sub-linear time procedure (in n, the number of parties). In particular, the space, computational, and communication complexities of each party are sub-linear.

  We follow the standard modeling of protocols with *static message filtering*, where every (honest) party must decide on the set of parties it will listen to before the beginning of each round (as a function of its internal view at the end of the previous round). Only messages from these parties are fed into the next-message function.[2]

Agreement protocols should satisfy the above properties even in the presence of an attacker that controls some of the parties and acts in their name. We consider a computationally unbounded adversary that corrupts parties statically (after the protocol is specified but before the protocol starts). Further, the adversary has full information (i.e., it sees all messages sent, even messages sent between two honest parties), and the adversary is rushing (i.e., it gets to send its messages after seeing the honest parties' messages for that round). Communication happens via synchronous rounds of communication and via point-to-point channels. The precise details of the model are given in Section 3.

1.1 Our Main Results

We provide the first protocols for all of the above-mentioned agreement tasks satisfying the above correctness and efficiency properties. Throughout this paper, we use the $\tilde{O}(\cdot), \tilde{\Omega}(\cdot)$ notation to suppress multiplicative factors that depend poly-logarithmically on the number of parties, n.

Theorem 1 (Scalable distributed applications; Informal)**.** *There are protocols for byzantine agreement, broadcast, and committee or leader election that reach agreement (correctness) with all but negligible probability of error. Each of these protocols is secure against an adversary that statically corrupts up to $1/3 - \epsilon$ fraction of parties for any constant $\epsilon > 0$. Furthermore, each of these protocols terminates within $\tilde{O}(1)$ rounds, each party sends $\tilde{O}(\sqrt{n})$ bits overall, and the next-message procedure can be described by a uniform $\tilde{O}(\sqrt{n})$-size circuit.*

[2]The idea is that in a large network every party is connected via a router that, based on hard-coded message filtering rules, decides whether to forward the message to its corresponding party or not. Message filtering should be done via a simple and "lightweight" test. Concretely, whether a message is forwarded or dropped should be a deterministic function of the incoming message's metadata, which includes the sender's identity and the message's size.

We complement the above result with a tight lower bound showing that our result is essentially optimal.

Theorem 2 (Lower bound; Informal)**.** *For any protocol that computes byzantine agreement with probability at least* $2/3$ *and is secure against adversaries that statically corrupt* t *parties, there must be an honest party that sends or processes at least* $\Omega(t/\sqrt{n})$ *messages.*

1.2 Prior Work and Relevant Background

We provide more context and compare our results with known ones. For concreteness, we shall use the byzantine agreement task (BA) as our running example, mostly because it is probably the most classical abstraction. Still, the state of affairs applies to all the other distributed agreement tasks that we consider like broadcast and committee or leader election.

Towards scalable agreement. For several decades since the seminal work of Lamport, Shostak, and Pease [LSP82] that introduced the BA problem, all protocols required a quadratic number of messages; essentially, every party had to communicate with every other party (e.g., [DS83, DLS88, CL99]). The breakthrough result of King, Saia, Sanwalani, and Vee [KSSV06] was the first instance of a scalable agreement protocol. Specifically, their protocol succeeds with all but negligible probability of error and each party speaks to only $\tilde{O}(1)$ other parties (within $\tilde{O}(1)$ many rounds). It can tolerate $1/3-\epsilon$ fraction of statically corrupted parties for every constant $\epsilon > 0$.[3] However, it **fails to achieve correctness**. It only achieves so-called *almost-everywhere* agreement, where $1 - O(\log^{-1} n)$ fraction of the parties reach agreement [DPPU88]. Extending almost-everywhere to full agreement (while preserving the efficiency properties) has been a major challenge since then.

In the following years, several attempts at this challenge were made. King and Saia [KS09] presented a protocol that satisfies correctness (i.e., full agreement), but **it is not scalable** since there are (few) parties that communicate essentially with everyone. In other words, their communication pattern is highly unbalanced. Several follow up works (e.g., [BGH13, ACD+19]) suffer from the same issue.[4]

King, Lonargan, Saia, and Trehan [KLST11] solved the unbalanced issue of [KS09] at the cost of space and computational inefficiency. Specifically, while in their protocol every party communicates only $\tilde{O}(\sqrt{n})$ bits within $\tilde{O}(1)$ rounds, their protocol's next-message function requires space and computational complexities that scale super-polynomially with n.[5] Thus, their protocol **is not computationally efficient**, let alone scalable. We mention that this protocol has another drawback: to determine whether an incoming message needs to be processed, the needed storage scales (at least) linearly with n; that is, it is not in the model of static filtering (same applies to [KS09]; see below).

Our Theorem 1 achieves correctness (i.e., full agreement) and scalability, and thereby strictly improves upon the above works [KS09, KLST11]. All of the above is summarized in Table 1.

[3] The line of works on scalable agreement, including the current work, considers the near-optimal resilience range, i.e., up to $(1/3-\epsilon)$ fraction of corruptions. It is impossible to tolerate $n/3$ corruptions [LSP82, FLM86, Bor96], even ignoring efficiency considerations (unless further assumptions are made such as some form of trusted setup or various limitations on the adversary).

[4] Intuitively, all of these protocols first elect a small "central committee" and use it to disperse information to all other parties quickly—the committee members typically communicate with all other parties.

[5] In more detail, [KLST11]'s protocol relies on the existence of two mappings that are only proven to exist (using the probabilistic method and counting arguments), and no succinct or efficient instantiations are known. At a high level, both their mappings map from $[n^c]$ to $[n]^d$, where d is roughly the same as c and the success probability of the protocol is $1 - n^{-c}$. Since they prove existence by counting, the naive explicit representation is of size roughly n^c. Thus, if we want negligible error, i.e., $c \in \omega(1)$, the representation size is super-polynomial in n.

	Full Agreement	Static Filtering	Balanced	Rounds	Average Complexity	
					Communication	Computation
[KSSV06]	no	yes	yes	$\tilde{O}(1)$	$\tilde{O}(1)$	$\tilde{O}(1)$
[KSSV06] + [KS09]	yes	no	no	$\tilde{O}(1)$	$\tilde{O}(\sqrt{n})$	$\tilde{O}(\sqrt{n})$
[KSSV06] + [KLST11]	yes	no	yes	$\tilde{O}(1)$	$\tilde{O}(\sqrt{n})$	$n^{\omega(1)}$
Theorem 1	yes	yes	yes	$\tilde{O}(1)$	$\tilde{O}(\sqrt{n})$	$\tilde{O}(\sqrt{n})$

Table 1: A comparison of known scalable agreement protocols.

Lower bounds. Holtby, Kapron, and King [HKK08] showed a lower bound for all BA protocols with full agreement and (a strong form of) scalability. Roughly, they showed that in any correct BA protocol where in each round every honest party is allowed to send and process $\tilde{O}(1)$ bits, then the protocol must take $\tilde{\Omega}(n^{1/3})$ rounds and there is at least one honest party that sends $\tilde{\Omega}(n^{1/3})$ messages. Our Theorem 2 morally improves upon the above lower bound by showing that, no matter the size of messages, at least one party must process $\Omega(\sqrt{n})$ messages. Formally, the lower bounds are incomparable: (1) we prove a stronger lower bound ($\Omega(n^{1/2})$ vs. $\tilde{\Omega}(n^{1/3})$) and we do not restrict the size of messages, but (2) their lower bound is on the outgoing communication while ours is on the sum of computational work and outgoing communication.

Static vs. dynamic filtering. As mentioned, we target and achieve protocols with static filtering rules, i.e., every party decides on the set of parties it will listen to before the beginning of each round (as a function of its internal view at the end of the previous round). This modeling is very well motivated given the way large communication networks are designed.

However, all existing scalable full agreement protocols, balanced or unbalanced, explicit or non-explicit, fall outside of this model. Specifically, all protocols (including [KS09, KLST11] and many more [KS11, BGH13, BGT13, BCG21, ACD+19, CKS20, BKLL20]) assume some form of "dynamic" filtering, where the decision of which message is received can be based on the content of received messages. For example, whether a message is received or not could depend on the number of received messages with particular properties in the same round, or if the message contains particular authentication (e.g., a digital signature). Dynamic filtering gives non-trivial power: not only do existing lower bounds ([HKK08]'s and ours) not apply, but in fact it is possible to get full agreement (against computationally bounded adversaries) with essentially optimal parameters, assuming various cryptographic primitives and trusted setup assumptions [BCG21].

Lastly, we emphasize that before the current work it was completely unknown if it is possible to obtain a correct and scalable agreement protocols against computationally unbounded attackers, even with dynamic filtering.

1.3 Applications of Scalable Agreement

First, of course we do not want to settle on computationally inefficient protocols. Not only they are less useful on their own, but they are also less applicable, even in theory. For instance, we will not be able to use them as building blocks in bigger protocol whose security holds only if parties are efficient (say, if the protocol relies on cryptographic assumptions). We mention the concrete example of secure multi-party computation protocols, all of which essentially rely on the availability of a broadcast channel. Below, we also discuss *scalable* secure multi-party computation protocols. But, first, we argue that even scalability (i.e., being not only efficient but sub-linear) is sometimes absolutely necessary.

Agreement for massively parallel computation. Consider the so-called "Massively Parallel Computation" model [KSV10], one of the most widely-accepted models for designing protocols for large-scale datasets residing on a parallel computing cluster. In this model, a huge dataset is stored on a collection of machines, each of which has bounded storage and cannot store the whole dataset. The goal of the machines is to perform some joint computation on the entire dataset. Depending on the exact parameters of the system, it is possible that every machine cannot even store a single message from every other machine in a single round. Designing agreement protocols in this model is a fundamental task, and (to the best of our knowledge) Theorem 1 is the first solution for this task.

In fact, this problem came up in a recent work of Chan, Chung, Lin, and Shi [CCLS20]. In their work, they needed to have a quorum (i.e., a "good" committee per party). Using existing solutions for scalable agreement (e.g., [KS09, KLST11], ignoring the computational inefficiency of the latter) requires some party to either send too many messages, or store and process a message from every other party in the system, both of which are impossible in the model of Massively Parallel Computation. They bypassed this problem by introducing a strong trusted setup assumption (a common random string chosen *after* the adversary corrupts parties) and combining it with cryptographic assumptions.[6] Naturally, we would like to avoid these. Our Theorem 1 can be used as a drop-in replacement for the above method.

Secure multi-party computation. Multi-Party Computation (MPC) enables a set of mutually distrusting parties to compute a function on their private inputs, while guaranteeing that the inputs remain private and the only information that is leaked from the computation is the output of the function. Feasibility results for MPC have been long known [GMW87, BGW88, CCD87]. For example, the BGW [BGW88] protocol gives a method for computing an arbitrary function with complexity that grows multiplicatively with the circuit size of the function and some polynomial in the number of parties. That is, the complexity of each of the n parties is $s \cdot \mathsf{poly}(n)$ when computing a function represented as a circuit of size s. The round complexity is $O(d)$, where d is the depth of the circuit representation of the function.

There has been a rich line of work on scalable MPC protocols. The main goal is to design protocols where the total communication complexity scales like $\tilde{O}(s+\mathsf{poly}(n))$ for securely computing a size s circuit by n parties (for example, [HM01, DI06, DN07, IPS09, DIK$^+$08, BH08, DIK10, GIP$^+$14, IKP$^+$16, CCXY18, CGH$^+$18]. In all of these works, while the obtained protocols satisfy strong security guarantees, a broadcast channel is assumed. But, often its usage is limited to a number of times which is independent of the circuit size.

Dani et al. [DKMS12, DKM$^+$17] were the first to given an MPC protocol in the peer-to-peer model (no free broadcast at all), where communication scales with the circuit size plus a sub-linear term in the number of n. In these works, they used the agreement protocol of [KLST11] to get an $\tilde{O}(d)$-round MPC protocol with per-party communication complexity $\tilde{O}(s/n + \sqrt{n})$ to securely compute a size s depth d circuit by n parties. The corruption model is $(1/3 - \epsilon)$ fraction of static corruptions for any constant $\epsilon > 0$. Since they relied on [KLST11], they inherited its computational inefficiency and so the space and computational resources required for their resulting MPC are super-polynomial in n.

Using our Theorem 1 into the protocol framework of [DKM$^+$17], we obtain the first truly scalable MPC. Specifically, our n-party MPC requires each party to process, store, and communication $\tilde{O}(s + \sqrt{n})$ bits throughout $\tilde{O}(d)$ rounds to securely compute any function given by a size s depth d circuit. Security holds assuming $(1/3 - \epsilon)$ fraction of static corruptions for any constant $\epsilon > 0$. See

[6] In short, they assume that a completely random string is known to all parties (but not the adversary!) and they generate a quorum by expanding it via a pseudorandom function.

Theorem 10 for a precise statement.

Scalable agreement with optimal optimistic complexity. Given the lack of progress in going below $O(\sqrt{n})$ per-party complexity for distributed agreement (which is partially explained by our lower bound from above), a recent work [GK23] suggested to go "beyond worst case". Specifically, they provide a protocol with the same complexity as the best-known scalable agreement protocol in the worst case, but is much better in honest executions. Their $\tilde{O}(1)$-round agreement protocol is guaranteed to terminate and reach agreement after sending (and processing) $\tilde{O}(1)$ bits per party *if all parties behave honestly*. If not, the guarantees of their protocol fall back to those of an off-the-shelf protocol. By using our agreement protocol instead of [KLST11]'s as the off-the-shelf protocol with the best worst-case guarantees, we get a truly scalable agreement protocol (both in communication and in computation and space, as in Theorem 1) which also has essentially optimal optimistic complexity.

2 Overview of Our Techniques

The starting point of our work is the almost-everywhere agreement protocol of King et al. [KSSV06]. Their protocol results with a "global string" seed of poly-logarithmic length and with poly-logarithmic (min-)entropy.[7] However, the string is only known to $1 - o(1)$ fraction of the honest parties. I.e., each party $i \in [n]$ holds seed_i and only for $1 - o(1)$ fraction of parties, $\mathsf{seed}_i = \mathsf{seed}$. No party "knows" if $\mathsf{seed}_i \stackrel{?}{=} \mathsf{seed}$. We call parties for which $\mathsf{seed}_i = \mathsf{seed}$ as *knowledgeable*. Our goal is to make *all* honest parties agree on seed. Boosting almost-everywhere agreement to full agreement was also the approach of all previous works in this line. We provide the first such *scalable* transformation, as stated next.

