Scalable Agreement Protocols with Optimal Optimistic Efficiency

Yuval Gelles[*] Ilan Komargodski[†]

Abstract

Designing efficient distributed protocols for various agreement tasks such as Byzantine Agreement, Broadcast, and Committee Election is a fundamental goal with many applications, including most secure multiparty computation (MPC) protocols. Motivated by modern large-scale settings, we are interested in *scalable* protocols for these tasks, where each (honest) party communicates a number of bits which is sub-linear in n, the number of parties. The state of the art protocols require each party to send $\tilde{O}(\sqrt{n})$ bits[1] throughout $\tilde{O}(1)$ rounds. Despite significant efforts, getting protocols with $o(\sqrt{n})$ communication per party has been a major challenge for several decades.

We propose a new framework for designing efficient agreement protocols. Specifically, we design $\tilde{O}(1)$-round protocols for all of the above tasks (assuming constant $< 1/3$ fraction of static corruptions) with the following guarantees:

- **Optimistic complexity**: In an honest execution, (honest) parties send only $\tilde{O}(1)$ bits.
- **Pessimistic complexity**: In any other case, (honest) parties send $\tilde{O}(\sqrt{n})$ bits.

Thus, all an adversary can gain from deviating from the honest execution is that honest parties will need to work harder (i.e., transmit more bits) to reach agreement and terminate. We use our new framework to get a scalable MPC protocol with optimistic and pessimistic complexities.

Technically, we identify a relaxation of Byzantine Agreement (of independent interest) that allows us to fall-back to a pessimistic execution in a coordinated way by all parties. We implement this relaxation with $\tilde{O}(1)$ communication bits per party and within $\tilde{O}(1)$ rounds.

[*]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]We use the notation $\tilde{O}(\cdot), \tilde{\Omega}(\cdot)$ to hide poly-logarithmic factors in n.

Contents

1 Introduction

Byzantine agreement [PSL80, LSP82] is a fundamental and well-studied abstraction used to illustrate the problem of designing reliable distributed systems where many parts of it can become unreliable. In more detail, the task in Byzantine agreement is to allow n parties, each with a private input, to agree on a single common output that is equal to some party's input. The challenge is to guarantee that all honest parties successfully terminate despite the presence of an adversary fully controlling the behaviour of some fraction of parties. There are few other intimately related agreement tasks such as Broadcast, Committee Election, and Leader Election (see Section 3.2 for a description of these abstractions). All of the above are not only interesting on their own right, but they serve as basic building blocks in many distributed protocols; for example, a broadcast channel underlies essentially all secure multi-party computation (MPC) protocols [BGW88, GMW87, CCD87].

Motivated by modern "large-scale" settings, we are interested in designing scalable protocols for all of the above tasks, supporting a very large number of parties. Letting n be the number of parties involved, a protocol is said to be scalable if the communication complexity of each party is sub-linear in n, ideally even poly-logarithmic. Despite decades of work and thousands of papers, we still have no fully compelling solution to any of the above tasks fitting large networks. Specifically, the state of the art protocols require $\tilde{O}(\sqrt{n})$ per-party communication and $\tilde{O}(1)$ rounds [KLST11, GK24].

It is a major open problem in this area to construct agreement protocols with full agreement and $\tilde{O}(1)$ per-party communication. This is true even in the model of synchronous communication and static corruptions that we consider. See Section 1.1 for related work, known barriers, and further details on known protocols.

Optimistic/pessimistic efficiency. In this work, we propose a new approach to make progress on the above major open problem by considering "beyond worst-case" complexity of protocols. Specifically, we consider protocols that could have different complexities, depending on the attacker's actions: If the attacker follows the protocol's specification, then the protocol will terminate with $\tilde{O}(1)$ per-party communication and agreement will be reached. Otherwise, if the adversary deviates from the prescribed protocol, the protocol will still terminate and agreement will be reached, but it will require more work per-party. We emphasize that in either case the protocol terminates and full agreement is reached. Thus, all an adversary can gain from deviating from the honest execution is that honest parties will need to work harder (i.e., perform additional computation and transmit more bits) to reach agreement and terminate. We state our main result next.

Theorem 1.1 (Scalable agreement; Informal)**.** *There are (computationally efficient) $\tilde{O}(1)$-round Byzantine Agreement, Committee Election, and Broadcast protocols tolerating $1/3 - \epsilon$ fraction of static corruptions*[2] *for any $\epsilon > 0$ with the following features:*

> ***Optimistic Complexity:*** *In an honest execution, each party sends $\tilde{O}(1)$ bits.*
>
> ***Pessimistic Complexity:*** *In the worst-case, each honest party sends $\tilde{O}(\sqrt{n})$ bits.*

The above result is useful in application where an attacker gains nothing from delaying the protocol's termination; in such cases, there is no point for it to deviate from the protocol, and therefore we could achieve better efficiency. We further remark that while we stated above that our protocols obtain optimistic efficiency ($\tilde{O}(1)$ communication overall per party) in honest executions, in fact, this is the case even if abort is allowed (i.e., an adversary can abort corrupted nodes).

[2]*Static corruptions* means that the set of corrupted parties is chosen by the adversary after the protocol is specified but before an execution begins.

The above theorem is obtained via instatiating a generic compiler that combines "optimistically fast" with "pessimistically slow" protocols in a non-trivial way. The "optimistically fast" protocol is very simple and is based on running an "almost-everywhere" agreement protocol [KSSV06] and then random polling. The "pessimistically slow" protocol is the one of [GK24]. The main technical contribution is the way we combine both protocols; see discussion below. First, here is a statement of our compiler.

Theorem 1.2 (Scalable agreement compiler; Informal)**.** *Let $X \in \{\mathsf{ByzantineAgreement}, \mathsf{Broadcast}, \mathsf{CommitteeElection}\}$ be a task. Assume that there is a protocol Π for X tolerating $1/3 - \epsilon$ fraction of static corruptions for any $\epsilon > 0$. Then, there is another protocol Π' for X (tolerating $1/3 - \epsilon$ fraction of static corruptions for any $\epsilon > 0$) with the following features:*

> ***Optimistic Complexity****: In an honest execution, each party sends $\tilde{O}(1)$ bits and the protocol terminates within $\tilde{O}(1)$ rounds.*
>
> ***Pessimistic Complexity****: In the worst-case, the communication and round complexities of Π' is the same as that of Π plus $\tilde{O}(1)$.*

Technical highlight: agreement for failures. The main technical building block of the above theorem is a method for composing an "optimistically fast" protocol with a "pessimistically slow" one. To this end, we introduce and construct a relaxed variant of an agreement functionality, called *agreement for failures*, which we explain next. Recall that in an agreement functionality the goal is to guarantee that all parties terminate and agree on some specified value. Agreement for failures relaxes the above by requiring that (1) in an honest execution, if all parties hold 0 (for "success"), it must be the output, and (2) in any execution if an honest party holds 1 (for "failure"), it must be the output. With this abstraction, we can identify failure in our "optimistically fast" protocol and then fallback and execute the "pessimistically slow" protocol. We manage to implement an agreement for failures protocol with essentially optimal complexity: $\tilde{O}(1)$ rounds and $\tilde{O}(1)$ per-party communication.

As mentioned, we use our agreement for failures protocol to obtain a generic framework for composing efficient optimistic protocols with less efficient pessimistic protocols. Specifically, we manage to combine an efficient protocol Π_{light} (solving a given task X) where it is guaranteed that a failure was noticed by at least one party with a less efficient protocol Π_{heavy} (solving the same task X) that is executed only if everyone knows that some failure occurred. The complexity of the combined protocol is inherited from Π_{light} in an honest execution and from Π_{heavy} in any other case. We refer to Section 6 for full details.

Application to Scalable MPC. 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. 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 complexity of each of the n parties is $s \cdot \mathsf{poly}(n)$ when computing a function represented as a circuit of size s.[3] The question of scalable MPC, i.e., protocols where the dominant term in the complexity

[3]Interestingly, in BGW it was already observed that their protocol has an optimistic/pessimistic flavor where in the former the polynomial in n is slightly better than in the pessimistic case.

is just the circuit size, is still an active topic and modern results achieve MPC protocols with strong security guarantees and communication complexity $\tilde{O}(s + \mathsf{poly}(n))$ (see Section 1.1 for an overview and references).

We use our scalable agreement protocols to obtain a new generic scalable MPC. The properties of our new protocol are summarized in the next theorem. Note that for this result we also assume private channels.

Theorem 1.3 (Scalable MPC; Informal)**.** *There is a statistically maliciously secure MPC protocol tolerating $1/3 - \epsilon$ fraction of static corruptions for any $\epsilon > 0$ with the following features. Given a circuit of size s and depth d over n inputs, the protocol has the following complexity:*

> ***Optimistic Complexity****: In an honest execution, each party sends $\tilde{O}(s/n)$ bits.*
>
> ***Pessimistic Complexity****: In the worst-case, each party sends $\tilde{O}(s/n + \sqrt{n})$ bits.*
>
> ***Round Complexity****: All parties terminate after $\tilde{O}(d)$ rounds.*

The additive $\tilde{O}(\sqrt{n})$ term in the pessimistic complexity bullet comes generically from Theorem 1.2. 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). We emphasize that our protocol has the appealing feature that in an honest execution the total communication complexity is essentially equal to the circuit size, for any circuit, and it is split in a balanced manner across parties. This idea largely appeared in the scalable MPC protocol of [DKM$^+$17], where the main difference is that we obtain optimistic/pessimistic complexity, while they only had pessimistic complexity, and this is due to the use of Theorem 1.2 instead of the protocol of [KLST11].

1.1 Related Work

Scalable agreement. The BA problem was introduced in the landmark work of Lamport, Shostak, and Pease [LSP82]. In the following couple of decades, several protocols were presented (e.g., [DS83, DLS88, CL99]) but they all had quadratic total overhead, that is, every party had to essentially communicate with every other party. A protocol of [KSSV06] was the first to break the quadratic barrier, but it had the caveat of *almost-everywhere* agreement; that is, they only guarantee that a $1 - o(1)$ fraction of parties agree on the output. Extending their almost-everywhere agreement to full agreement in a scalable manner and with minimal cost is still an exciting challenge. We mention some of the key papers addressing this challenge, noting that almost-everywhere agreement has a long history of research, and it was originally proposed to obtain best-possible guarantees in sparse networks [DPPU88].

First, [KS09] presented a protocol that satisfies full agreement but it is not balanced. That is, while most parties do communicate a sublinear amount of bits in n overall, there are few parties that communicate essentially with everyone. Several follow up works (e.g., [BGH13, ACD$^+$19]) suffer from the same issue. Then, [KLST11] presented a protocol that satisfies full agreement and it is balanced, but this comes with an extra $\tilde{O}(\sqrt{n})$ term in the communication cost. A recent work [GK24] gave a computationally efficient variant of the protocol of [KLST11].

The extra cost in efficiency is partially explained by an $\Omega(\sqrt[3]{n})$ lower bound on the communication complexity of at least one party in any BA protocol with full agreement, due to [HKK08]. This lower bound, however, applies only to protocols with *static filtering.* In static filtering, 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). The recent work of [GK24] gave a related $\tilde{\Omega}(\sqrt{n})$ lower bound on the sum of communication and computational complexities of at least one party in any Byzantine agreement protocol with static filtering and with full agreement.

Lastly, we mention works of [BCG21, FGK24]. They assume cryptographic and trusted setup assumptions as well as a polynomial-time adversary. Also, they assume dynamic filtering; namely, the decision of which message is received can be based on the content of received messages (in their case, every message is checked if it contains a valid digital signature). With these relaxations of the model, no lower bound is known. They showed a communication-optimal protocol: only $\tilde{O}(1)$ bits of communication per party are needed to reach full agreement.

We remark that all of the above works, as well as ours, assume near-optimal resilience, i.e., up to $(1/3-\epsilon)$ fraction of corruptions (i.e., near-optimal resilience range). Less than $n/3-1$ corruptions is strictly necessary due to lower bounds of [LSP82, FLM86, Bor96] (unless further assumptions are made such as a trusted setup or a computationally bounded adversary).

All of the above discussion considers static corruptions. We mention that a work of [KS10] considers adaptive adversaries and they show a scalable protocol in the hidden-channel model. Committee-based techniques do not seem to be applicable (because the adversary can corrupt the committee members adaptively) and so does our framework.

Scalable MPC. 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 $O(s+\mathsf{poly}(n))$ for securely computing a size s circuit by n parties. This was studied in the context of perfect or statistical security and optimal resilience (up to $n/3-1$ or $n/2-1$ corrupted parties) [HM01, DN07, BH08, GIP$^+$14, IKP$^+$16, CCXY18, CGH$^+$18], or with perfect or statistical security and near-optimal resilience (up to $(1/3-\epsilon)$ or $(1/2-\epsilon)$ fraction of corrupted parties) [DI06, IPS09, DIK$^+$08, DIK10]. In all of these works a broadcast channel is assumed but its usage is limited to a number of times which is independent of the circuit size. All of these works obtain somewhat stronger notions of security than what we obtain in Theorem 1.3 (e.g., they often tolerate adaptive corruptions while we handle only static ones). Note that we can use our broadcast protocol from Theorem 1.2 to instantiate the broadcast channel in the above works, achieving statistical security for $(1/3-\epsilon)$ fraction of static corruptions.

Most related to us are the works [DKMS12, DKM$^+$17]. In these works the authors used the full agreement protocol of [KLST11] to get a scalable MPC with complexity $\tilde{O}(s/n+\sqrt{n})$ to compute a size s circuit by n parties, per party. The corruption model is $(1/3-\epsilon)$ fraction static corruptions, same as ours. Our MPC prootcol is very similar to theirs (associating the wires of the circuit to quorum members); but, our description is somewhat simpler because we use generic maliciously secure MPC as black-box whereas they sometime use internal building blocks such as verifiable secret sharing. The optimistic/pessimistic aspect is new to our work. Lastly, we mention the work of [DKMS14] who studied scalable MPC protocols in an asynchronous setting.

2 Technical Overview

In this section we give a high level overview of our protocols. We start by explaining the ideas underlying Theorems 1.1 and 1.2, i.e., the agreement protocols with optimistic/pessimistic efficiency features. We then continue to explain our framework for composing general optimistic and pessimistic protocols, and finally, we explain our application for secure multiparty computation, Theorem 1.3.