Theorem 3 (Scalable almost-everywhere to everywhere; Informal)**.** *There is a protocol that translates an almost-everywhere agreement into everywhere agreement. The protocol terminates within $O(1)$ rounds, and each party stores, processes, and sends $\tilde{O}(\sqrt{n})$ bits overall. The transformation is secure against an adversary that statically corrupts up to $1/3 - \epsilon$ fraction of parties for any constant $\epsilon > 0$.*

This transformation, combined with the protocol of [KSSV06], immediately gives Theorem 1. Thus, we focus on Theorem 3. To build intuition, it is worth considering the following two (flawed) naive approaches for getting everywhere agreement from almost-everywhere agreement:

1. Each party chooses at random a small set of parties and polls them for their candidate string seed_i. Each party will output the string that is voted by the majority of incoming messages.

2. Each party chooses at random a small set of parties and tells them its string seed_i. Each party will output the string that is voted by the majority of incoming messages.

If we choose the sizes of the above sets to be poly-logarithmic in n, then since these sets are chosen uniformly at random, with all but negligible probability of error, the resulting protocol will satisfy efficiency. Also, by a simple concentration bound, in *an honest execution* full agreement is reached with all but negligible probability of error. However, both ideas are completely broken in the presence of an adversary.

[7] The protocol of [KSSV06] results with a "layered network" where internal nodes are small committees and the leaves are the parties. This structure allows a central committee (the root of the network) to perform various computation and disseminate information to all but $o(1)$ of the leaves. For example, the central committee can choose a sufficiently long string with sufficient min-entropy and distribute it.

In the first approach, the adversary can flood the honest parties, namely, the adversary can send a given honest party a query from each of the corrupted parties and thereby block it from answering other honest parties' queries. In the second approach, the adversary can force an honest party to learn a wrong value by targeting it and sending it a wrong string from all corrupted parties. While these solutions are not good enough, they do lead us to a viable approach: all we need is to make sure that the adversary does not have too much freedom in choosing who they message; at the same time, we need to allow sufficient freedom for honest parties as otherwise the adversary can corrupt a specific subset of parties. The challenge is of course balancing between these two requirements.

2.1 Restricting Polled Lists

As mentioned, we want to restrict the way polled parties are chosen, but yet allow sufficient freedom for the honest parties so they can learn the right value. Roughly, we want the following features:

> **Efficiency:** sampling a poll list can be done fast, in sub-linear time. In particular, the size of a poll list should be small-ish. Looking forward, a poll list in our construction will be of size $O(\sqrt{n})$ and this is also the complexity required to sample one.
>
> **Correctness:** ideally, the fraction of honest and knowledgeable parties in each such set is at least 1/2. Looking forward, we will not be able to satisfy such a strong requirement and we therefore settle for a more fine grained definition: we only want the fraction of "good" poll lists to be slightly higher than the fraction of "bad" ones, where a poll list is good if the fraction of honest and knowledgeable parties in it is more than 2/3, and it will be bad if the fraction of honest and knowledgeable parties in it is less than 1/3 (note that a poll list might be neither good nor bad).
>
> **Security:** roughly, the number of times parties in an honestly generated poll list that can be polled by maliciously generated poll lists is small. (Otherwise, many honest party can be flooded and blocked from replying to honest parties' poll requests.)

To achieve the above properties we utilize a finite affine plane. Each party is associated with a point on the plane. Fix a field $\mathbb{F} = \mathbb{F}_{p^k}$ for a prime p and an integer k. Since each party is a point in the plane, we map each number in $\{1, \ldots, n\}$ to $\mathbb{F} \times \mathbb{F}$, so $n = p^{2k}$.[8] Note that, in the technical section we explain how to handle all n's, even ones that cannot be written as even powers of a prime, by a certain padding argument; see Remark 2 for details. Roughly, poll lists are defined as the points that reside on a line: Party i's possible poll lists are defined as the possible sets of points that reside on lines that pass through point i.

Let us explain how we use this structure to satisfy the needed properties. Let us start with arguing efficiency. First, observe that by the way we set the parameters, every point has p^k lines that pass through it and so a possible poll list consists of at most $p^k = \sqrt{n}$ other parties. For efficiency of computation, all one needs is a field and a canonical mapping of IDs to points on the plane. It is easy to perform operations on this structure efficiently (in time $\tilde{O}(\sqrt{n})$), like checking which points reside on a given line or which lines pass through a given point.

We proceed by arguing correctness and security. Let us focus on a concrete honest party, say (without loss of generality) the first one, P_1, and see how to make sure it learns the correct value. Party P_1 picks a line that passes through its associated point uniformly at random and the points on this line tell it which parties it will poll. A poll request is sent out to each of these parties. P_1 gets

[8] We remark that our structure is somewhat reminiscent of projective planes, although it is slightly different and simpler.

back (up to) $\sqrt{n}$ votes and a decision is registered if more than 2/3 of the votes are for the same value; otherwise, all votes are discarded. Looking ahead, we actually repeat this process sufficiently many times, and as we will argue below, this suffices to learn the correct value.

Consider a single execution of the above process. We first argue that an adversary has no point in trying to flood parties in P_1's poll list.

Claim 1 (Security)**.** *Any adversary can flood (only)* $o(1)$ *fraction of the parties in* P_i*'s poll list.*

In other words, an adversary cannot affect the outcome of the result of the vote in any useful manner by flooding (i.e., it can affect the count by only additive $o(\sqrt{n})$ which will not matter). To see why the claim is true, observe that in our construction every two lines intersect only at one point (or zero for parallel lines). Therefore, whatever line a malicious party chooses to poll, it will only cause a poll request to a single party in P_1's poll list. So, roughly speaking, since every party can handle $\omega(\sqrt{n}) \cap \tilde{O}(\sqrt{n})$ requests, in order to flood $\Theta(\sqrt{n})$ parties from P_1's poll list, each of them needs to get polled by at least $\omega(\sqrt{n})$ malicious parties and so there must be at least $\Theta(\sqrt{n}) \cdot \omega(\sqrt{n}) \in \omega(n)$ malicious parties, which is a contradiction. Note that we ignored parties that were included in P_1's poll list since there are at most $O(\sqrt{n})$ of them (so there is not enough of them to flood).

So, from now on, we assume that each polled party replies. (Again, in a real execution a given polled party might not reply because it was flooded with malicious poll requests. But, as argued by the claim above, only rather few of the parties can be flooded, and this will not affect the arguments below.) The outcome of the poll of P_1 can have one of three outcomes:

1. Event E_1: More than 2/3 of the votes were consistent but for a wrong value.
2. Event E_2: More than 2/3 of the votes were consistent and for the correct value.
3. Event E_3: There is no value that was voted for more than 2/3 of the votes.

We show the following claim.

Claim 2 (Correctness)**.** *There are two positive constants* α, β *such that*

$$\Pr[E_1 \vee E_2] \geq \alpha \text{ and } \Pr[E_2 \mid E_1 \vee E_2] \geq 1/2 + \beta.$$

So, *some* value is learnt by P_1 with constant probability; and this value is more likely (by a constant factor β) to be the right value. Thus, by repeating the voting process $\mathsf{polylog}(n)$ times independently (as we mentioned above), and choosing the vote that appears the majority of times, we will indeed learn the right output except with negligible probability of error.

We now explain why the claim is true. First, recall that the adversary statically corrupts $1/3 - \epsilon$ fraction of parties and moreover only $o(1)$ fraction of parties are honest and *not* knowledgeable. For simplicity, let us assume that the identity of the latter $o(1)$ fraction as well as their behaviour are completely controlled by the adversary. In other words, there are $1/3 - \epsilon + o(1) < 1/3 - \epsilon'$ fraction of corrupted parties for some constant $\epsilon' < \epsilon$. So, a given party is either malicious or honest and knowledgeable.

Let us first argue that $\Pr[E_1 \vee E_2] \geq \alpha$ for some positive constant α. Because E_1, E_2, and E_3 are disjoint and they cover the entire space, it suffices to show that $\Pr[E_3] \leq 1 - \alpha$. To cause E_3, the adversary needs to maximize the number of poll lists where it controls at least 1/3 fraction of parties. We call such poll lists *partially corrupted*. How many of P_1's poll lists can the adversary partially corrupt? There are $(1/3 - \epsilon') \cdot n$ corrupted parties and by our affine plane construction (i.e., the possible poll lists of each party are completely disjoint) it can then partially corrupt roughly $(1/3 - \epsilon') \cdot n/(1/3 \cdot \sqrt{n}) = (1 - 3\epsilon') \cdot \sqrt{n}$ poll lists. This is only a constant (i.e., $1 - 3\epsilon'$) fraction of

P_1's poll lists. Now, since P_1 chooses its poll list uniformly at random, it will evade these partially corrupted poll lists with probability $1 - \alpha$, where $\alpha = 3\epsilon'$.

Next, following similar logic, we argue that $\Pr[E_2 \mid E_1 \vee E_2] \geq 1/2 + \beta$ for some constant β. To cause E_1, the adversary needs to maximize the number of poll lists where it controls at least $2/3$ fraction of parties. We call such poll lists *fully corrupted*. How many of P_1's poll lists can the adversary fully corrupt? Let $\gamma \cdot \sqrt{n}$ be the number of partially corrupted poll lists. By a similar argument to the above, an adversary can fully corrupt roughly $(1/3-\epsilon'-\gamma/3)\cdot n/(2/3\cdot\sqrt{n}) = (1/2-3\epsilon'/2-\gamma/2)\cdot\sqrt{n}$ poll lists. Since P_1's poll list is chosen uniformly at random, $\Pr[E_1 \mid E_1 \vee E_2] = (1/2-3\epsilon'/2-\gamma/2)/(1-\gamma) \leq 1/2 - 3\epsilon'/2$, which means that $\Pr[E_2 \mid E_1 \vee E_2] \geq 1/2 + \beta$ for $\beta = 3\epsilon'/2$, as needed.

2.2 Enforcing Poll Lists with the Prescribed Structure

There is one important detail that we have glanced over so far. While the affine plane structure puts a significant constraint on the adversarial choices of polled parties, it is only useful if malicious parties choose their polled parties via the above-given method. Up until now, we have not described any mechanism that prevents the adversary from completely ignoring the prescribed structure of poll lists. Note that we also have not yet used the fact that seed has entropy. We will use this fact now to enfore "legal" poll list choices even for malicious parties.

We enforce choices of poll lists with the prescribed structure by associating a small designated "committee" for each party. The committee will basically serve as a proxy for each party and will perform all communication in the name of its associated party. We emphasize that we want a different committee for each party and not one global committee, because the latter will result with unbalanced protocols. Once we associate with each party a designated committee, we require each party to "commit" on a poll list with a prescribed structure by sending it to the committee members whose role is to verify that this is indeed a legally-structured poll list. All poll requests will then be sent from "committees" and not from parties.

Let us first specify the properties of these n committees. We need n committees, each of which is rather small—for us, we will have $\tilde{O}(1)$ parties in a committee. Each committee will consist of at least $1/2$ fraction of honest parties, and we need the committees to be "balanced" in the sense that every party will appear in roughly the same number of committees. The former is needed for the committee to be considered "honest", and the latter is necessary to obtain a balanced protocol. We proceed to explain the construction.

Our starting point is the well-known connection between generating a single committee and a pseudorandom object called an *averaging sampler* [BR94, Zuc97]. An averaging sampler is a mapping that gets an input with high enough (min-)entropy and returns a poly-logarithmic size multiset such that for every "statistics function" from the set space to [0, 1], the average of that function on the elements in the multiset is "not too far" from the mean of the function (see Definition 3). Thinking of the function that returns 1 if the party is honest and 0 if the party is corrupted, w.h.p the output of the sampler is a set with a majority of honest parties. Thus, using seed as the input for the averaging sample, we get a committee with a majority of honest parties (with high probability). There are several known constructions of averaging sampler, e.g., [BR94, Zuc97, Gil98, GUV09] (thanks to their equivalence to seeded extractors). For concreteness, we use the averaging sampler that follows from the work of Guruswami, Umans, and Vadhan [GUV09]. But, we need n committees and not just one; also, the committees should be "balanced", as mentioned above.

It might be tempting to "open up" existing averaging sampler constructions and see if these properties could be achieved by some adaptation. Somewhat surprisingly, we observe that a very simple and generic approach works, assuming the averaging sampler is "good enough" to begin with. Assume that the chosen committee (via the sampler, as above) consists of parties $C = \{p_1, \ldots, p_\ell\}$,

where we imagine every p_i as an integer between 1 and n (formally, a committee is a multi-set as duplicates are allowed). We generate n committees $C_1, \ldots, C_n$ as follows:

$$C_i = \{p_1 + i, p_2 + i, \ldots, p_\ell + i\} \qquad (\text{mod } n),$$

That is, the committees are all the cyclic permutations of the committee. Why are all the resulting committees have a majority of honest parties? This is where we use the fact that the committee was chosen by an averaging sampler! Specifically, it must be that the permutation on the output space does not change the probability of having a "good" output. This is because the mean of the "fraction of corrupt parties" function does not change even if we permute the names of parties. The formal proof uses a union bound to argue that the sampler property is held for each of the n committees; this is where we need to start off with a good enough sampler. Lastly, the fact that the resulting committees are "balanced" is satisfied directly by construction: A party p will appear in the ith committee only if $p - i \bmod n$ appears in the original committee C. Since the size of a committee is poly-logarithmic, we get that every party appears in at most poly-logarithmically many committees.

The above construction has another property crucial for the computational efficiency of our protocol: it is not only easy to compute committee members of a given party, but it is also easy to compute the set of committees of which a given party is a member.

2.3 Putting Things Together for Everywhere Agreements

Each party randomly chooses a description of a poll list and sends it to its corresponding committee which is defined using the global string seed. The committee forwards the requests to each party in the poll list and the parties that receive the poll request send a reply directly back to the party. Since parties know each other's committee, an honest party that is also knowledgeable will accept the request. Also since w.h.p, there is a majority of honest and knowledgeable parties in each committee, they will forward messages only to the correct poll list. Before we conclude the overview, we explain two additional (simpler) remaining caveats.

Caveat 1: Some parties are not knowledgeable. Since some of the honest parties do not know seed, they do not know the "right" set of committees used to enforce usage of valid poll lists. To overcome this we use an idea of [KLST11]. Each party P_i randomly sends seed_i to $\approx \sqrt{n}$ parties, and in turn each party chooses randomly $\approx \sqrt{n}$ parties to accept messages from them. By the birthday paradox, we know that each honest party will receive (and keep) seed at least once. This is true even if all corrupted parties send wrong seeds to all the honest parties. This guarantees that each party knows a small list (of size $\approx \sqrt{n}$) of seed candidates, one of which is the correct one. Thus, the protocol is basically repeated for each candidate in the list. Therefore, with high probability it will send it also to the committee that is generated from seed, and the protocol will proceed as above. See Section 5 for details.