2.1 Optimistic/Pessimistic Agreement

At a very high level, the challenge of composing an optimistic protocol with a pessimistic one is that once fall-back to the pessimistic protocol is needed, all parties need to be aware of this and proceed in a coordinated fashion. If the optimistic protocol has a "global failure detection" property, guaranteeing that all honest parties know that the optimistic protocol failed, then we can directly execute the pessimistic protocol, if needed.

Our main observation is that a "local failure detection" property is sufficient. In local failure detection, we require (only) that there exists an honest party that notices that a failure occurred. However, given a "local failure detection" property getting a "global failure detection" property is by itself already some form of an agreement, which is exactly the problem we are trying to solve. Our main observation is that a "local to global failure detection" protocol does not have to reach agreement(!), but rather it only has to satisfy a "global failure detection" property, i.e., it must have the property that if agreement is not reached, then all honest parties know about it. This observation allows up to implement an efficient "local to global failure detection" protocol. Next, we explain the above in more detail and show how to achieve this property in the context of scalable agreement protocols.

Local to global failure detection. Recall that in an agreement protocol, abstractly, the goal is for all honest parties to terminate and agree on some value. Specifically, we will consider the task of agreeing on a long enough string with high enough min-entropy[4]—this will be sufficient to obtain all of our applications, including Byzantine Agreement, Broadcast, and Committee Election.

Our starting point is the almost-everywhere agreement protocol of [KSSV06]. In their protocol, the above task is met by almost all parties. That is, after poly-logarithmically many rounds of communication, a poly-logarithmically long random string str with poly-logarithmic min-entropy is known to $1 - o(1)$ fraction of parties. The goal is to "boost" the knowledge of this string to all other honest parties as well. At this point, a naive attempt would be to let every party poll sufficiently many random parties and ask for their str value. Then, each party will decide on a string that it obtained from more than $1/2$ of the polled parties. By a simple application of Chernoff's bound, it suffices to poll independently poly-logarithmically many random parties.

This solution, however, is flawed because an adversary can easily prevent an honest party from learning the right value. Specifically, assume that honest party P *that does not know* str polled parties $P_1, \ldots, P_\ell$. The adversary can *flood* all parties in $P_1, \ldots, P_\ell$ and thereby prevent them from replying to P.[5] In such case, P will not learn str. At this point, previous works [KLST11, GK24] suggested various techniques limiting the adversary's ability to perform such flooding attacks, but this came with a significant cost in efficiency, requiring $\tilde{\Theta}(\sqrt{n})$ communication.

We deviate from all of these previous works by making the following critical observation: While the flooding attack does prevent P from learning str, it does provide P (and some of its poll list members) with important knowledge: *they were attacked!* At this point we would have been happy to merely fall-back to one of the known agreement protocols (such as [GK24] itself) and pay the extra cost in efficiency. But, we are in a state of inconsistency: some of the parties know str and think they are done while others do not know str and want to fall back to a heavy protocol.

[4]The min-entropy of a random variable is the negative logarithm of the probability of the most likely outcome. We say that the min-entropy is high enough if the probability of the most likely outcome is $\mathsf{negl}(n)$.

[5]Flooding attacks (or "denial of service") are a threat because we put a constraint on the honest parties' communication complexity. Specifically, the adversary (controlling a constant fraction of parties) can send a poll request (in the name of each controlled party) to every honest party. Since the honest nodes need to reply to all of these poll requests, then (if per-party communication is limited to $o(n)$ bits) there is no budget to reply to any honest poll requests.

It may seem, at first sight, that we have not gained much because we are again in a situation where some of the parties are not aware of the right state of the protocol and so we many need to perform some heavy agreement protocol. But, a closer look reveals that we have actually made a lot of progress:

1. A party that did not know str after the almost-everywhere agreement now knows str or knows the polling phase failed.
2. If the polling phase failed for at least one party, there must exist at least one party that did know str *and* learned that the protocol failed.[6]

Our goal now is to propagate the knowledge of failure to everyone so that everyone can consistently fall back to a heavy agreement protocol. This task is much simpler than agreeing on str to begin with because (a) all the parties that know that the protocol failed already know what to output and so we do not need to deal with them, and (b) all other parties necessarily agree on str. Thus, we remain with the following simpler task: parties need to perform agreement given a shared sufficiently long high min-entropy string str. The solution to this problem is quite standard and we explain it next.

Semi-random string to a quorum. A "good" quorum is a collection of well-balanced "good" committees, one per party. A "good" committee is a collection of $c = \mathsf{polylog}(n)$ parties where at least $2/3$ of them are honest. A good quorum consists of n such good committees, one per party, and where every single party participates in $\mathsf{polylog}(n)$ committees. By a Chernoff bound, a uniformly random collection of n small subsets of parties will be a good quorum, but we do not have a uniformly random shared string. Here, we use a lemma of [GK24] saying that one can turn a semi-random string to a good quorum. Note that, since every committee contains an honest majority, the committees can be viewed as honestly-behaving parties so we need not worry about Byzantine behavior.

Given a quorum, it is straight-forward how to make sure that if one of the parties has a Fail input, then all parties will know it. The party that holds Fail will broadcast (using an arbitrary polynomial-time protocol, e.g., [DS83]) it to its committee and then the committees will propagate this in a tree-like fashion and inform all other committees of this. Finally, each committee will send the Fail symbol to its party.

Applications. Once we obtain a quorum, we can get many applications in a straight-forward way, similarly to how we disseminated the Fail symbol. For example, performing broadcast can be done by the following simple protocol: the broadcaster sends its message to its committee and then the committee broadcasts this value in a tree-like fashion among all other committees; the latter send this value to their respective parties. (By saying that a committee sends or receives a value, we mean that more than $2/3$ of the parties in the committee send or receive it.) Similar protocols can be described for Byzantine Agreement and Committee Election; see Section 7 for details.

2.2 A Best-Possible Optimistic/Pessimistic Framework

We generalize the above idea and present a general-purpose framework for building protocols with an optimistic/pessimistic complexity. Specifically, consider a protocol Π_{light} with the following

[6]The claim is obvious for parties that know str and detect a failure (e.g., because their poll fails or because they are flooded). Otherwise, if the poll of a party that knows str fails, then (w.h.p) at least one honest party that knows str does not respond to its poll request (recall that we are guaranteed that most parties know str). The latter honest party must have been flooded! So, we are back to the case that an honest party knows str and detects a failure.

properties:

- In an honest execution, it achieves some goal.
- In any other case, at least one party knows that something went wrong (but privacy, if considered, was not broken).

Additionally, we have a protocol Π_{heavy} that achieves the same goal, no matter what the adversary does.

We would like to combine these two protocol to a single protocol where we first run Π_{light} and then, if something goes wrong, run Π_{heavy}. Unfortunately, this is tricky since the state of all parties at the end of Π_{light} may not be consistent: not all parties know that something went wrong. Thus, we use the idea from above (the optimistic/pessimistic agreement) to propagate the knowledge about the occurrence of a failure. Our combined protocol has the following structure:

1. Run Π_{light}. Each party has an output y_i and a symbol $o_i \in \{\mathsf{Fail}, \mathsf{Success}\}$.
2. Generate a quorum, as explained in Section 2.1. If this phase fails, each party that detects failure sets its o_i to Fail.
3. In a tree-like fashion, via the quorum committees, compute o which is an indicator for whether there exists a party P_i with $o_i = \mathsf{Fail}$. Note that, only parties that know the right quorum description will learn the right value of o.
4. For each party P_i, if $o_i = \mathsf{Fail}$ or $o = \mathsf{Fail}$, participate in Π_{heavy}; Otherwise, output y_i.

Observe that in an honest execution, Π_{heavy} will never be invoked and so the complexity of the protocol is similar to the complexity of Π_{light} plus $\tilde{O}(1)$ (the complexity of ByzantineAgreementForFailures). If a failure is detected, either during Π_{light} or during ByzantineAgreementForFailures, Π_{heavy} is invoked and its complexity dominates. Refer to Section 6 for the precise statement and full details.

2.3 Secure MPC

The idea to obtain an MPC protocol is simple given the quorum structure we obtained above. Following [DKM+17], once we have a good quorum, we associate each wire to some committee, in a balanced way (i.e., if there are s wires, each committee is in charge of $\approx s/n$ wires). We compute the circuit gate by gate, basically by relying on some known perfectly secure MPC to compute each gate and provide input for the next gate. Notice that no committee member holds the wire, but instead they have it in a secret shared fashion, and so when we process a gate, we first reconstruct the input wires, and at the end share the outputs to the appropriate committee members. All of these details are taken care of by a generic MPC, such as the one of BGW [BGW88]. Since each committee has a strong majority of honest parties and since each committee consists of poly-logarithmically many parties, this MPC costs $\tilde{O}(1)$ communication and rounds.

3 The Model and Problem Definitions

3.1 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 $1, \dots, n$. 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.

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

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

Note that the space and computational complexity of a party are only computed as a function of the messages that were not discarded during the filtering phase.

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].

3.2 Problem Definitions

Byzantine agreement. In this problem, there are n parties, $P_1, \dots, P_n$. 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 in 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 (since the fraction of honest parties in the whole input is constant). 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.

4 Preliminaries

A function $\mathsf{negl}\colon \mathbb{N} \to \mathbb{R}^+$ is *negligible* if for every constant $c > 0$ there exists an integer N_c such that $\mathsf{negl}(n) < n^{-c}$ for all $n > N_c$. The *statistical distance* between two random variables X and Y over a finite domain Ω is defined by $\mathsf{SD}(X,Y) \triangleq \frac{1}{2} \cdot \sum_{x\in\Omega} |\Pr[X = x] - \Pr[Y = x]|$. Two sequences of random variables $X = \{X_n\}_{n\in\mathbb{N}}$ and $Y = \{Y_n\}_{n\in\mathbb{N}}$ are *statistically close* (or statistically indistinguishable) if there exists a negligible function $\mathsf{negl}(\cdot)$ for which $\mathsf{SD}(X_n, Y_n) \leq \mathsf{negl}(n)$ for all $n \in \mathbb{N}$. For an integer $n \in \mathbb{N}$ we denote by $[n]$ the set $\{1, \dots, n\}$.

4.1 Secure Computation

We consider execution of distributed protocols in adversarial environments. All of our protocols assume n parties denoted $P_1, \dots, P_n$.

Definition 4.1 (View)**.** *Given a protocol Π, an input $\vec{x}$, and party P_i, we let $\mathsf{view}_i^{\Pi}(\vec{x})$ be a random variable corresponding to the information seen by P_i throughout an execution of Π. This consists of $(x_i, r_i, m_{i_1}, \dots, m_{i_k})$, where x_i is P_i's private input, r_i is its internal coin tosses, and m_{i_j} is the jth message that was received by P_i in the protocol execution. For a subset of parties B, we denote $\mathsf{view}_B^{\Pi}(\vec{x}) = \cup_{i\in B}\mathsf{view}_i^{\Pi}(\vec{x})$.*

Attacker classes. A class of adversaries that is often studied is adversaries that can *abort*. Such adversaries can deviate from the protocol specification only in a limited way, as follows. At any point during the execution of the protocol, the adversary can cause a corrupted party to abort. After a party aborts, it cannot perform any computation nor send or receive messages.

Definition 4.2 (Aborting adversary)**.** *An adversary is an* aborting adversary[7] *if every message that is sent by a corrupt party is computed by following the protocol's specification (as if it were semi-honest), except that the adversary can dynamically cause a corrupted party to abort. Let $\mathscr{A}_{Abort}$ denote this class of adversaries.*

Next, we consider malicious adversaries that can follow an arbitrary strategy in order to carry out their attack.

Definition 4.3 (Malicious adversary)**.** *An adversary is* malicious *if it follows an arbitrary strategy, arbitrarily deviating from the protocol. Let $\mathscr{A}_{All}$ denote this class of adversaries.*

It follows immediately from the definitions above that

$$\mathscr{A}_{Abort} \subseteq \mathscr{A}_{All}.$$

[7]Also known as an adversary that can cause crash failures.

We shall only consider $\mathsf{static\ corruptions}$ in this work. Namely, the set of corrupted parties will be chosen by the adversary after the protocol is specified but before an execution begins. Also, the adversary will be allowed to control some constant fraction of the parties, typically, $1/3 - \epsilon$ for any $\epsilon > 0$. Thus, we introduce the notation $\mathscr{A}_\alpha^{static}$ for the family of adversaries that statically corrupt up to $\alpha - \epsilon$ fraction of parties for any positive constant ϵ. Lastly, we emphasize that in all of the above definitions, we do not limit the computational power of the adversary in any way.

Security definition. Next, we provide a security definition that will be used throughout this work. We follow the real-ideal paradigm [Can00, Can01, Gol04] that is helpful in modularizing our proofs. Concretely, this method allows us to transition to a hybrid model where some of the building blocks are implemented using ideal functionalities. Since all of our simulators will be simple enough (so called "straight-line") we will be able to automatically apply standard composition theorems.

The real-world execution: In the real-world execution, the protocol Π is carried out among n parties, where some subset $\mathcal{C}$ of parties is controlled by the adversary $\mathcal{A}$. We shall also assume that the execution is performed in the presence of some auxiliary input aux known to $\mathcal{A}$; this is important for composition. Then, we let real-world execution consist of the view of the adversary and the output of the honest parties. That is, for an input $\vec{x} = (x_1, \ldots, x_n)$ and auxiliary information aux, let $\mathsf{real}_{\mathcal{A}}^{\Pi}(\vec{x}, \mathsf{aux}) = \left(\mathsf{view}_{\mathcal{C}}^{\Pi}(\vec{x}), \{y_i\}_{i\in[n]\setminus\mathcal{C}}\right)$, where y_i is P_i's output.

*The ideal-world execution:*The ideal-world execution is augmented with a "trusted party" constituting a general algorithmic entity called an ideal functionality and denoted $\mathcal{F}$. The ideal functionality, which is modeled as another machine, repeatedly receives inputs from the parties and provides them with appropriate output values, while maintaining local state in between. To this end, we define an experiment involving an imaginary adversary $\mathcal{S}$ (often called the simulator) that knows $\mathcal{A}$ and can interact with $\mathcal{F}$. The experiment's output, given input $\vec{x} = (x_1, \ldots, x_n)$ and auxiliary information aux, is denoted $\mathsf{ideal}_{\mathcal{S},\mathcal{A}}^{\mathcal{F}}(\vec{x}, \mathsf{aux})$ and is defined as follows. Note that we consider only static corruptions.