Caveat 2: How poll requests are distributed. Above, we said that a committee forwards the poll requests of the associated party to each member in the poll list. But, an honest party, say P_1, might end up getting too many poll requests. This could happen either because all malicious parties choose a poll list where P_1 is a member, or because malicious parties, acting as committee members, can distribute (bogus) poll requests.[9] Thus, we restrict the way poll requests are distributed in a related way to how they were generated. Specifically, committees distribute poll requests via a $\sqrt{n} \times \sqrt{n}$ communication grid, where each committee is associated with a (unique) point on the

[9] Recall that due to static filtering, polled parties *cannot* process so many incoming messages.

grid. The grid is implemented via the same affine plane that we used to choose poll lists, restricted to two gradients. This choice is important because we make sure that the poll list associated with a given point in the grid contains exactly one point in every other column.

To see how distribution is done, we give an example. Assume that (x, y) wants to send a poll request to (a, b). It uses the committee corresponding to point (x, b) as a proxy. Because only (fixed) $\sqrt{n}$ committees (i.e., all $(x, *)$) can send poll requests via (x, b), it can process all of them. Now, before conveying all poll requests from $(*, b)$, i.e., the (fixed) proxy committees of committee (a, b), we can directly check (say by summing up the total number at (a, b)) if there are too many poll requests intended for (a, b) and in such a case discard all of them. From a geometrical perspective, communication between committees is allowed to occur via vertical or horizontal lines only; see Figure 1 for an example of an elaborate scenario.

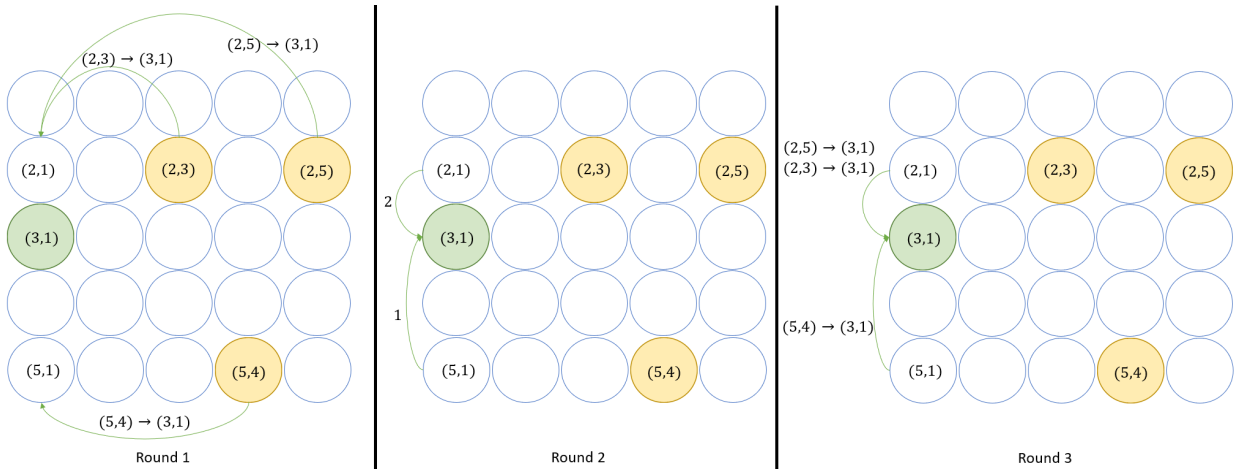

Figure 1: An example of how poll requests are distributed. In this example, $(3, 1)$ is in the poll list of $(2, 5), (2, 3)$, and $(5, 4)$.

Finally, the way a single iteration of the main part of the protocol is depicted in Figure 2.

Remark 1 (Comparison with [KLST11])**.** *King et al.'s [KLST11] protocol is most related to ours. While most of the details of our protocol differ from King et al.'s, we mention the points of similarity. At a very high level, their protocol follows a similar blueprint of restricting valid poll lists and using committees to enforce "legal" choices. However, the implementations and all lower-level details are entirely different. In particular, King et al.'s protocol relies on an* existential *way to restrict the valid poll lists and committees, and they do not provide a concrete instantiation. Also, they require very strong properties from their object and as such we do not know how to obtain a explicit version of their protocol. Additionally, their protocol has an iterative* $(\log n)$*-round process to geometrically decrease the fraction of honest but unknowledgeable parties. We do not have such a process and achieve full agreement in "one shot" using only constant (i.e., 7) rounds.*

The fact that we were able to give explicit constructions comes from two novel new observations that we make. First, since we are aiming for $\tilde{O}(\sqrt{n})$ *communication, we can work with poll lists that consist of* $\sqrt{n}$ *parties. In comparison, King et al.'s [KLST11] transformation uses (the existence of) poll lists of poly-logarithmic size. Second, we utilize the fact that the adversary controls only* $1/3 - \epsilon$ *fraction of parties. King et al.'s transformation works even if* $1/2 - \epsilon$ *fraction of parties are corrupted. The technical details of our construction (including our usage of an affine plane and an explicit averaging samplers) are completely new to this work.*

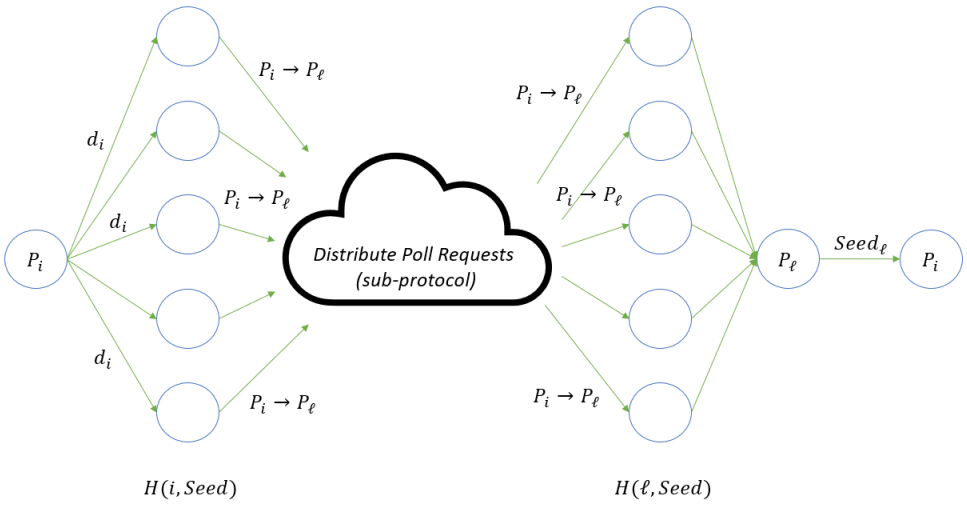

Figure 2: An example of the way poll requests are distributed. H is the function that maps a party to its committee members. J is the function that maps a party and a direction to the set of polled parties. In the above figure, we assume that ℓ is in the poll list of i in direction d_i.

2.4 From Agreement to Byzantine Agreement, Broadcast, and More

So far we showed how to reach consensus on a poly-logarithmic length string with poly-logarithmic min-entropy. Next, we observe that this abstraction suffices for many classical distributed tasks, including Byzantine Agreement, Broadcast, and Quorum/Committee/Leader Election. There are several ways to obtain each of these applications given our protocol, but here we explain a generic method.

Consider the problem of Byzantine Agreement. To achieve it, we first let the n parties run our agreement protocol (as above). Using the resulting string seed, we use our method of generating n "good" committees, a.k.a a quorum. We think of the committees as additional input-less and randomness-less honest parties. We let each party send its input bit to its corresponding committee and then the committees count which input bit was the most common one. This can be done in a tree like fashion. This bit is then distributed to all the committees (again in a tree-like fashion), and then the committees send the result to their associated parties.

Broadcast can be achieved in a similar way by only letting the broadcasting party send its input to its associated committee which will then send it to all other committees (in a tree like manner). Finally, committees will send the message to their associated parties. Obtaining a quorum is basically what we achieved above. A (single) committee could be any particular committee in the quorum. A leader can be obtained by running any non-scalable leader election protocol within a committee and then broadcasting the result to everyone. Secure multi-party computation is obtained by plugging in our quorum generation protocol into the protocol of Dani et al. [DKM$^+$17]. Full details appear in Section 6.

2.5 The Lower Bound

We identify a communication pattern that must exist in any protocol that satisfies the property that each party sends and processes at most $o(\sqrt{n})$ messages. Given a protocol and the inputs of

all the parties, the probability that party P_i ever sends a message to party P_j is a (deterministic) function of n. Therefore, for each party P_i, we can split the other parties into two sets. The first set consists of all the parties with whom P_i tries to communicate with high probability ($\Omega(1/\sqrt{n})$, over an honest execution). The second set is the complementary one: i.e., all parties with whom P_i tries to communicate rarely (with small probably, $o(1/\sqrt{n})$).[10] Note that the definition is not symmetric: P_j could want to communicate with P_i often, but P_i may not want to do so.

Given these sets, by counting the expected number of messages overall in the course of the protocol, we proved the following:

1. For each party, the first set (the high probability set) is not too large ($o(n)$).
2. The number of parties that communicate through their second set (the low probability set) is not too large ($o(n)$).

We now define the following adversary. Following the above statements, it finds a set S of parties that with high probability communicate only within itself and with another small-ish set, denoted T (we show how to obtain such an S and T). The adversary might not find sets that fit to the requirements if each party can sends or receives $\Omega(\sqrt{n})$ bits. Then, the adversary can cause the parties in S to believe that the input configuration was 0 to everyone, and to parties outside S to believe that the input configuration was 1 to everyone. This readily implies a contradiction.

The above attack requires only static corruptions of any constant fraction of parties, does not need the "full information model", and the adversary is not even rushing. I.e., it works even in the much weaker model of authenticated private channels. Lastly, we remark that our attack is computationally inefficient (or non-uniform) because the adversary needs to know the probabilities that certain events happen throughout an honest execution of the protocol. But using standard techniques (repetition and tail bounds), it seems that the attack could be made efficient (we leave formalizing this to future work). See Section 7 for details.

3 The Model

Communication model. We consider n parties in a fully connected network (i.e., clique). Each party has a unique ID and the IDs are common knowledge. We assume the IDs are $1, \ldots, n$. Parties can perform arbitrary computation and they have a source of private randomness. Communication is authenticated, that is, whenever a party sends a message directly to another, the identity of the sender is known to the recipient. We assume synchronous communication. That is, communication proceeds in rounds; messages are all sent out at the same time at the start of the round, and then received at the same time at the end of the same round. All parties have synchronized clocks.

Definition 1 (Terminology)**.** *We use the following terms:*

- *The number of rounds of communication required for all parties to terminate is called the* round complexity *of the protocol.*
- *A protocol is said to be* balanced *if all parties send the same amount of bits in every round of communication.*
- *A protocol is said to have* communication complexity cc *if the total amount of bits transmitted throughout an execution of the protocol (by all parties together) is at most* cc.

[10] In the formal proof, we divide each set again to two sets: one for the parties that P_i listens to and the other for the parties that P_i sends messages to. The static filtering model implies that these sets are well-defined given the protocol's description.

All of the above are measured in a worst-case sense (over all possible randomness and inputs).

We also care about the computational and storage complexity of each party in the protocol. Note that an adversary can always blow up the storage and computational complexity of a protocol by flooding honest parties with many bogus messages. It is therefore standard to count only messages that are actually processed by honest parties. We follow the modelling of Boyle et al. [BCDH18], who formalized this intuition. Namely, message receival consists of two phases:

1. **(static) filtering phase**: incoming messages are inspected according to specific filtering rules defined by the protocol specification, and some messages may be discarded.

   We require the filtering phase to be completely static: at the beginning of a round the party fixes the identities of a set of parties it is willing to receive messages from and also specifies a bound on the maximal allowed incoming message length (per party). A message passes the filtering phase only if both the sender of the message is in the set and the message is in the right length; otherwise, it is discarded.

2. **storage and processing phase**: each party computes its next-message function based on the remaining non-discarded messages.

The space and computational complexity of a party are determined by the complexities of the next-messsage function, that is, they only depend on the messages that were not discarded during the filtering phase.

We mention that some papers allow the filtering procedure to perform somewhat heavy operations like keeping track of how many messages were sent by each party (e.g., [KLST11]) or even verifying a digital signature for every incoming message (e.g., [BCG21]). This model was called *dynamic* filtering by Boyle et al. [BCG21]. Clearly, it is less desirable to require expensive computational work for every incoming message, and so ultimately protocols with static filtering rules are preferable.

Adversarial model (point-to-point full information). We assume that there is an adversary that controls up to t parties and whose goal is to cause the protocol to fail in some way, depending on the context. The adversary chooses which parties to control non-adaptively, i.e., it chooses the set of corrupted parties at the start of the protocol. The adversary is malicious: corrupted parties can engage in any kind of deviations from the protocol and send arbitrary messages in the name of the corrupted parties. We emphasize that corrupted parties can send arbitrarily long messages to arbitrary other parties. We assume that the adversary is *rushing*, that is, it can view all messages sent by the honest parties in a round before the corrupted parties send their messages in the same round. Moreover, the adversary sees all messages, even messages sent between two honest parties (but honest parties only see messages that are sent directly to them). This model is known as the *point-to-point full information model* [KSSV06].

Byzantine agreement. In this problem, there are n parties, $P_1, \ldots, P_n$. At most t of the parties may be corrupted. Each party P_i begins with an input bit $x_i \in \{0, 1\}$ and outputs a bit y_i. The goal of the protocol is (with high probability) for all honest parties to terminate and, upon termination, agree on a bit held by at least one honest party at the start. This should hold even when the t corrupted parties collude and actively try to prevent it. More precisely, the Byzantine agreement problem is defined as a protocol that satisfies with high probability the following *agreement*, *validity*, and *termination* properties:

- **Agreement:** For every pair of honest parties P_i and P_j it holds that $y_i = y_j$.

- **Validity:** If there exists a bit x such that for every honest party P_i it holds that $x_i = x$, then the common output is x.
- **Termination:** Every honest party eventually outputs a bit.

Other distributed tasks. There are few other intimately related basic distributed functionalities. The first is *committee election* and the second is *broadcast*. In **committee election**, the goal is to bring all honest parties to agree on a small subset of parties with a fraction of corrupted parties close to the fraction for the whole set. In the extreme case, the committee is of size 1 in which case a single leader is chosen (i.e., **leader election**) and the goal is to guarantee that the leader is an honest party with constant probability. In **broadcast**, the goal is to allow an honest party to communicate a message to all other parties so that the message that all other honest parties end up knowing is the same as the one sent.

These tasks are very much related but they are not equivalent in our setting. Indeed, often a "light-weight" committee election protocol is used as a proxy to get other primitives by first electing a small committee and then running a "heavy-weight" protocol among the committee members for task X, where X could be (for instance) leader election, BA, or broadcast. Indeed, if the committee is small, then we can afford to execute somewhat inefficient protocols among the committee members. While such a blueprint would result with (total) communication efficient protocols, they will be highly unbalanced. Indeed, for the committee to broadcast the result to all other parties, the committee members would need to communication with all other parties, causing the protocol to be highly unbalanced.

Lastly, we mention that all of the above abstractions are special cases of a much more general concept called **secure multi-party computation**, where an arbitrary computation is to be performed and where additionally privacy of inputs is required.

4 Restricting Poll Lists and Enforcing Consistency

This section gives two explicit mappings that our protocols use. The first mapping, denoted H, is used to create a quorum out of a single random-enough string. The quorum is used to enforce consistency and well-formedness of (adversarially chosen) poll lists. The second mapping, denoted J, is used to sample private poll lists of a prescribed structure. We start by explaining the construction of H since it is a bit easier and then explain the construction of J.

4.1 Generating a Quorum (the H Mapping)

This section shows a method for generating a good quorum given a good committee. The method is completely non-interactive and requires every party to perform a certain (local) polynomial-time computation in its input size.

We start with the definition of a good committee and a good quorum. Roughly, a good committee is one that has a significant fraction of honest parties. A good quorum is a collection of such committees, one for each party, which is also "balanced" in the sense that no party participates in too many committees.

Definition 2 ((ϵ, d, B)-good quorum)**.** *A quorum is a collection of n committees $C_1, \ldots, C_n \in [n]^d$ The quorum is (ϵ, d, B)-good for $\epsilon > 0$, $d = d(n)$, and $B \subseteq [n]$, if*

1. *For every $i \in [n]$, it holds that $(1/d) \cdot \sum_{z \in C_i} \mathbb{1}_{z \in B} < \epsilon$.*

2. *For every $j \in [n]$, party j appears in the collection $O(d)$ times overall. That is, $\sum_{i=1}^{n} \sum_{z \in C_i} \mathbb{1}_{z=j} = O(d)$.*

Our transformation is done via the following theorem which can be viewed as a pseudorandom object that takes a sufficiently random short string, and outputs many random-enough committees. Indeed, the proof of this theorem relies on the existence of a good enough randomness extractor, viewed as a sampler.

Theorem 4. *For every integer $n \in \mathbb{N}$, $c = c(n)$, there exist $d = O(c \cdot \log n)$ and two mappings $H, H^{-1} \colon [n] \times [n^c] \to [n]^d$ such that*

1. *The functions H, H^{-1} are efficiently computable (i.e., can be evaluated in polynomial time in their input size).*
2. *If $j \in H(i, x)$, then $i \in H^{-1}(j, x)$.*
3. *Viewing each $H(i, x)$ as outputting a multiset of $[n]$ of size d, the following holds. For every $B \subseteq [n]$, for at least $1 - 1/n^{c-3}$ fraction of the x's, the collection $\{H(i, x)\}_{i \in [n]}$ is a $\left(\frac{|B|}{n} + o(1), d, B\right)$-good quorum.*

As a direct corollary of the above theorem, we obtain that if x is chosen from a distribution with sufficiently high min-entropy, then the resulting quorum is good. Below, a k-source is a distribution D over a finite domain Ω such that for all $\omega \in \Omega, D(\omega) \leq 2^{-k}$.

Corollary 1. *Let $n \in \mathbb{N}, c = c(n), d = O(c \cdot \log n)$, and $H, H^{-1} \colon [n] \times [n^c] \to [n]^d$ be as in Theorem 4. Let $B \subseteq [n]$ be arbitrary such that $|B| < (1/3 - \epsilon) \cdot n$, where ϵ is arbitrary small positive constant. Then, for any positive $c' = c'(n)$ and any constant $\epsilon' < \epsilon$, if x is sampled from a $(c' \cdot \log n)$-source over $[n^c]$, then with probability at least $1 - 1/n^{c'-4}$, the collection $\{H(i, x)\}_{i \in [n]}$ is a $(1/3 - \epsilon', d, B)$-good quorum.*

Proof. Assume (for contradiction) that x is chosen according to $(c' \cdot \log n)$-source over $[n^c]$, but the probability that $\{H(i, x)\}_{i \in [n]}$ is a $(1/3 - \epsilon', d, B)$-good quorum is less than $1 - 1/n^{c'-4}$. By Theorem 4 and since $|B| < (1/3 - \epsilon) \cdot n$, there are at most n^3 bad choices of x's and at least one of them should be chosen with probability at least $n^3/n^{c'-4} = 1/n^{c'-1}$. This is a contradiction to the assumption that x is chosen from a $(c' \cdot \log n)$-source. □

The rest of this section is devoted to the proof of Theorem 4.

Proof of Theorem 4. As mentioned the proof of Theorem 4 relies on the existence of a sufficiently good sampler. Therefore, we start with recalling what samplers are and how they can be instantiated. We then use any sufficiently good sampler to prove our theorem.

Definition 3 (Averaging sampler)**.** *A function $G \colon \{0,1\}^r \to [n]^d$ is a (δ, ϵ)-averaging sampler if on input a uniformly random r-bit string, it outputs a sequence of d sample points $x_1, \ldots, x_d \in [n]$ such that for any function $f \colon [n] \to [0, 1]$, we have $\left| \frac{1}{d} \sum_{i=1}^{d} f(x_i) - \mathbb{E}[f] \right| \leq \epsilon$ with probability $\geq 1 - \delta$. Here, $\mathbb{E}[f] = \frac{1}{n} \cdot \sum_{x \in [n]} f(x)$.*

Using a result of Guruswami, Umans, and Vadhan [GUV09], there exists an explicit (i.e., efficiently computable) sampler with the following parameters.

Theorem 5 (Follows from [Zuc97, GUV09])**.** *For all constant $\alpha \in (0, 1)$, for all positive integers r, n and for all $\epsilon = \epsilon(n) > 0$, there is an explicit $(2^{k-r+1}, \epsilon)$-averaging sampler $G \colon \{0,1\}^r \to [n]^d$ with $d = O(r + \log(\epsilon^{-1}))$ and $k = \log(n)/(1 - \alpha)$.*

Proof. The statement in [GUV09] is given in the language of extractors, but the latter are closely related to samplers. Let us recall what seeded extractors are and their relation to samplers. A function $E\colon \{0,1\}^r \times \{0,1\}^t \to \{0,1\}^n$ is a (k,ϵ)-extractor if for any k-source D over $\{0,1\}^r$, it holds that the total variation distance[11] between $E(D,U_t)$ and U_n is at most ϵ, where U_ℓ is the uniform distribution over ℓ bits. Zuckerman [Zuc97] showed that any such (k,ϵ)-extractor can be turned into a $(2^{k-r+1},\epsilon)$-averaging sampler $G\colon \{0,1\}^r \to [n]^{2^t}$. If the extractor is efficiently computable, namely, on every $x \in \{0,1\}^r$ and $s \in \{0,1\}^t$, $E(x,s)$ can be computed in $\mathsf{poly}(r,t)$ time, then the sampler, on every $x \in \{0,1\}^r$, can be computed in $\mathsf{poly}(r,2^t)$ time.

Now, we plug in the extractor of [GUV09]. They show that for all $\epsilon > 0, \alpha \in (0,1)$ and for all positive integers r,k, there is an explicit construction of an (k,ϵ)-extractor with $t = O(\log r)$ and $n \geq (1-\alpha)k$. Thus, for every $\epsilon > 0, \alpha \in (0,1)$ and positive integers r,n, there is a $(2^{k-r+1},\epsilon)$-averaging sampler $G\colon \{0,1\}^r \to [n]^d$ for $d = O(r + \log(\epsilon^{-1}))$ and $k = n/(1-\alpha)$, as required. Since the extractor is efficiently computable and since $t = O(\log r)$, the sampler G can be computed in time $\mathsf{poly}(r)$. □

To complete the proof we show that there exists a choice of the sampler parameters for which we can instantiate a function H as required in the statement. We use the averaging sampler from Theorem 5 with $\alpha = 1/10$ and $\epsilon = \log^{-1} n$. We get a function $G\colon [n^c] \to [n]^d$ which is a $(2^{k-c\cdot\log n+1}, \log^{-1} n)$-averaging sampler, where $d = O(c \cdot \log n)$ and $k = (10/9) \cdot \log n$. Thus, G is a $(n^{1.5-c}, \log^{-1} n)$-averaging sampler.

We define $H, H^{-1}\colon [n] \times [n^c] \to [n]^d$ as follows:

$$H(i,x) = \{z + i \bmod n \mid z \in G(x)\}$$
$$H^{-1}(j,x) = \{j - z \bmod n \mid z \in G(x)\}.$$

Since G is efficiently computable, so are H and H^{-1}. The second property in the statement follows directly from the construction. We proceed with the proof that the collection $H(1,x),\ldots,H(n,x)$ is a good quorum with high probability over the choice of x.

First, we show that for every $B \subseteq [n]$, with high probability over the choice of x, for every $i \in [n]$, it holds that $(1/d) \cdot \sum_{z \in H(i,x)} \mathbb{1}_{z \in B} \leq |B|/n + \log^{-1} n$. Fix $i \in [n]$. Define $f_i\colon [n] \to \{0,1\}$ as

$$f_i(j) = \begin{cases} 1 & j + i \bmod n \in B; \\ 0 & \text{otherwise.} \end{cases}$$

Since G is an averaging sampler, we have that

$$\Pr_{x \leftarrow [n^c]}\left[\left|\frac{1}{d}\sum_{j \in G(x)} f_i(j) - \frac{|B|}{n}\right| \leq \log^{-1} n\right] \geq 1 - \frac{1}{n^{c-1.5}}.$$

On the other hand, by definition of f_i and H,

$$\Pr_{x \leftarrow [n^c]}\left[\left|\frac{1}{d}\sum_{j \in G(x)} f_i(j) - \frac{|B|}{n}\right| \leq \log^{-1} n\right] \leq \Pr_{x \leftarrow [n^c]}\left[\frac{1}{d} \cdot \sum_{z \in H(i,x)} \mathbb{1}_{z \in B} \leq \frac{|B|}{n} + \log^{-1} n\right].$$

[11] The total variation distance (a.k.a statistical distance) between two distributions D_1 and D_2 over the same finite space Ω is $\frac{1}{2}\sum_{\omega \in \Omega} |D_1(\omega) - D_2(\omega)|$.

Therefore, for every $i \in [n]$,

$$\Pr_{x \leftarrow [n^c]} \left[\frac{1}{d} \cdot \sum_{z \in H(i,x)} \mathbb{1}_{z \in B} \leq \frac{|B|}{n} + \log^{-1} n \right] \geq 1 - \frac{1}{n^{c-1.5}} \,.$$

By a union bound over all i's, we get that the above holds for every $i \in [n]$ simultaneously with probability at least $1 - 1/n^{c-2.5}$, as needed.

Next, we show that for every $j \in [n]$, party j participates in at most d committees. Indeed, since we defined our quorum using shifts, we have that

$$\begin{aligned} \sum_{i=1}^{n} \sum_{y \in H(i,x)} \mathbb{1}_{j=y} &= \sum_{i=1}^{n} \sum_{z \in G(x)} \mathbb{1}_{j=z+i \bmod n} \\ &= \sum_{z \in G(x)} \sum_{i=1}^{n} \mathbb{1}_{j=z+i \bmod n} \\ &= \sum_{z \in G(x)} 1 = |G(x)| = d. \end{aligned}$$

□

4.2 Choosing Poll Lists (the J Mapping)

The parties in our protocol reach consensus on the global seed by polling from a randomly chosen set of other parties for their vote. Intuitively, since most parties agree on the global seed to begin with, a majority of parties will reply with the correct seed, thereby reaching consensus. However, implementing this naively fails since bad parties can flood all parties with requests, preventing them from replying to honest parties' polls. Thus, we need to somehow limit the set of "legal" polls. To this end, we make use of an object that, on the one hand, provides sufficient entropy in the choice of poll lists for honest parties, and on the other hand, adversarially-chosen possibilities should be limited.

Our object is formalized as a function $J : [n] \times [\sqrt{n}] \to \binom{[n]}{\sqrt{n}}$. We think of the first input as an index of a party and the second input as its randomness space. The output of $J(i, d)$, for $i \in [n]$ and $d \in [\sqrt{n}]$, is a subset of $[n]$ indicating a set of parties to poll from. The properties of J are listed in the following lemma.

Lemma 1 (Poll list)**.** *For every $n \in \mathbb{N}$ even power of a prime[12] (i.e., $n = p^{2k}$ for a prime p and an integer k), there exists an efficiently computable mapping $J\colon [n] \times [\sqrt{n}] \to \binom{[n]}{\sqrt{n}}$ such that:*

1. *For every $i \in [n]$ and $d \in [\sqrt{n}]$, $|J(i,d)| = \sqrt{n}$ and $i \in J(i,d)$. (That is, every party is in its own poll list and the size of a poll list is $\sqrt{n}$).*
2. *For every $i, j \in [n]$ and every $d \in [\sqrt{n}]$, if $j \in J(i,d)$ then $i \in J(j,d)$. (For the same randomness d, party i is in the poll list of j if party j is in the poll list of i).*
3. *For every $i, j \in [n]$ and all $d_1 \neq d_2 \in [\sqrt{n}]$, $|J(i,d_1) \cap J(j,d_2)| = 1$. (For every two parties, as long as they use different randomness, there is a single party that participates in both poll lists).*

[12] By padding we can handle all n's. See Remark 2.

Proof. Let $\mathbb{F} = \mathsf{GF}(\sqrt{n} = q^k)$ (Galois field of order q^k). We assume that there is an efficiently computable one-to-one efficient mapping $g\colon [\sqrt{n}] \to \mathbb{F}$ (for instance, by computing all the elements in $\mathbb{F}$ and storing them in an array—this requires $\tilde{O}(\sqrt{n})$ space and time). Let $f\colon [n] \to \mathbb{F}^2$ be any one-to-one mapping (say, we can use $f(i) = (g(\lceil i/\sqrt{n} \rceil), g(i \bmod \sqrt{n}))$. We define $\tilde{J}\colon \mathbb{F}^2 \times \mathbb{F} \to 2^{\mathbb{F}^2}$ such that:

$$\tilde{J}((x,y),m) = \left\{(x',y') \;\middle|\; y' = m \cdot (x'-x) + y\right\}.$$

Finally, define:

$$J(i,d) = \left\{ f^{-1}(x,y) \;\middle|\; (x,y) \in \tilde{J}(f(i), g(d)) \right\}.$$

Note that from a geometric view, every $f(i)$ is a point in $\mathbb{F}^2$ and $\tilde{J}(f(i), g(d))$ are all the points on the line with gradient m that pass through $f(i)$. Therefore, **Bullet 1** means that each point belongs to the lines that passes through it and all the lines contains $|\mathbb{F}|$ points, **Bullet 2** means that if line passes through a point, then it belongs to this line and **Bullet 3** means that each two lines with different gradients meet at exactly one point. In what follows, we prove that this construction satisfies the three properties from the statement.