1. **Initialization**: $\mathcal{S}$ chooses $\mathcal{C} \subset [n]$, the corrupted parties. $\mathcal{S}$ receives $\{x_i\}_{i\in\mathcal{C}}$ and aux.
2. **Evaluation**: $\mathcal{F}$ receives $\mathcal{C}$ and $\{x_i\}_{i\in[n]\setminus\mathcal{C}}$ and then interacts with $\mathcal{S}$. At the end of the interaction, $\mathcal{S}$ knows $\{y_i\}_{i\in\mathcal{C}}$ and $\mathcal{F}$ outputs $\{y_i\}_{i\in[n]}$.
3. **Output**: $\mathcal{S}$ computes $\{\mathsf{view}_i\}_{i\in\mathcal{C}}$ (based on everything that it knows) and the output of the experiment is $(\{\mathsf{view}_i\}_{i\in\mathcal{C}}, \{y_i\}_{i\in[n]\setminus\mathcal{C}})$.

The following definition says that a protocol is considered secure with respect to a functionality and a class of adversaries if it is possible to come up with a simulator that causes the above two worlds to be indistinguishable.

Definition 4.4. *We say that a protocol Π realizes functionality $\mathcal{F}$ against adversaries family $\mathscr{A}$, if for every $\mathcal{A} \in \mathscr{A}$ there exists a simulator $\mathcal{S}$ such that for every input $\vec{x}$ and every $\mathsf{aux} \in \{0,1\}^*$, it holds that $\mathsf{real}_{\mathcal{A}}^{\Pi}(\vec{x}, \mathsf{aux})$ is statistically indistinguishable from $\mathsf{ideal}_{\mathcal{S},\mathcal{A}}^{\mathcal{F}}(\vec{x}, \mathsf{aux})$.*

If the statistical distance between $\mathsf{real}_{\mathcal{A}}^{\Pi}(\vec{x}, \mathsf{aux})$ and $\mathsf{ideal}_{\mathcal{S},\mathcal{A}}^{\mathcal{F}}(\vec{x}, \mathsf{aux})$ is zero, we say that the protocol Π is *perfectly* secure.

4.2 (Attacker-Dependent) Complexity Measures of Protocols

Given a protocol Π, consider some abstract complexity measure for Π, denoted $\mathsf{complexity}(\cdot)$. For concreteness, assume $\mathsf{complexity}$ stands for communication complexity, denoted also cc, but it could be anything else like round complexity, denoted rc, or computational complexity. Typically, $\mathsf{complexity}(\cdot)$ of a protocol is defined as the maximal total communication (of honest parties) in Π, over all possible executions, in the presence of an arbitrary adversary. However, we want to be more fine grained.

Definition 4.5 (Complexity)**.** *Given a protocol Π, an adversary $\mathcal{A} \in \mathscr{A}_{All}$, and randomness r, we define $\mathsf{complexity}(\Pi, \mathcal{A}, i, r)$ as the number of bits sent by party i throughout the execution of Π in the presence of $\mathcal{A}$ and randomness r.*

The standard definition of the $\mathsf{complexity}$ of the ith party in Π is then the worst-case over all possible adversaries and randomnesses. Namely,

$$\mathsf{complexity}(\Pi, i) = \max_{r,\mathcal{A}} \{\mathsf{complexity}(\Pi, \mathcal{A}, i, r)\}.$$

For the complexity of the protocol Π, we sum up the complexities of individual parties (worst-case over all possible adversaries and randomnesses).

$$\mathsf{complexity}(\Pi) = \max_{r,\mathcal{A}} \left\{ \sum_{i=1}^{n} \mathsf{complexity}(\Pi, \mathcal{A}, i, r) \right\}.$$

We can now define the notion of balanced protocols.

Definition 4.6 (Balanced protocol)**.** *A protocol Π is said to be* balanced *if*

$$\frac{\mathsf{cc}(\Pi)}{\max_{i\in[n]} \{\mathsf{cc}(\Pi, i)\}} \in \tilde{\Theta}(n).$$

Adversary-dependent complexity. It is useful to adapt the above definitions with respect to a particular sub-class of attackers. For $\mathscr{A} \subseteq \mathscr{A}_{All}$, define

$$\mathsf{complexity}(\Pi, \mathscr{A}, i) = \max_{r,\mathcal{A}\in\mathscr{A}} \{\mathsf{complexity}(\Pi, \mathcal{A}, i, r)\}$$

and analogously,

$$\mathsf{complexity}(\Pi, \mathscr{A}) = \max_{r,\mathcal{A}\in\mathscr{A}} \left\{ \sum_{i=1}^{n} \mathsf{complexity}(\Pi, \mathcal{A}, i, r) \right\}.$$

In words, we restrict the possible set of actions the adversary can do, and then measure how expensive the protocol is if such an adversary is present. Note that if $\mathscr{A} = \mathscr{A}_{All}$, then the above definition coincides with the general (worst-case over all adversaries) definition above. That is, $\mathsf{complexity}(\Pi, \mathscr{A}_{All}, i) = \mathsf{complexity}(\Pi, i)$ for every $i \in [n]$ and $\mathsf{complexity}(\Pi, \mathscr{A}_{All}) = \mathsf{complexity}(\Pi)$.

5 Propagating Failure Detection

In this section, we show how to propagate the knowledge that a failure occurred during the execution of a protocol from a single party to everyone. Of course this task can be easily achieved by using a generic broadcast protocol or some variant of Byzantine Agreement, but this is expensive in terms of communication and we would like to do it more efficiently. First, we explain the task at hand by defining the functionalities we want to achieve formally. Then, we will show how to implement our transformation efficiently.

5.1 Failure Detection Functionalities

Generically, the behavior of a functionality in case some of the adversarial parties deviate from the protocol's specification in an evident way (say by completely aborting), depends on the exact specification of the functionality. For us, we need to be more precise and we do so next.

We consider attackers that cheat and some honest party notices that something went wrong, and how it affects the outcome of the protocol. Specifically, we define two variants (or wrappers) of a given functionality $\mathcal{F}$. The first variant, denoted $\mathcal{F}^{gFD}$ (where gFD stands for *global failure detection*), says that if some party noticed that something went wrong during an execution, then every honest party should know about this. The second variant, denoted $\mathcal{F}^{lFD}$ (where lFD stands for *local failure detection*), says that at least one honest party should end up being aware that the protocol did not execute honestly. In this case, we have no guarantee or requirement about the output of all other parties.

It is worth pointing out that a priori it seems much easier to satisfy local failure detection. This is true because going from local failure detection to global failure detection requires some form of broadcast or agreement which are very close to the problem we are to solve in the first place.

We formalize both variants ($\mathcal{F}^{gFD}$ first and then $\mathcal{F}^{lFD}$) as wrappers of a given functionality $\mathcal{F}$. In the above, the simulator $\mathcal{S}$ has to identify which parties notice that something went wrong during the execution and inform the functionality. The latter will either set the output of all honest parties to Fail or just a single party that notices, respectively.