Bullet 1: Fix $i \in [n]$ and $d \in [D]$, and let $(x_i, y_i) = f(i)$. Since $J(i,d)$ consists of solutions of a linear equation over $\mathbb{F}$, then for every $w \in \mathbb{F}$, there exists a $z \in \mathbb{F}$ such that $w = g(d) \cdot (z - x_i) + y_i$. Also, since the equation is linear, the solutions must be distinct. Therefore, $|J(i,d)| = |\mathbb{F}| = \sqrt{n}$. For the second part, observe that $y_i = g(d) \cdot (x_i - x_i) + y_i$, and so $i \in J(i,d)$.

Bullet 2: For every $i, j \in [n]$ and $d \in [\sqrt{n}]$, if $j \in J(i,d)$, then

$$y_j = g(d) \cdot (x_j - x_i) + y_i.$$

By rearranging, we get that

$$y_j + g(d) \cdot (x_i - x_j) = y_i$$

which means that $i \in J(j,d)$, as needed.

Bullet 3: For every $i, j \in [n]$, and all $d_1 \neq d_2 \in [\sqrt{n}]$, $k \in J(i,d_1) \cap J(j,d_2)$ for $k \in [n]$ only if $(x_k, y_k) = f(k)$ and the following two equations hold:

$$y_k = g(d_1) \cdot (x_k - x_i) + y_i$$

and

$$y_k = g(d_2) \cdot (x_k - x_j) + y_j.$$

Since $g(d_1) \neq g(d_2)$, there is only one solution to this pair of equations. □

Remark 2. *In the above construction, we assumed that n is an even power of a prime. We observe that a padding argument can be used to cover all n's. Specifically, it is known that for sufficiently large k, the gap between every two consecutive primes is $p_{k+1} - p_k < p_k^{0.6}$ (see, e.g., [BHP01] for a better bound and history of the problem). Since $p_{k+1}^2 - p_k^2 = (p_{k+1} + p_k)(p_{k+1} - p_k) < 3(p_k^2)^{0.8}$, for sufficiently large integer n, there is a prime p such that $p^2 > n$ and $p^2 - n < 3n^{0.8}$. So, if we use $\mathbb{F} = \mathsf{GF}(p)$, then we can choose arbitrarily $p^2 - n$ unused points from $\lceil p - n/p \rceil \in o(p)$ lines with the same gradient. We do not allow this gradient to be used for sampling poll lists. We next argue that the sample size of each party remains large enough. Since each line with a different gradient intersects each of the chosen lines in one point, then the sample size of each party (i.e., number of valid points on each line) remains $p - o(p) \geq (1 - o(1))\sqrt{n}$ which is enough for the proof to work.*

5 The Scalable Agreement Protocol

In this section we build a protocol that transform an almost-everywhere agreement into full agreement. The theorem is stated next.

Theorem 6 (From almost-everywhere to everywhere)**.** *There is a protocol that translates an almost everywhere into everywhere agreement. The protocol requires $O(1)$ rounds, and each party stores, processes, and sends $\tilde{O}(\sqrt{n})$ bits overall. The transformation is secure against an adversary that statically corrupts up to $1/3 - \epsilon$ fraction of parties for any constant $\epsilon > 0$.*

For simplification, we assume that $n = p^{2k}$, where p is a prime and k is a positive integer, and refer to Remark 2 on how to extend it for all n's. The starting point of our protocol is the output of the *almost everywhere agreement* protocol of King et al. [KSSV06]. The output of their protocol is a bit string seed with the following properties (see Footnote 7):

- **Length**: $|\mathsf{seed}| = \Theta(\log^3 n)$.
- **Entropy**: seed is sampled from a $\left(|\mathsf{seed}| \cdot \left(\frac{2}{3} + \epsilon\right)\right)$-source channel.
- **Agreement**: Each honest party P_i has a seed candidate seed_i, and for $(1 - o(1))$ fraction of the honest parties, $\mathsf{seed}_i = \mathsf{seed}$.

In other words, the almost-everywhere agreement protocol results with a string of $\Theta(\log^3 n)$ bits with min-entropy $\Omega(\log^2 n)$ that is known to $(1 - o(1))$ fraction of parties. The agreement protocol that we achieve in Theorem 6 has exactly the same parameters except that all parties agree on the same string (of length $\Theta(\log^3 n)$ bits and with min-entropy $\Omega(\log^2 n)$).

To achieve our transformation, we use the $J : [n] \times [\sqrt{n}] \to \binom{[n]}{\sqrt{n}}$ and $H, H^{-1} : [n] \times [n^c] \to [n]^d$ functions from Lemma 1 and Corollary 1, respectively, where we use $c = c(n) = \log^4 n$ and $d = O(\log^5 n)$. We refer to Section 2 for an overview of the protocol.

The protocol proceeds in two phases. In the first phase, each party disseminates its belief of seed to a randomly chosen subset of $\approx \sqrt{n}$ parties. Each party, upon every incoming message, independently decides whether to keep the message with some probability and discard it otherwise. At the end of this phase, choosing the parameters correctly, with very high probability, every party ends up with a candidate list $Candidates$ with the guarantee that at least one of the candidates is exactly seed.

In the second phase, there are six rounds of communication. First, each party chooses random poll lists by sampling $d_{i,j} \leftarrow [\sqrt{n} - 2]$ for $j \in [\log^2 n]$, and distributes its choices to its committee $H(i, \mathsf{seed})$. Then, we execute a 3-rounds sub-protocol to distribute the poll requests to the committees of the poll list members (see Section 5.3), who in turn, forward them to the associated poll list member. A party P_ℓ in the jth poll list of P_i counts how many times it received the same poll request from its committee, and if sufficiently many were received, then it sends its candidate string seed_ℓ to P_i directly. Then, for each $j \in [\log^2 n]$, P_i increases the counter of seed candidate if it received it from $2/3$ fraction of the jth poll list members. A seed candidate that was received by P_i and has the highest counter is then elected as the right string.

The protocol's full description is given next.

A. **Phase 1:**

1. **Round 1:**

   Description: Create a set of candidates for seed. One of them will be the right one.
   Filtering rule: P_i chooses $\log n \cdot \sqrt{n}$ parties uniformly at random to accepts messages from them.

   For each party P_i:
   i. P_i initialize $Candidates \leftarrow \emptyset$.
   ii. P_i chooses $\sqrt{n} \cdot \log n$ parties uniformly at random and sends them seed_i.

B. **End of Phase 1:** For each incoming message from P_j, P_i adds seed_j to $Candidates$.

C. **Phase 2, repeat $\log^2 n$ times, in parallel:**

1. **Round 1:**

   Description: Choose a poll list and share the choice with all committee candidates.
   Filtering rule: P_j accepts a message from P_i only if $i \in H^{-1}(j, \mathsf{seed}_j)$, and if the message size is at most $\log n$ bits.

   For each party P_i:
   i. P_i chooses $d_i \leftarrow [\sqrt{n} - 2]$ uniformly at random.
   ii. For each party $P_j \in \bigcup_{\mathsf{seed}' \in Candidates} H(i, \mathsf{seed}')$, P_i sends d_i to P_j.

2. **Rounds 2-4:**

   All the committees runs the **DistributePollRequests** sub-protocol (Section 5.3).

3. **Round 5:**

   Description: Committees forward requests to poll lists.
   Filtering rule: P_ℓ accepts a message from P_u if $u \in H(\ell, \mathsf{seed}_\ell)$ and if the message size is less than $\sqrt{n} \cdot \log^2 n$ bits.

   For each party P_u:
   i. For each message of type "$P_i \to P_\ell$", P_u forwards the message to P_ℓ.

4. **Round 6:**

   Description: Poll list members respond to requests with their seed candidate.
   Filtering rule: P_i accepts a message from P_ℓ only if $\ell \in J(i, d_i)$ and if the message size is less than $\log^4 n$ bits.

   For each party P_ℓ, up to $\sqrt{n} \cdot \log n$ times:
   i. For each incoming message "$P_i \to P_\ell$" that was received more than $|H(\ell, \mathsf{seed}_\ell)|/2$ times, P_ℓ sends seed_ℓ to P_i.

5. **End of Phase 2:** For each P_i: if seed^* was received more than $(2/3) \cdot |J(i, d_i)|$ times, then P_i adds plus 1 to seed^* counter.

D. **End of all $\log^2 n$ times of Phase 2:** Each P_i sets seed_i to be the seed^* with the highest counter.

Agreement. We proceed with the correctness and security of the protocol. We prove that at the end of the protocol, with very high probability, every party holds the right string seed. The fact that the protocol terminates within $O(1)$ rounds is immediate from the description. The proof appears in Section 5.2.

5.1 Efficiency

The protocol consists of exactly seven rounds of communication. The computational complexity of each party depends on two main tasks. The first one is the work needed to compute the H, H^{-1} and J functions. By Theorem 4 and Lemma 1, these functions are efficiently computable.[13] The second one is the work needed to send and process messages, which is what we discuss in the rest of the section.

During Phase 1, each party P_i sends $\sqrt{n} \cdot \log n$ messages each of size $\Theta(\log^3 n)$ bits. Furthermore, since each party processes each incoming message with probability $\log n/\sqrt{n}$, with all but negligible probability, for each P_i it holds that $|Candidates| \in O(\sqrt{n} \cdot \log^3 n)$ and this is also a bound on the number of messages it receives during Phase 1. We now analyze a single iteration of Phase 2:

- **Round 1**:
  - Incoming:$|H^{-1}(j, \mathsf{seed}_j)| \cdot \log n = O(\log^6 n)$ bits.
  - Outgoing: Each party P_i sends messages of size $O(\log n)$ to at most $|Candidates| \cdot |H(i, *)|$ parties. Since $|Candidates| \in O(\sqrt{n} \cdot \log^3 n)$, the communication is $O(\sqrt{n} \cdot \log^9 n)$ bits.
- **Rounds 2-4**: (see Section 5.3.2)
  - Incoming: $O(\log^{12} n \cdot \sqrt{n})$ bits.
  - Outgoing: $O(\log^{11} n \cdot \sqrt{n})$ bits.
- **Round 5**:
  - Incoming:$|H(\ell, \mathsf{seed}_\ell)| \cdot \log^2 n \cdot \sqrt{n} = O(\log^7 n \cdot \sqrt{n})$ bits.
  - Outgoing: Each party P_u sends messages of size $O(\log^2 n \cdot \sqrt{n})$ to $|H^{-1}(u, \mathsf{seed}_u)|$ parties. Thus, the communication is $O(\sqrt{n} \cdot \log^7 n)$ bits.
- **Round 6**:
  - Incoming:$|J(i, d_i)| \cdot \log^4 n = O(\sqrt{n} \cdot \log^4 n)$ bits.
  - Outgoing: Each party P_ℓ sends messages of size $O(\log^4 n)$ to at most $\log n \cdot \sqrt{n}$ parties. So, the communication is $O(\sqrt{n} \cdot \log^5 n)$ bits.

Since Phase 2 consists of $\log^2 n$ parallel repetitions of the above, the total communication complexity (incoming + outgoing) of each party is $O(\sqrt{n} \cdot \log^{14} n)$ bits.

[13] A function f is "efficiently computable" if there is a polynomial $p(\cdot)$ such that for every x, the work that needed to compute $f(x)$ is $|f(x)| \cdot p(|x|)$.

5.2 Correctness

We say that an event happens with high probability (w.h.p) if it happens with probability at least $1 - n^{-c}$, for every constant $c > 0$ (i.e., with all but negligible probability).

Claim 3. *During Phase 2 (Round 1), w.h.p each honest party P_i sends d_i to $H(i, \mathsf{seed})$.*

Proof. By description, an honest P_i sends d_i to $H(i, \mathsf{seed})$ if seed was received during Phase 1. Let X_i be a random variable that counts the number of times that seed was received by P_i. Recall that each honest party sends its candidate to a uniformly random subset of $\sqrt{n} \cdot \log n$ parties. In turn, each honest party accepts messages from a subset of $\sqrt{n} \cdot \log n$ parties sampled uniformly at random. Therefore, for each honest party P_i, the random variable X_i is distributed like a binomial with parameters $\left(\left(\frac{2}{3} + \epsilon - o(1)\right) \cdot n, \frac{\log^2 n}{n}\right)$. Therefore, the probability that party P_i never added seed to its candidate list is

$$\begin{aligned}\Pr[X_i = 0] &= \left(1 - \frac{\log^2 n}{n}\right)^{\left(\frac{2}{3}+\epsilon-o(1)\right)\cdot n} \\ &= e^{-\left(\frac{2}{3}+\epsilon-o(1)\right)\cdot \log^2 n} \leq n^{-\omega(1)}.\end{aligned}$$

By the union bound and since the above bound holds for every $i \in [n]$, we get that

$$\Pr[\exists i : X_i = 0] \leq n \cdot \Pr[X_i = 0] \leq n^{-\omega(1)}.$$

□

Claim 4. *During Phase 2 (Round 5), if P_ℓ appears in less than $\log n \cdot \sqrt{n}$ poll lists, then w.h.p more than $|H(\ell, \mathsf{seed})|/2$ of the members in $H(\ell, \mathsf{seed})$ forward all poll requests to P_ℓ.*

Proof. Recall that seed is sampled from a $\left(|\mathsf{seed}| \cdot \left(\frac{2}{3} + \epsilon\right)\right)$-source channel. Further, $|\mathsf{seed}| \cdot \left(\frac{2}{3} + \epsilon\right) \geq \log^2 n$. Thus, there are less than $n/3$ parties that are either corrupted or honest but without knowledge. Therefore, by Corollary 1, with probability at least $1 - 1/n^{\log(n)-4}$ for each $i \in [n]$, the ith committee $H(i, \mathsf{seed})$ contains a majority of honest parties with knowledge, and from the correctness of **DistributePollRequests** sub-protocol (Section 5.3) they will hold P_ℓ poll requests. □

Claim 5. *Fix $d_i \in [\sqrt{n} - 2]$ and $0 < \eta \leq 1/3$. For each honest party P_i, if at most $\frac{1}{3} - \eta$ fraction of the parties in $J(i, d_i)$ are either corrupted or honest without knowledge, then at every iteration of Phase 2, w.h.p P_i increases the counter of some candidate.*

Proof. An honest party $P_\ell \in J(i, d_i)$ does not respond to P_i in one of two cases: First, if P_ℓ did not receive P_i's request. Second, if P_ℓ received $\sqrt{n} \cdot \log(n)$ requests. By Claim 4, except with negligible probability, the first case will not happen. From properties 2 and 3 of Lemma 1, for all corrupted parties $\tilde{P}_j \notin J(i, d_i)$ and for all $d \in [\sqrt{n} - 2]$, it holds that $|J(i, d_i) \cap J(j, d)| \leq 1$. Therefore, each corrupted party $\tilde{P}_j \notin J(i, d_i)$ can send requests only to at most 1 member of $J(i, d_i)$. So, the number of members in $J(i, d_i)$ that the adversary can send at least $\sqrt{n} \cdot \log(n)$ requests are at most:

$$\begin{aligned}&\frac{\left(\left(\frac{1}{3} - \epsilon\right) \cdot n - \left(\frac{1}{3} - \eta\right) \cdot \sqrt{n}\right)}{\sqrt{n} \cdot \log(n) - \left(\frac{1}{3} - \eta\right) \cdot \sqrt{n}} \leq \frac{n}{\left(\log(n) - \frac{1}{3} + \eta\right) \cdot \sqrt{n}} \leq \\ &\frac{\sqrt{n}}{\left(\log(n) - \frac{1}{3}\right)} \leq \frac{3}{2} \cdot \sqrt{n} \cdot \log^{-1} n \in o(\sqrt{n}).\end{aligned}$$

So, at least

$$\left(1-\frac{1}{3}+\eta-o(1)\right)\cdot\sqrt{n}>\left(1-\frac{1}{3}\right)\cdot\sqrt{n}$$

parties will send the same candidate. Thus, some counter will be increased. □

Lemma 2. *At every iteration of Phase 2, each honest party increases the counter of some candidate with probability at least ϵ.*

Proof. Assume, by contradiction that the claim is false. Without loss of generality, assume that P_1 is the honest party with the lowest probability to increase the counter of some candidate. Let $\epsilon' < \epsilon$ be the smallest probability that at every iteration of Phase 2, P_1 increases the counter of some candidate. Since d_1 is chosen independently in every iteration of Phase 2, and by Claim 5, the number of parties that do not respond with the right seed is at least

$$(1-\epsilon')\cdot(\sqrt{n}-2)\cdot|J(1,d_1)|\cdot\left(\frac{1}{3}-o(1)\right),$$

where the first and second terms represent the number of "bad" choices of d_1, the third term is the number of parties in each poll list, and the fourth term is the fraction of corrupted or honest without knowledge parties in the poll list. Simplifying this expression, we get

$$\begin{aligned}(1-\epsilon')\cdot(\sqrt{n}-2)\cdot|J(1,d_1)|\cdot\left(\frac{1}{3}-o(1)\right)&=\left(\frac{1}{3}-\frac{\epsilon'}{3}-o(1)\right)\cdot n\\&>\left(\frac{1}{3}-\frac{2\epsilon}{3}\right)\cdot n.\end{aligned}$$

Since there are at most $o(n)$ honest parties that do not respond with the right seed, it means that there are at least $\left(\frac{1}{3}-\frac{2\epsilon}{3}-o(1)\right)\cdot n$ corrupted parties. This is a contradiction to the fact that the adversary can corrupt at most $\left(\frac{1}{3}-\epsilon\right)\cdot n$ parties. □

Lemma 3. *At every iteration of Phase 2, if honest party increases the counter of candidate* seed^**, then the probability that* $\mathsf{seed}^*=\mathsf{seed}$ *is at least* $\frac{1}{2}+\epsilon$*.*

Proof. Assume, by contradiction that the claim is false. Without loss of generality, assume that P_1 is the honest party with the lowest probability that $\mathsf{seed}^*=\mathsf{seed}$, and let $\epsilon'<\frac{1}{2}+\epsilon$ be that probability. Let $\alpha\in[0,1)$ be the fraction of d_1 choices such that P_1 does not increase the counter of any candidate. Since d_1 is chosen independently in every run of Phase 2, from Claim 5, and Step C.5 of the protocol, it follows that there are at least

$$\alpha\cdot(\sqrt{n}-2)\cdot|J(1,d_1)|\cdot\left(\frac{1}{3}-o(1)\right)+(1-\alpha)\cdot(1-\epsilon')\cdot(\sqrt{n}-2)\cdot|J(1,d_1)|\cdot\left(\frac{2}{3}+o(1)\right)$$

parties that do not respond with the right seed. Denoting this expression $(*)$ and expanding it, we get

$$\begin{aligned}(*)&=\left(\frac{\alpha}{3}+\left(\frac{2-2\alpha}{3}\right)\cdot(1-\epsilon')\pm o(1)\right)\cdot n>\left(\frac{\alpha}{3}+\left(\frac{2-2\alpha}{3}\right)\cdot\left(\frac{1}{2}-\epsilon\right)\pm o(1)\right)\cdot n\\&=\left(\frac{1}{3}-\frac{2\epsilon}{3}+\frac{2\epsilon\alpha}{3}\pm o(1)\right)\cdot n\geq\left(\frac{1}{3}-\frac{2\epsilon}{3}\pm o(1)\right)\cdot n>\left(\frac{1}{3}-\frac{5\epsilon}{6}\right)\cdot n.\end{aligned}$$

Since there are at most $o(n)$ honest parties that do not respond with the right seed, there are at least $\left(\frac{1}{3}-\frac{5\epsilon}{6}-o(1)\right)\cdot n$ corrupted parties. This is a contradiction to the fact that the adversary can corrupt at most $\left(\frac{1}{3}-\epsilon\right)\cdot n$ parties. □

Theorem 7. *At the end of the protocol w.h.p each honest party outputs* seed.

Proof. Let $X_{i,r}$, for $r \in [\log^2 n]$ and $i \in [n]$, be a random variable such that:

$$X_{i,r} = \begin{cases} 1 & P_i \text{ learned the wrong value at the } r\text{th parallel execution of Phase 2;} \\ 0 & P_i \text{ learned nothing at the } r\text{th parallel execution of Phase 2;} \\ -1 & P_i \text{ learned the right value at the } r\text{th parallel execution of Phase 2.} \end{cases}$$

Let $Z_i = \sum_{r=1}^{\log^2 n} X_{i,r}$. By definition, if $Z_i \geq 0$ then P_i may output $\mathsf{seed}^* \neq \mathsf{seed}$. By Lemma 2 and Lemma 3, for each honest party P_i, it holds that

$$\Pr[X_{i,r} = x] = \begin{cases} \epsilon \cdot \left(\frac{1}{2} - \epsilon\right) & x = 1; \\ 1 - \epsilon & x = 0; \\ \epsilon \cdot \left(\frac{1}{2} + \epsilon\right) & x = -1. \end{cases}$$

Since each time we perform Phase 2, every honest party P_i chooses d_i independently, then the $X_{i,r}$'s are all independent from each other. Thus,

$$\mathbb{E}[Z_i] = \log^2 n \cdot \mathbb{E}[X_{1,r}] = -\log^2 n \cdot 2\epsilon^2.$$

By a union bound and a Hoeffding's inequality, we get that

$$\begin{aligned} &\Pr[\exists i : Z_i \geq 0] \leq n \cdot \Pr[Z_1 \geq 0] = n \cdot \Pr[Z_1 - \mathbb{E}[Z_1] \geq -\mathbb{E}[Z_1]] \\ &\leq n \cdot e^{\frac{-2 \cdot (\log^2 n \cdot 2\epsilon^2)^2}{4 \cdot \log^2 n}} \in n^{-\omega(1)}. \end{aligned}$$

□

5.3 Distribute Poll Requests Sub-Protocol

At the beginning of this (sub)protocol each committee holds the poll list of the corresponding party, and its goal is to convey the poll request to the committees associated to each party in the poll list. There are two challenges: The high level idea of our protocol is to imagine the parties as residing on an $\sqrt{n} \times \sqrt{n}$ grid, where the rows are defined by $J(*, \sqrt{n} - 1)$ and the columns by $J(*, \sqrt{n})$. At every round of communication, communication is allowed only over rows or over columns. So, each committee can communicate only with $\sqrt{n}$ other committees in every round.

Our protocol consists of three rounds, where each committee forwards each request through the committee that corresponds to the intersection between the corresponding row and column. This limits the set of committees from which we can accept messages. Before we send the poll requests from the intersection committee to their destination, each such intersection committee tells its destination how many requests it is planning to forward. Only if the total number is smaller than $\approx \sqrt{n}$, they will be allowed to forward the poll requests. This extra counting phase is used to prevent targeting of specific parties by the attacker (by choosing a poll list that contains the targeted party).

Notations: Let Q be a quorum of n committees, the committee that corresponding to party P_i dented by Q_i. When we say that "Q_i send message to Q_j" we mean that every party in Q_i sends the message to every party in Q_j, and each party in Q_j drop every message that received less than $|Q_i|/2$ times.

DistributePollRequests Sub-Protocol

Input: Each committee Q_i holds d_i.

Output: Each committee Q_ℓ holds $S_\ell = \{P_i \mid i \in [n], \ell \in J(i, d_i)\}$ if $|S_\ell| \leq \log n \cdot \sqrt{n}$.

1. **Round 1:**
   Filtering rule: Q_t accepts messages from Q_i only if $i \in J(t, \sqrt{n} - 1)$, and if message size is less than $3 \log n$ bits.
   For each committee Q_i:

   i. For each ℓ in $J(i, d_i)$, Q_i sends "$P_i \to P_\ell$" to Q_t, where $t = J(i, \sqrt{n} - 1) \cap J(\ell, \sqrt{n})$.

2. **Round 2:**
   Filtering rule: Q_ℓ accepts messages from Q_t only if $t \in J(\ell, \sqrt{n})$, and if message size is less than $\log n$ bits.
   For each committee Q_t:

   For each $\ell \in J(t, \sqrt{n})$:

   i. Q_t prepare the set $S_{t,\ell} = \{P_i \mid$ for all message of type "$P_i \to P_\ell$"$\}$.
   ii. Q_t sends $|S_{t,\ell}|$ to Q_ℓ.

3. **Round 3:**
   Filtering rule: Q_ℓ accepts messages from Q_t only if $\sum_{t \in J(\ell, \sqrt{n})} |S_{t,\ell}|$ is less than $\log n \cdot \sqrt{n}$, and if message size is at most $\log n \cdot |S_{t,\ell}|$ bits.
   For each committee Q_t:

   i. For each $\ell \in J(t, \sqrt{n})$, Q_t sends $S_{t,\ell}$ to Q_ℓ.

5.3.1 Correctness

At first we mention that quorum is assumed, therefore the proof holds only if every committee has majority of "honest with knowledge" parties.

Claim 6. *During round 1 each committee Q_i sends exactly 1 poll request to each committee associated with a party in $J(i, \sqrt{n} - 1)$.*

Proof. By property 3 of Lemma 1, during round 1 for each Q_i, t always exists and it is unique (i.e., $|J(i, \sqrt{n} - 1) \cap J(\ell, \sqrt{n})| = 1$). Combined with property 2, for every $\ell_1 \neq \ell_2 \in J(i, d_i)$, it holds that $t_1 \neq t_2$. □

By property 2 of Lemma 1 and the above claim, each committee Q_t will accept all poll requests. Since Q_t holds $\sqrt{n}$ poll request, then for each $\ell \in J(t, \sqrt{n})$, it holds that $\log(|S_{t,\ell}|) < \log n$. Combining this with property 2, at the end of round 2, each committee Q_ℓ will know $\sum_{t \in J(\ell, \sqrt{n})} |S_{t,\ell}|$. If the latter sum is less than $\log n \cdot \sqrt{n}$, after round 3, it will know $\bigcup_{t \in J(\ell, \sqrt{n})} S_{t,\ell} = \{P_i \mid i \in [n], \ell \in J(i, d_i)\}$, as needed.

5.3.2 Efficiency

For each committee we analyze both incoming and outgoing communication.

- **Round 1**:
  - Incoming: $|J(t, \sqrt{n}-1)| \cdot 3 \log n = O(\log n \cdot \sqrt{n})$ bits.
  - Outgoing: Each committee sends messages of size $|\text{“}P_i \to P_\ell\text{”}|$ to $|J(i, d_i)|$ committees, so the communication is $O(\log n \cdot \sqrt{n})$ bits.
- **Round 2**:
  - Incoming: $|J(\ell, \sqrt{n})| \cdot \log n = O(\log n \cdot \sqrt{n})$ bits.
  - Outgoing: Each committee sends messages of size at most $\max_{\ell \in J(t,\sqrt{n})} |S_{t,\ell}|$ to $|J(t, \sqrt{n})|$ committees, so the communication is $O(\log n \cdot \sqrt{n})$ bits.
- **Round 3**:
  - Incoming: $\log^2 n \cdot \sqrt{n} = O(\log^2 n \cdot \sqrt{n})$ bits.
  - Outgoing: Each committee forwards at most $|J(t, \sqrt{n}-1)|$ poll requests of size $|\text{“}P_i \to P_\ell\text{”}|$ to committees, so the communication is $O(\log n \cdot \sqrt{n})$ bits.

Since each party is a member of $O(\log^5 n)$ committees, and each committee contains $O(\log^5 n)$ members, the total communication (outgoing + incoming) is $O(\log^{12} n \cdot \sqrt{n})$ bits. The bound on the computational complexity follows similarly.

6 Multi-Party Computation, Byzantine Agreement, Broadcast, and More

In Section 5 we showed a transformation from almost-everywhere agreement to everywhere agreement. More specifically, the method shows how to guarantee agreement by all parties on a string that has some non-trivial (min-)entropy. Here, we explain how to use our protocol to get classical abstractions such as Byzantine agreement, broadcast, and leader or committee election, while preserving scalability.

To this end, it is convenient to define an intermediate abstraction. On the one hand, this intermediate model would relatively straightforwardly imply all of the above mentioned (and possibly more) applications. On the other hand, it is not hard to show how to realize this abstraction with our agreement protocol from Section 5. We proceed with the description of the model. Then, we explain how to realize it (Section 6.1) and finally we explain how it implies all of the above mentioned applications (Section 6.2). Throughout this section we assume that n, the number of participants in the protocol, is of the form $n = p^{2k}$, where p is a prime and k is an integer (see in Remark 2 how to generalize to all n's).

The quorum model. The model is an extension of the full-information model that we described in Section 3. In addition to the regular n parties, there are n additional special parties called together “quorum”, where single party called “committee”. Each committee has a unique ID and the IDs are common knowledge. Assume that the committee IDs are $1, \ldots, n$.

In terms of functionality, committee are just like parties: any party (either committee or regular) can send a message to any other party (either committee or regular). That is, the communication

network consists of $2n$ nodes and it is fully connected (i.e., clique configuration). However, unlike parties, committees do not have a private source of randomness and therefore can perform only arbitrary deterministic computation. Furthermore, during each round, the committees' message size is hardcoded in the protocol description (of course, committees may send shorter messages and pad appropriately).