Functionality 1: $\mathcal{F}^{gFD}$

1. $\mathcal{S}$ and $\mathcal{F}^{gFD}$ interact according to $\mathcal{F}$. At the end, $\mathcal{F}^{gFD}$ knows $\vec{y} = (y_1, \ldots, y_n)$.
2. $\mathcal{S}$ submits Fail or $\mathsf{Success}$.
3. If $\mathcal{S}$ submitted $\mathsf{Success}$, then output $\vec{y}$.
4. If $\mathcal{S}$ submitted Fail, output $(\mathsf{Fail}, \ldots, \mathsf{Fail})$.

Functionality 2: $\mathcal{F}^{lFD}$

1. $\mathcal{S}$ and $\mathcal{F}^{lFD}$ interact according to $\mathcal{F}$. At the end, $\mathcal{F}^{lFD}$ knows $\vec{y} = (y_1, \ldots, y_n)$.
2. $\mathcal{S}$ submits Fail or $\mathsf{Success}$.
3. If $\mathcal{S}$ submitted $\mathsf{Success}$, then output $\vec{y}$.
4. If $\mathcal{S}$ submitted Fail, then $\mathcal{S}$ specifies a vector $(y'_1, \ldots, y'_n)$, where at least one y'_i corresponding to an honest party is equal to Fail. Output $(y'_1, \ldots, y'_n)$.

Relation between $\mathcal{F}$, $\mathcal{F}^{gFD}$, and $\mathcal{F}^{lFD}$. A simulator $\mathcal{S}$ for $\mathcal{F}$ directly implies a simulator $\mathcal{S}^{gFD}$ for $\mathcal{F}^{gFD}$. Indeed, notice that $\mathcal{S}^{gFD}$ can merely submit $\mathsf{Success}$ in step 2 and output the output of the interaction between $\mathcal{S}$ and $\mathcal{F}$. By similar logic, a simulator $\mathcal{S}^{gFD}$ for $\mathcal{F}^{gFD}$ directly implies a simulator $\mathcal{S}^{lFD}$ for $\mathcal{F}^{lFD}$. In other words, it is easiest to realize the functionality $\mathcal{F}^{lFD}$, harder to realize the functionality $\mathcal{F}^{gFD}$, and hardest to realize the functionality $\mathcal{F}$.

5.2 Propagating Knowledge of Failures

We show a method for propagating knowledge of failures from a single party to all of them. Specifically, we translate protocols that realize *local* failure detection to ones that realize global failure detection. Recall that it is only interesting to consider the above task if we insist on preserving some other non-trivial property of the protocol. Indeed, if we only care about realizing $\mathcal{F}^{gFD}$ for some functionality $\mathcal{F}$, then it is trivial: all parties will just output Fail. So, naturally, we will require the propagation procedure to preserve the fact that the protocol realizes $\mathcal{F}$. Details follow.

We start off with a protocol Π that realizes a functionality $\mathcal{F}$ against a class of attackers $\mathscr{A}$. Further Π realizes $\mathcal{F}^{lFD}$ against $\mathscr{A}^*$, where $\mathscr{A} \subset \mathscr{A}^*$. Thus, it is immediate that Π realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}$. Our next theorem shows that it is possible to transform Π into Π' that realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}^*$ (which contains $\mathscr{A}$ and therefore this task is non-trivial); at the same time, we preserve the fact that Π' realizes $\mathcal{F}$ against $\mathscr{A}$. The transformation costs only an additive poly-logarithmic term in n overhead in rounds and communication complexity (per party).

Theorem 5.1. *Consider a protocol Π and a functionality $\mathcal{F}$. Further, consider two attacker classes $\mathscr{A}^* \subseteq \mathscr{A}_{1/3}^{static}$ and $\mathscr{A} \subseteq \mathscr{A}_{Abort} \cap \mathscr{A}^*$. Assume that Π realizes $\mathcal{F}$ against $\mathscr{A}$ and realizes $\mathcal{F}^{lFD}$ against $\mathscr{A}^*$.*

Then, there is a protocol Π' that realizes $\mathcal{F}$ against $\mathscr{A}$ and realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}^*$. Furthermore, $\mathsf{rc}(\Pi') = \mathsf{rc}(\Pi) + \mathsf{polylog}(n)$, and for every party i, $\mathsf{cc}(\Pi', i) = \mathsf{cc}(\Pi, i) + \mathsf{polylog}(n)$.

At a very high level, we obtain Theorem 5.1 by designing a new type of Byzantine Agreement abstraction. This abstraction, referred to as ByzantineAgreementForFailures, still requires agreement among all honest parties, like standard BA. The main difference is that we require that if some honest party has the input Fail, then all honest parties will agree on Fail. If all parties hold $\mathsf{Success}$ and the adversary behaves somewhat honestly, then everyone will agree on $\mathsf{Success}$. Using the ByzantineAgreementForFailures protocol, we can first run a protocol with local failure detection (as guaranteed in Theorem 5.1), and then "boost" the knowledge of failure of any party. In the resulting protocol, either everyone gets the needed output or everyone agrees that something went wrong.

5.3 Byzantine Agreement for Failures

First, we introduce a strong variant of Byzantine Agreement that we refer to as *Biased Byzantine Agreement*, since this variant is stronger, it must satisfy the *Agreement, Validity, and Termination* properties, but in addition there is a *Bias* property, meaning that, if there exists an honest party with input bit 1, then it must be the output.

In this section we implement a protocol that solves the "Biased Byzantine Agreement" problem and is secure against adversary $\mathcal{A} \in \mathscr{A}_{Abort}$, where for any other adversary it achieves the global failure detection property (i.e., if the protocol fails, all honest parties detect it and output the special symbol Fail). Associating Fail with 1 and $\mathsf{Success}$ with 0, our protocol (referred to as ByzantineAgreementForFailures) achieves the following properties:

- **Agreement:** For all honest parties the output is same.
- **Termination:** After fixed time (in n), each party must hold an output.
- **Failure Detection:** If an honest party P_i has $x_i = \mathsf{Fail}$, then the agreed value is Fail.
- **Fairness:** For all adversaries in $\mathscr{A}_{Abort}$ (see Definition 4.2), if all parties' inputs are $\mathsf{Success}$ (includes corrupted parties), then all parties output $\mathsf{Success}$.

We do not require or provide any guarantee as to what the agreed value should be if (say) the adversary is outside $\mathscr{A}_{Abort}$ and all the inputs (of honest parties) are $\mathsf{Success}$. This relaxation allows us to implement our protocol much more efficiently than full BA.

The protocol. Our protocol relies on the existence of a good quorum. A committee is said to be good if it contains $\mathsf{polylog}(n)$ many parties and more than 2/3 of them are good. A good quorum is a set of n good committees where additionally each party appears in roughly the same number of committees as any other party. In the next theorem we argue that there are efficient ways to come up with a quorum. Put another way, we use a pseudorandom object that takes a sufficiently random short string as input, and efficiently outputs many sufficiently-random good committees. Such a construction is given in [GK24, Theorem 4].

Theorem 5.2 (Quorum generation [GK24])**.** *For every integer $n \in \mathbb{N}$, $c = c(n) \in \Omega(\log^2 n)$, there exist a $d = O(c \cdot \log n)$ and an efficiently computable mapping $H\colon [n] \times [n^c] \to [n]^d$ such that, 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 $1 - \mathsf{negl}(n)$ fraction of the x's from $[n^c]$, it holds that*

1. *For every $i \in [n]$, $H(i, x)$ contains at most $|B|/n + o(1)$ fraction of parties from B.*
2. *Define $H^{-1}(j, x) = \{i \mid j \in H(i, x)\}$. Then, for every $j \in [n]$, $|H^{-1}(j, x)| \in O(d)$. H^{-1} is also efficiently computable.*

The above theorem allows us, at a high level, to turn a sufficiently high (min-) entropy source to a good quorum (with high probability). We now proceed with the description of the protocol. Below, we associate Fail with 1 and $\mathsf{Success}$ with 0 so that Boolean operations (such as logical OR, denoted $\vee$) are well defined.

The ByzantineAgreementForFailures Protocol

- **Setup Phase:** Each party P_i set $y_i \leftarrow x_i$.
- **Almost-Everywhere Agreement Phase:** All parties run together the almost-everywhere Byzantine agreement protocol of [KSSV06].[8] At the end, each party P_i holds a string str_i of size $\Theta(\log^3 n)$.
- **Agreement Phase:**
  1. Each party P_i samples a set of $\log^2 n$ parties (denoted S_i), and sends them "*ping*".
     Filtering: Each party accepts at most $2 \cdot \log^2 n$ "*ping*"s.
  2. For each party P_j: if it received fewer than $2 \cdot \log^2 n$ "*ping*"s, P_j answers with str_j; else, P_j sets $y_j \leftarrow \mathsf{Fail}$.
     Filtering: P_i accepts messages with size at most $\log^4 n$, only from parties in S_i.
  3. For each party P_i: if there is a string str^* that was received more than $|S_i|/2$ times, P_i sets $\mathsf{str}_i \leftarrow \mathsf{str}^*$; else, P_i sets $y_i \leftarrow \mathsf{Fail}$.
- **Distribution Phase:** This phase uses the quorum generation functions H, H^{-1} from Theorem 5.2.

[8]The original almost-everywhere protocol provided by King et al. [KSSV06] is described as a leader election protocol. However, a simple modification gives an almost-everywhere agreement protocol where the output of all but $o(1)$-fraction of parties is a poly-logarithmically long string with poly-logarithmic min-entropy.

1. For each party P_i, if $y_i = \mathsf{Fail}$, P_i sends "*ping*" to $H(i, \mathsf{str}_i)$.
   Filtering: P_j accepts messages of $|$"*ping*"$|$ bit size, only from parties in $H^{-1}(j, \mathsf{str}_j)$.
2. For each committee Q_i, if "*ping*" is received, then each party in the committee sets $y_j \leftarrow \mathsf{Fail}$, else $y_j \leftarrow \mathsf{Success}$.
3. The committees compute the logical OR of the y_i values: all committees communicate their values upstream in a (depth $O(\log n)$) tree-like manner, calculating a logical OR along the way, and then broadcast the results in reverse order. (The exact pseudocode is given in Appendix A as the DISTRIBUTE protocol.)
4. Each party P_j sends x_i to each party in $H^{-1}(j, \mathsf{str}_j)$.
   Filtering: P_i accepts messages of 1 bit size, only from parties in $H(i, \mathsf{str}_i)$.

- **Output Phase:** If there is x^* that was received more than $|H(i, \mathsf{str}_i)|/2$ times, then P_i sets $x \leftarrow x^*$; else, P_i sets $x \leftarrow \mathsf{Success}$. After that, P_i outputs $y_i \vee x$.

Lemma 5.3. *Consider an adversary $\mathcal{A} \in \mathscr{A}_{1/3}^{static}$ corrupting a set of parties $\mathcal{C}$. At the end of the* BYZANTINEAGREEMENTFORFAILURES *protocol, with probability $1 - \mathsf{negl}(n)$, it holds that:*

1. ***Agreement:*** *All honest parties output the same value.*
2. ***Failure Detection:*** *If $\bigvee_{j \in [n] \setminus \mathcal{C}} x_j = \mathsf{Fail}$, then the output is Fail.*
3. ***Fairness:*** *If $\mathcal{A} \in \mathscr{A}_{Abort}$ and $\bigwedge_{j \in [n]} x_j = \mathsf{Success}$, then the output is $\mathsf{Success}$.*

Efficiency. We analyze the round and communication complexity of each party in the above protocol and argue that both of them are bounded by some $\mathsf{polylog}(n)$. First, the almost-everywhere agreement phase requires $\mathsf{polylog}(n)$ rounds and each party sends and receives $\mathsf{polylog}(n)$ bits. Now, we analyze the remaining phases separately. The agreement phase consists of two communication rounds. In the first, each party sends $O(\log^2 n)$ bits and receives at most $O(\log^2 n)$ bits. In the second, each party sends $O(\log^6 n)$ bits and receives at most $\log^6 n$ bits. The distribution phase consists of 3 communication phases. The first requires each party to send $O(\log^5 n)$ bits and receive at most $O(\log^5 n)$ bits. The DISTRIBUTE sub-protocol (see Appendix A) requires $O(\log n)$ rounds, and each party sends and receives $O(\log^{10} n)$ bits within each round. In the last round, each party sends $O(\log^5 n)$ bits and receives at most $\log^5 n$ bits. Overall, the total number of rounds is $\mathsf{polylog}(n)$ and the total communication is also $\mathsf{polylog}(n)$, per party, as needed.

Correctness. Next, we prove that the protocol satisfies all the claimed properties (Lemma 5.3). First, we prove that after the agreement phase every honest party gained some knowledge in the following sense: either it knows the "correct" string that defines the quorum (denoted str), or it knows that something went wrong.

Claim 5.4. *Assume that for at least $1 - o(1)$ fraction of honest parties $\mathsf{str}_i = \mathsf{str}$. For every adversary $\mathcal{A} \in \mathscr{A}_{1/3}^{static}$, at the end of the Agreement Phase, with probability $1 - \mathsf{negl}(n)$, for all honest parties $\{P_i\}_{i \in [n] \setminus \mathcal{C}}$, either $\mathsf{str}_i = \mathsf{str}$ or $y_i = \mathsf{Fail}$.*

Proof. By Step 3 of the protocol, the conclusion could be wrong only if more than half of the parties in some S_i hold $\mathsf{str}^* \neq \mathsf{str}$. Recall that each S_i is chosen uniformly at random. Let $\mu = (2/3 + \epsilon) \cdot \log^2 n$, and let $\delta = 1/\sqrt{(2/3 + \epsilon) \cdot \log\log n}$. Thus, for every $i \in [n] \setminus \mathcal{C}$, by a Chernoff bound, with probability $1 - e^{-\delta^2 \mu/2} = 1 - e^{-\log^2 n/2 \log\log n} = 1 - \mathsf{negl}(n)$, the fraction of honest parties in S_i is at least $2/3+\epsilon-o(1)$. So, by the assumption, at least $(1-o(1))\cdot(2/3+\epsilon-o(1)) > 1/2$ fraction of honest parties in S_i holds str. By a union bound, the above holds for all i's simultaneously with probability $1-\mathsf{negl}(n)$. $\square$

Next, we prove that if there is an honest party for which $\mathsf{str}_i = \mathsf{str}$ but $y_i = \mathsf{Fail}$, then all honest parties that know str will necessarily also output Fail.