The corruption model is the same as in the standard full information model, restricting corruptions to regular parties. That is, an adversary can corrupt and control only regular parties and not committee parties. Still, the adversary can see all of the communication including all messages sent to or from a committee party. For concreteness, we assume that the adversary can corrupt $1/3 - \epsilon$ fraction of the regular parties.

6.1 Realizing the Quorum Model

In the following theorem we explain how to realize the quorum model given our agreement protocol. Specifically, we show how to reduce a protocol in the quorum model to the standard model with only "small" overhead (in storage, computation, and communication).

Theorem 8. *Every protocol in the quorum model can be translated to a protocol in the standard full information model with all but negligible probability of failure. Furthermore, if each party in the protocol in the quorum model stores, processes and sends C bits within r rounds, then the protocol in the standard full information model stores, processes and sends $\tilde{O}(C) + \tilde{O}(\sqrt{n})$ bits within $r + \tilde{O}(1)$ rounds.*

Proof. By Theorem 6, there is a protocol so that at the end of it, all parties agree on a string seed sampled from a $(\log^2 n)$-source channel and of length $|\mathsf{seed}| = \Theta(\log^3 n)$. Using seed, we invoke the function H from Corollary 1, where we use B as the set of corrupted parties, $x = \mathsf{seed}$ and $c = \log^4 n$. The result of this is a collection of n multisets with the guarantee that with all but negligible probability of error all of them contain a majority of honest parties.

Therefore, we define each of these n multisets as a quorum, where a message from a committee is valid only if it is received from the majority of the multiset. Now, we get the properties of the quorum model, and we can invoke the given protocol in the quorum model.

Overall, the cost of the translation depends on the cost of the full agreement protocol times the cost of communicating to committee parties. The first is $\tilde{O}(1)$ rounds and $\tilde{O}(\sqrt{n})$ storage, processing, and communication per party. The latter depends on the committee size which is $O(\log^5 n)$ which gets hidden in the $\tilde{O}$ notation. □

6.2 Applications in the Quorum Model

We give examples of how to realize several applications of interest within the quorum model.

Byzantine agreement (BA). In this problem, every regular party has an input bit and the committee parties do not have any input. The goal is for the regular parties to reach an agreement on an input which is held by one of the honest parties. In the quorum model this can be achieved as follows.

Byzantine Agreement:

1. For each $i \in [n]$, a regular party i sends its input bit to committee party i.
2. The quorums count and distributes among themselves, in a tree-like fashion, the bit that is held by most parties.

3. Committee party i sends this bit to regular party i.

Denote γ the arity of the tree in the above communication. The round complexity is $2 \cdot \log_\gamma n + 2$. Furthermore, each party stores, sends and processes $\gamma \cdot \log n$ bits. If γ is constant, the protocol requires $O(\log n)$ rounds and each party stores, sends and processes $O(\log n)$ bits. If $\gamma = n^\epsilon$, for a constant $\epsilon > 0$, the protocol requires constant rounds and each party stores, sends and processes $O(n^\epsilon \cdot \log n)$ bits.

Broadcast. In broadcast, one of the regular parties has an input and its goal is to distribute it to all other regular parties. In the quorum model this can be achieved as follows, assuming that party i wants to broadcast message m.

Broadcast:

1. Regular party i sends its message m to committee party i.
2. Committee party i sends the message upstream in tree-like communication pattern and the root distributes it downstream over the same tree.
3. Committee party j sends its message to regular party j for every $j \in [n]$.

The complexity of the protocol is just like the BA complexity from above.

Quorum/committee/leader election. Like in Section 6.1, it is immediate to create quorum by combining Theorem 6 and Corollary 1. For committee election, we just need to fix the first index of the H function. To obtain a leader we first elect a poly-logarithmic size committee and then run a standard leader election protocol that could have polynomially many rounds and is not necessarily scalable. For instance, one can use the HEAVYWEIGHT-LEADER-ELECTION protocol from [KSSV06], which is an adaptation of Feige's protocol [Fei99] to the point-to-point full information model.

To summarize, plugging these protocols into Theorem 8 along with the almost everywhere agreement protocol of [KSSV06], we get the following (full) BA, broadcast, quorum/committee/leader election protocols in the standard full information model. (In the above protocols where communication is done over a tree, i.e., BA and broadcast, we use constant γ.)

Theorem 9. *There are BA, broadcast, quorum/committee/leader election protocols in the full information model with the following properties. They have negligible probability (in n) of failure, they require $\tilde{O}(1)$ rounds and in each of them, every party communicates $\tilde{O}(\sqrt{n})$ bits throughout.*

Multi-party computation. The above agreement tasks can be thought of as special cases of secure multi-party computation (MPC) [GMW87, BGW88, CCD87]. MPC protocols enable a set of mutually distrusting parties to compute a function on their private inputs, while guaranteeing various properties such as correctness, privacy, independence of inputs, and more. We consider the problem of scalable MPC in the peer-to-peer synchronous communication model with private channels (i.e. the adversary can't see the content of messages between honest parties).

Feasibility results for (non-scalable) MPC have been long known, e.g., the BGW [BGW88] protocol gives a method for computing an arbitrary function with communication cost that grows multiplicatively with the circuit size of the function and some polynomial in the number of parties. That is, the communication and work of each of the n parties is $s \cdot \mathsf{poly}(n)$ when computing a function represented as a circuit of size s. The question of scalable MPC, i.e., protocols where the dominant

term in the complexity is just the circuit size, is still an active topic and modern results achieve MPC protocols with strong security guarantees and communication complexity $\tilde{O}(s + \mathsf{poly}(n))$.

Plugging in our Theorem 1 into the protocol framework of [DKM+17], we get the following result.

Theorem 10 (Scalable MPC)**.** *For any circuit with size s and depth d, there is a statistically maliciously secure MPC protocol tolerating $1/3 - \epsilon$ fraction of static corruptions for any $\epsilon > 0$ that requires $\tilde{O}(d)$ rounds, and each party need to send store and process $\tilde{O}(s/n + \sqrt{n})$ bits.*

At a high level, the above MPC is obtained by using our agreement protocol to generate a *quorum*: assign to each party its own representative (small and balanced) committee where there is a strong majority of honest parties. Then, we distribute the gates of the circuit to these committees. Each gate is evaluated by its assigned committees using some standard MPC (e.g., BGW).

7 An $\Omega(\sqrt{n})$ Lower Bound for Byzantine Agreement

In this section we prove a lower bound on the number of messages that an honest party need to send *or* process in every scalable Byzantine agreement protocol with static filtering. In fact, for our lower bound to apply it suffices to consider a weaker adversarial model than the model where our upper bound applies: we only need the peer-to-peer authenticated channels model. Namely, our adversary does not need to see messages sent between honest parties.

Theorem 11. *Let Π be any protocol that computes byzantine agreement with probability $p > 2/3$ and secure against adversaries that statically corrupt at most t parties. Then, there is an honest party that sends or processes at least $\Omega(t/\sqrt{n})$ messages.*

The main idea for the proof of Theorem 11 is to show that if the conclusion of the theorem is false, then there must be a "communication bottleneck". Specifically, there is a special set of parties that w.h.p communicate only within themselves and with another set of parties that is somewhat small. The latter will be smaller than the corruption budget of the adversary. Looking ahead, once we prove this, we can design an attack that controls this small set and thereby "isolate" the spacial set. The statement about the existence of the above sets is given next. Its proof is given below in Section 7.1.

Lemma 4 (Bottleneck lemma)**.** *Given a protocol Π in the static filtering model such that each honest party is allowed to send and process at most $o(t/\sqrt{n})$ messages during the protocol, where t is the number of corrupted parties, then for any input configuration B (the protocol specification denoted by Π_B), there exist sets $S, T \subset [n]$, such that:*

1. *$|T| \leq t$.*
2. *$|S| \in \omega(1)$.*
3. *$S \cap T = \emptyset$.*
4. *With probability at least $1 - o(1)$ over the randomness of the protocol, during an execution of the protocol, every party in S communicates (sends and receives messages) only with parties from $S \cup T$.*

We apply this lemma in the context of Byzantine Agreement. Let Π be an any Byzantine Agreement protocol, and let $\Pi' = \Pi_{B_0} || \Pi_{B_1}$ (i.e., an execution of the first protocol followed by an execution of the second one), where B_b is the input configuration such that all parties' initial bit is

b. Applying Lemma 4 on Π' gives the two sets S and T. An adversary can choose the initial bit of all parties in S to be 0, and 1 for the rest, and corrupt the set T. Thus, w.h.p it can force the parties in S to believe that all parties start with 0. At the same time, it can force the rest of the parties to believe that all parties start with 1. This is a contradiction to the correctness property. We formalize this next.

Definition 4. *Let* view_i *be a random variable that contains all the data (statements, messages, initial bit, and randomness) of party* P_i*. For a configuration* W *(e.g., fixed inputs), we explicitly write* view_i^W *for the view of party* i *given configuration* W*. We say that a party cannot distinguish between two configurations,* W_1 *and* W_2*, if the statistical distance between* $\mathsf{view}_i^{W_1}$ *and* $\mathsf{view}_i^{W_2}$ *is 0 (i.e., the random variables are identically distributed).*

Proof of Theorem 11 using Lemma 4. Assume for contradiction that the statement is false. Then, for every bit assignment B, each party sends and processes at most $o(t/\sqrt{n})$ messages during the execution of Π_B. Let $\Pi' = \Pi_{B_0}||\Pi_{B_1}$. Let $S, T \subset [n]$ be the sets from Lemma 4 on Π', and let B' be the assignment such that $B'(i) = \begin{cases} 0 & i \in S \\ 1 & \text{otherwise.} \end{cases}$

We define the following worlds:

- **Zero world**: An honest execution of Π_{B_0}.
- **One world**: An honest execution of Π_{B_1}.
- **Hybrid World**: An execution of $\Pi_{B'}$ in the presence of the following attack. An adversary corrupts the set of parties in T (from Lemma 4) and simulates an execution of Π_{B_0} and Π_{B_1}, where each corrupted party sends messages as follows: Communication with parties in S are simulated using the transcript of Π_{B_0} while the rest of the communication is simulated using the transcript of Π_{B_1}.

With probability at least $1 - o(1)$, for each party $i \in S$, $\mathsf{view}_i^{Hybrid} = \mathsf{view}_i^{Zero}$, and for each party $i \in [n] \setminus (S \cup T)$, $\mathsf{view}_i^{Hybrid} = \mathsf{view}_i^{One}$. Therefore, with probability at least $(1 - o(1)) \cdot p$ each party in S outputs 0 (the output of Zero World), and each party in $[n] \setminus (S \cup T)$ outputs 1 (the output of One World). Since parties in S never communicate with parties in $[n] \setminus (S \cup T)$, with probability at least $((1 - o(1)) \cdot p)^2 = (1 - o(1)) \cdot p^2 > 1/3$, there exist parties that end up with different outputs. Therefore, on the initial assignment $\Pi_{B'}$, Byzantine Agreement is computed correctly only with probability $p < 2/3$ which is a contradiction. □

Remark 3 (Efficiency of attack)**.** *We present the attack without caring about computational efficiency. In particular, our attack would need to know the probabilities in which certain events occur within a typical execution of the protocol. These probabilities could be approximated within sufficient accuracy using standard techniques (repetition and tail bounds), so we believe that the attack could be made efficient, but leave this for future work.*

7.1 Proof of the Bottleneck Lemma (Lemma 4)

Recall that we consider static filtering rules, namely, who communicates with whom is predetermined in the protocol specification. In particular, it means that the probability that some party talks to some other party is well defined from the description of the protocol. More precisely, for every two parties P_i and P_j, there are two associated indicator random variable $X_{i \to j}$ and $X_{i \leftarrow j}$ defined as:

- $X_{i\to j} = 1$ if and only if throughout the execution of the protocol party i sends a message to party j.
- $X_{i\leftarrow j} = 1$ if and only if throughout the execution of the protocol party j expects to process a message from party i.
- $X_{i\leftrightarrow j} = 1$ if and only if throughout the execution of the protocol party j process a message from party i. We note that $X_{i\leftrightarrow j} \not\equiv X_{i\leftarrow j} \cdot X_{i\to j}$, Since party i may send a message to party j, and party j may expect to process a message from party i, but this can occur in different rounds. Nevertheless, it holds that $X_{i\leftrightarrow j} \leq X_{i\leftarrow j} \cdot X_{i\to j}$.

We define

$$p_{i\to j} = \Pr[X_{i\to j} = 1] \qquad \text{and} \qquad p_{i\leftarrow j} = \Pr[X_{i\leftarrow j} = 1] \qquad \text{and} \qquad p_{i\leftrightarrow j} = \Pr[X_{i\leftrightarrow j} = 1],$$

where these probabilities are over the randomness of the protocol. Since $X_{i\leftrightarrow j} \leq X_{i\leftarrow j} \cdot X_{i\to j}$, it follow that

$$p_{i\leftrightarrow j} \leq p_{i\to j} \cdot p_{i\leftarrow j}.$$

Given any protocol Π, for any party i, we can define the set of parties with whom it communicates often and those with whom it does not. Specifically, for each party i, we consider four sets $H_{out}(i)$, $L_{in}(i)$, $H_{in}(i)$, and $L_{in}(i)$, defined as follows:

- $H_{out}(i) = \{j \mid p_{i\to j} \in \Omega(\frac{1}{\sqrt{n}})\}$.
  This is the set of parties to whom i sends messages rather often.
- $H_{in}(i) = \{j \mid p_{j\leftarrow i} \in \Omega(\frac{1}{\sqrt{n}})\}$.
  This is the set of parties that expect to process messages from i rather often.
- $L_{out}(i) = [n] \setminus H_{out}(i)$ and $L_{in}(i) = [n] \setminus H_{in}(i)$

With this notation, we can define the sets L_{out} and L_{in} as the set of parties that communicate rather

- $L_{out} \equiv \{i \mid \sum_{j\in L_{out}(i)} p_{i\leftrightarrow j} \in \Omega(1)\}$.
- $L_{in} \equiv \{i \mid \sum_{j\in L_{in}(i)} p_{j\leftrightarrow i} \in \Omega(1)]\}$.

Claim 7. *Assume that a protocol Π is in the static filtering model, and each honest party is allowed to send and process at most $o(t/\sqrt{n})$ messages during the protocol, where t is the number of corrupted parties. Then,*

1. *For every $i \in [n]$, $|H_{out}(i)| \in o(t)$.*
2. *For every $i \in [n]$, $|H_{in}(i)| \in o(t)$.*
3. *$|L_{out}| \in o(n)$.*
4. *$|L_{in}| \in o(n)$.*

Proof. For the first bullet, assume towards contradiction that $|H_{out}(i)| \in \Omega(t)$. Further, Let $X = \sum_{j\in[n]} X_{i\to j}$. By linearity of expectation, we have that

$$\mathbb{E}[X] = \sum_{j\in[n]} p_{i\to j} \geq \sum_{j\in H_{out}(i)} p_{i\to j} \geq |H_{out}(i)| \cdot \min_{j\in H_{out}(i)} p_{i\to j}.$$

Since $|H_{out}(i)| \in \Omega(t)$ and $\min_{j\in H_{out}(i)} p_{i\to j} \in \Omega(1/\sqrt{n})$ it follow that $\mathbb{E}[X] \in \Omega(t/\sqrt{n})$. This is a contradiction since each party sends $o(t/\sqrt{n})$ messages throughout the protocol.

For the second bullet, assume towards contradiction that $|H_{in}(i)| \in \Omega(t)$. Let $X = \sum_{j\in[n]} X_{j\leftrightarrow i}$. Consider an adversary that corrupts t parties from $H_{in}(i)$ (or all parties in $H_{in}(i)$ if $|H_{in}(i)| \leq t$). Denote the set of corrupted parties by $C \subseteq H_{in}(i)$. Then, by linearity of expectation,

$$\mathbb{E}[X] = \sum_{j\in[n]} p_{j\leftrightarrow i} \geq \sum_{j\in C} p_{j\leftrightarrow i}.$$

Thus, if an adversary corrupts all parties in C and floods i, then

$$\sum_{j\in C} p_{j\leftrightarrow i} = \sum_{j\in C} p_{j\leftarrow i} \geq |C| \cdot \min_{j\in C} p_{j\leftarrow i}.$$

Since $|C| \in \Omega(t)$ and $\min_{j\in C} p_{j\leftarrow i} \in \Omega(1/\sqrt{n})$ it follow that $\mathbb{E}[X] \in \Omega(t/\sqrt{n})$. Overall, we obtained a contradiction since each party processes $o(t/\sqrt{n})$ messages throughout the protocol.

For the third bullet, assume for contradiction that $|L_{out}| \in \Omega(n)$. For $i \in L_{out}$,

$$\sum_{j\in L_{out}(i)} p_{i\leftrightarrow j} \leq \sum_{j\in L_{out}(i)} p_{i\leftarrow j} \cdot p_{i\to j} \leq \max_{j\in L_{out}(i)} p_{i\to j} \cdot \sum_{j\in L_{out}(i)} p_{i\leftarrow j}$$

Since $\sum_{j\in L_{out}(i)} p_{i\leftrightarrow j} \in \Omega(1)$ and $\max_{j\in L_{out}(i)} p_{i\to j} \in o(1/\sqrt{n})$, it must hold that $\sum_{j\in L_{out}(i)} p_{i\leftarrow j} \in \omega(\sqrt{n})$. Now, since $|L_{out}| \in \Omega(n)$ it follow that

$$\sum_{i\in[n]} \sum_{j\in[n]} p_{i\leftarrow j} \geq \sum_{i\in L_{out}} \sum_{j\in L_{out}(i)} p_{i\leftarrow j} \in \omega(n \cdot \sqrt{n}).$$

Thus, by averaging, there is a $j \in [n]$ such that

$$\sum_{i\in[n]} p_{i\leftarrow j} \in \omega(\sqrt{n})$$

Also, by averaging, there is a set $S \subseteq [n]$ of size t such that:

$$\sum_{i\in S} p_{i\leftarrow j} \geq \frac{t}{n} \cdot \sum_{i\in[n]} p_{i\leftarrow j} \in \omega(\frac{t}{\sqrt{n}}).$$

Thus, if an adversary corrupts all parties in S and floods j, then:

$$\sum_{i\in S} p_{i\leftrightarrow j} = \sum_{i\in S} p_{i\leftarrow j} \in \omega(\frac{t}{\sqrt{n}}).$$

This is a contradiction since each party processes at most $o(t/\sqrt{n})$ messages throughout the protocol.

For the last (fourth) bullet, assume for contradiction that $|L_{in}| \in \Omega(n)$. For $i \in L_{in}$,

$$\sum_{j \in L_{in}(i)} p_{j \leftrightarrow i} \leq \sum_{j \in L_{in}(i)} p_{j \leftarrow i} \cdot p_{j \rightarrow i} \leq \max_{j \in L_{in}(i)} p_{j \leftarrow i} \cdot \sum_{j \in L_{in}(i)} p_{j \rightarrow i}.$$

Since $\sum_{j \in L_{in}(i)} p_{j \leftrightarrow i} \in \Omega(1)$ and $\max_{j \in L_{in}(i)} p_{j \leftarrow i} \in o(1/\sqrt{n})$, the above means that $\sum_{j \in L_{in}(i)} p_{j \rightarrow i} \in \omega(\sqrt{n})$. In turn, since $|L_{in}| \in \Omega(n)$ it follows that $\sum_{i \in [n]} \sum_{j \in [n]} p_{j \rightarrow i} \geq \sum_{i \in L_{in}} \sum_{j \in L_{in}(i)} p_{j \rightarrow i} \in \omega(n \cdot \sqrt{n})$. This is a contradiction since $t \leq n$ and each party sends at most $o(t/\sqrt{n})$ messages during the protocol. □

Observation 1. *For each party $i \notin L_{out}$ (resp. L_{in}), the probability that there exists a party $j \in L_{out}(i)$ (resp. $L_{in}(i)$) such that party j (resp. i) ever processed a message from party i (resp. j) is $o(1)$.*

Proof. The proof follows from a Markov inequality. We only give the proof for the first case and note that the second case (which is in parentheses) is proven analogously. Let $X = \sum_{j \in L_{out}(i)} X_{i \leftrightarrow j}$. So,

$$\Pr[\exists j \in L_{out}(i) : X_{i \leftrightarrow j} = 1] = \Pr[X \geq 1].$$

From Markov's inequality,

$$\Pr[X \geq 1] \leq \mathbb{E}[X] = \sum_{j \in L_{out}(i)} p_{i \leftrightarrow j} \in o(1).$$

□

We can finally conclude the proof of Lemma 4. Let:

- $k = k(n) = \max_{i \in [n]}(|H_{in}(i)| + |H_{out}(i)|)$.
- $\ell = \ell(n) = \max_{i \in [n] \setminus (L_{in} \cup L_{out})}(\Pr[\exists j \in L_{out}(i) : X_{i \leftrightarrow j} = 1] + \Pr[\exists j \in L_{in}(i) : X_{j \leftrightarrow i} = 1])$.
- $m = m(n) = \frac{\min(t/k, 1/\ell, n)}{\log\log(\min(t/k, 1/\ell, n))}$. (Assume $t/0 = 1/0 = \infty$.)

By Observation 1, we get that $\ell \in o(1)$, and by Item 1 in Claim 7 and Item 2 in Claim 7 it follow that $k \in o(t)$. So, $m \in \omega(1) \cap o(n)$. Let $S \subset [n] \setminus (L_{in} \cup L_{out})$ be an arbitrary subset of size m, and let $T = \left(\bigcup_{i \in S} H_{out}(i) \cup H_{in}(i)\right) \setminus S$.

1. $|T| \leq |S| \cdot k \leq \frac{t}{k \cdot \log\log(t/k)} \cdot k = \frac{t}{\log\log(t/k)} \in o(t)$, as needed.
2. Since $m \in \omega(1) \cap o(n)$, it suffices to show that $|[n] \setminus (L_{in} \cup L_{out})| \geq m$. By Item 3 in Claim 7 and Item 4 in Claim 7, it follows that $|L_{in} \cup L_{out}| \in o(n)$. This means that $|[n] \setminus (L_{in} \cup L_{out})| \in \Omega(n)$, as needed.
3. Follows directly from the construction of T.
4. For every $i \in S$, it holds that

   $$\bigcup_{i \in S} H_{out}(i) \cup H_{in}(i) \subseteq S \cup \bigcup_{i \in S} H_{out}(i) \cup H_{in}(i) = S \cup T.$$

   This means that if party $i \in S$ communicates with party $j \in [n] \setminus (S \cup T)$, then j must be either in $L_{in}(i)$ or in $L_{out}(i)$. By a union bound, the probability that there exists a party in S that communicates with party not in $S \cup T$ is at most

   $$\Pr[\exists i \in S, j \in L_{out}(i) \colon X_{i \leftrightarrow j} = 1] + \Pr[\exists i \in S, j \in L_{in}(i) \colon X_{j \leftrightarrow i} = 1] \leq |S| \cdot \ell$$
   $$\leq \frac{1}{\ell \cdot \log\log(1/\ell)} \cdot \ell \in o(1).$$

References

[ACD+19] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC*, pages 317–326, 2019.

[BCDH18] Elette Boyle, Ran Cohen, Deepesh Data, and Pavel Hubácek. Must the communication graph of MPC protocols be an expander? In *Advances in Cryptology - CRYPTO*, pages 243–272, 2018.

[BCG21] Elette Boyle, Ran Cohen, and Aarushi Goel. Breaking the $O(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 319–330, 2021.

[BGH13] Nicolas Braud-Santoni, Rachid Guerraoui, and Florian Huc. Fast byzantine agreement. In *ACM Symposium on Principles of Distributed Computing, PODC*, pages 57–64, 2013.

[BGT13] Elette Boyle, Shafi Goldwasser, and Stefano Tessaro. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In *Theory of Cryptography Conference, TCC*, pages 356–376, 2013.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, STOC*, pages 1–10, 1988.

[BH08] Zuzana Beerliová-Trubíniová and Martin Hirt. Perfectly-secure MPC with linear communication complexity. In *Theory of Cryptography, TCC*, pages 213–230, 2008.

[BHP01] Roger C Baker, Glyn Harman, and János Pintz. The difference between consecutive primes, ii. *Proceedings of the London Mathematical Society*, 83(3):532–562, 2001.

[BKLL20] Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In *Theory of Cryptography Conference, TCC*, pages 353–380, 2020.

[Bor96] Malte Borderding. Levels of authentication in distributed agreement. In *Distributed Algorithms, 10th International Workshop, WDAG*, pages 40–55, 1996.

[BR94] Mihir Bellare and John Rompel. Randomness-efficient oblivious sampling. In *35th Annual Symposium on Foundations of Computer Science, FOCS*, pages 276–287. IEEE Computer Society, 1994.

[CCD87] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract). In *Advances in Cryptology - CRYPTO*, volume 293, page 462, 1987.

[CCLS20] T.-H. Hubert Chan, Kai-Min Chung, Wei-Kai Lin, and Elaine Shi. MPC for MPC: secure computation on a massively parallel computing architecture. In *11th Innovations in Theoretical Computer Science Conference, ITCS*, pages 75:1–75:52, 2020.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In *Advances in Cryptology - CRYPTO*, pages 395–426, 2018.

[CGH+18] Koji Chida, Daniel Genkin, Koki Hamada, Dai Ikarashi, Ryo Kikuchi, Yehuda Lindell, and Ariel Nof. Fast large-scale honest-majority MPC for malicious adversaries. In *Advances in Cryptology - CRYPTO*, pages 34–64, 2018.

[CKS20] Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP. In *34th International Symposium on Distributed Computing, DISC*, pages 25:1–25:17, 2020.

[CL99] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation, (OSDI)*, pages 173–186, 1999.

[CM19] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.

[DI06] Ivan Damgård and Yuval Ishai. Scalable secure multiparty computation. In *Advances in Cryptology - CRYPTO*, pages 501–520. Springer, 2006.

[DIK+08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam D. Smith. Scalable multiparty computation with nearly optimal work and resilience. In *Advances in Cryptology - CRYPTO*, pages 241–261, 2008.

[DIK10] Ivan Damgård, Yuval Ishai, and Mikkel Krøigaard. Perfectly secure multiparty computation and the computational overhead of cryptography. In *Advances in Cryptology - EUROCRYPT*, pages 445–465, 2010.

[DKM+17] Varsha Dani, Valerie King, Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Secure multi-party computation in large networks. *Distributed Computing*, 30:193–229, 2017.

[DKMS12] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Breaking the $O(mn)$ bit barrier: Secure multiparty computation with a static adversary. In *8th Student Conference*, page 64, 2012.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DN07] Ivan Damgård and Jesper Buus Nielsen. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO*, pages 572–590, 2007.

[DPPU88] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[Fei99] Uriel Feige. Noncryptographic selection protocols. In *40th Annual Symposium on Foundations of Computer Science, FOCS*, pages 142–153, 1999.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Comput.*, 1(1):26–39, 1986.

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

[Gil98] David Gillman. A chernoff bound for random walks on expander graphs. *SIAM J. Comput.*, 27(4):1203–1220, 1998.

[GIP+14] Daniel Genkin, Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and Eran Tromer. Circuits resilient to additive attacks with applications to secure computation. In *Symposium on Theory of Computing, STOC*, pages 495–504, 2014.

[GK23] Yuval Gelles and Ilan Komargodski. Scalable agreement protocols with optimal optimistic efficiency. 2023.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, STOC*, pages 218–229, 1987.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil Vadhan. Unbalanced expanders and randomness extractors from parvaresh–vardy codes. In *J. ACM*, 2009.

[HKK08] Dan Holtby, Bruce M. Kapron, and Valerie King. Lower bound for scalable byzantine agreement. *Distributed Comput.*, 21(4):239–248, 2008.

[HM01] Martin Hirt and Ueli M. Maurer. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology - CRYPTO*, pages 101–118, 2001.

[IKP+16] Yuval Ishai, Eyal Kushilevitz, Manoj Prabhakaran, Amit Sahai, and Ching-Hua Yu. Secure protocol transformations. In *Advances in Cryptology - CRYPTO*, pages 430–458, 2016.

[IPS09] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Secure arithmetic computation with no honest majority. In *Theory of Cryptography, TCC*, pages 294–314, 2009.

[KLST11] Valerie King, Steven Lonargan, Jared Saia, and Amitabh Trehan. Load balanced scalable byzantine agreement through quorum building, with full information. In *Distributed Computing and Networking - ICDCN*, pages 203–214, 2011.

[KS09] Valerie King and Jared Saia. From almost everywhere to everywhere: Byzantine agreement with $\tilde{O}(n^{3/2})$ bits. In *Distributed Computing, 23rd International Symposium, DISC*, pages 464–478, 2009.

[KS11] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, 2011.

[KSSV06] Valerie King, Jared Saia, Vishal Sanwalani, and Erik Vee. Scalable leader election. In *17th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 990–999, 2006.

[KSV10] Howard J. Karloff, Siddharth Suri, and Sergei Vassilvitskii. A model of computation for mapreduce. In *SODA*, 2010.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[RB89] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, STOC*, pages 73–85, 1989.

[Zuc97] David Zuckerman. Randomness-optimal oblivious sampling. In *Random Struct. Algorithms*, page 345–367, 1997.