Claim 5.5. *Assume that for at least $1-o(1)$ fraction of honest parties $\mathsf{str}_i = \mathsf{str}$ and that the min-entropy of str is $\Omega(\log^2 n)$. For every adversary $\mathcal{A} \in \mathscr{A}^{static}_{1/3}$, if at the end of the Agreement Phase there is an honest party P_i with $\mathsf{str}_i = \mathsf{str}$ and $y_i = \mathsf{Fail}$, then at the end of* ByzantineAgreementForFailures, *with probability $1-\mathsf{negl}(n)$, every honest party with $\mathsf{str}_i = \mathsf{str}$ will output $y_i = \mathsf{Fail}$.*

Proof. By Theorem 5.2, with probability $1-\mathsf{negl}(n)$ over str, each committee in the resulting quorum has at least $2/3$ fraction of honest parties. This means that with probability $1-\mathsf{negl}(n)$, the fraction of honest parties in each committee with $\mathsf{str}_i = \mathsf{str}$ is $2/3 - o(1) > 1/2$. Therefore, the prerequisite for the Distribute sub-protocol (see Appendix A) is satisfied and it results with every honest party setting $y_i = \mathsf{Fail}$. $\square$

We remain with the case where after the Agreement Phase only parties without knowledge of str have $y = \mathsf{Fail}$. However, as we show next, such a scenario is unlikely to occur.

Claim 5.6. *Assume that for at least $1-o(1)$ fraction of honest parties $\mathsf{str}_i = \mathsf{str}$. For every adversary $\mathcal{A} \in \mathscr{A}^{static}_{1/3}$, at the end of the Agreement Phase, if there is an honest party P_i with $y_i = \mathsf{Fail}$, then with probability $1-\mathsf{negl}(n)$, there is an honest party P_j with $\mathsf{str}_j = \mathsf{str}$ and $y_j = \mathsf{Fail}$.*

Proof. If for P_i it holds that $\mathsf{str}_i = \mathsf{str}$, we are done (as we can use $j = i$). So, assume that $\mathsf{str}_i \neq \mathsf{str}$ at the end of the Agreement Phase. As we argued in the proof of Claim 5.4, with probability $1-\mathsf{negl}(n)$, this scenario can occur only if there is at least one honest party $P_j \in S_i$ with $\mathsf{str}_j = \mathsf{str}$ that does not send a message to P_i in Step 2. From the way the protocol works, in this case, P_j sets $y_j \leftarrow \mathsf{Fail}$, as needed. $\square$

Now, we show that an adversary that is semi-honest with drops cannot force an honest party to not learn str.

Claim 5.7. *Assume that for at least $1-o(1)$ fraction of honest parties $\mathsf{str}_i = \mathsf{str}$. For every adversary $\mathcal{A} \in \mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}$, at the end of the Agreement Phase, with probability $1-\mathsf{negl}(n)$, each honest party P_i holds $\mathsf{str}_i = \mathsf{str}$ and $y_i = x_i$.*

Proof. As in the proof of Claim 5.6, for an honest party P_i, $\mathsf{str}_i \neq \mathsf{str}$ only if there is at least one honest party $P_j \in S_i$ that did not answer to P_i at Step 2. From the protocol description, the latter is possible only if P_j receives $2\cdot\log^2 n$ "pings". However, $\mathcal{A}$ is semi-honest with drops meaning that it has to follow the protocol except it can decide to drop some of its outgoing messages. So, by a Chernoff bound, too many "pings" arrive to P_j only with probability $1-e^{-(2/3+\epsilon)\cdot\log^2 n} = 1-\mathsf{negl}(n)$, and from a union bound, too many "pings" arrive to *some* honest party in S_i with probability $1-\mathsf{negl}(n)$. Finally, in Step 2 and Step 3 each honest party will never set $y \leftarrow \mathsf{Fail}$. $\square$

Finally, we are ready to combine all the above claims to prove Lemma 5.3.

of Lemma 5.3. From the correctness of the almost-everywhere agreement protocol, after the Almost-Everywhere Agreement Phase at least $1-o(1)$ fraction of honest parties holds $\mathsf{str}_i = \mathsf{str}$, where the min-entropy of str is $\Theta(\log^2 n)$. Let K be the set of honest parties that do not know str after the Agreement Phase, that is, $K \equiv \{P_i \mid P_i \in [n] \setminus \mathcal{C} \wedge \mathsf{str}_i \neq \mathsf{str}\}$ at the end of the Agreement Phase.

- If $K = \emptyset$:

  1. **Agreement**: By Claim 5.5, w.h.p $(1 - \mathsf{negl}(n))$ for each honest party P_i, $y_i \leftarrow \bigvee_{j \in [n] \setminus \mathcal{C}} y_j$.
  2. **Failure Detection**: From the protocol description, if $x_i = \mathsf{Fail}$, then at every step $y_i = \mathsf{Fail}$. So, if $\bigvee_{j \in [n] \setminus \mathcal{C}} x_j = \mathsf{Fail}$, then also $\bigvee_{j \in [n] \setminus \mathcal{C}} y_j = \mathsf{Fail}$.
  3. **Fairness**: If $\mathcal{A} \in \mathscr{A}_{Abort}$ and $\bigwedge_{j \in [n]} x_j = \mathsf{Success}$, then by Claim 5.7 w.h.p at the start of the Distribution Phase, for each party P_i, $y_i = \mathsf{Success}$, and therefore nobody will send "*ping*" at Step 1. So, at the end, each honest party will output $\mathsf{Success}$.

- If $K \neq \emptyset$:

  1. **Agreement**: By Claim 5.4, w.h.p it follows that each party in K will output Fail. By Claim 5.6, w.h.p it follows that there is a party $P_j \in [n] \setminus (\mathcal{C} \cup K)$ with $y_j = \mathsf{Fail}$, so from Claim 5.5 w.h.p it follows that each party in $[n] \setminus (\mathcal{C} \cup K)$ will output Fail.
  2. **Failure Detection**: The output of all parties is Fail by correctness of the Distribution sub-protocol.
  3. **Fairness**: By Claim 5.7, w.h.p for each honest party P_i, $\mathsf{str}_i = \mathsf{str}$ which mean that $P_i \notin K$, and so $K = \emptyset$. This is a contradiction to $K \neq \emptyset$, which can only happen if $\mathcal{A} \notin \mathscr{A}_{Abort}$ which is again a contradiction.

□

5.4 Proof of Theorem 5.1

In this section we prove Theorem 5.1, boosting protocols with local failure detection to ones with global failure detection.

The protocol Π'

Assumption: Π is a protocol that realizes $\mathcal{F}$ against $\mathscr{A}$ and realizes $\mathcal{F}^{lFD}$ against $\mathscr{A}^*$.

The protocol:

1. All parties participate in an execution of Π. At the end, each party P_i holds an output y_i.
2. For each party P_i, if $y_i = \mathsf{Fail}$, then P_i sets $x_i^* \leftarrow \mathsf{Fail}$; else $x_i^* \leftarrow \mathsf{Success}$.
3. All parties run the ByzantineAgreementForFailures protocol from Section 5.3 with $\{x_i^*\}_{i \in [n]}$. The outputs are denoted $\{y_i^*\}_i$.
4. For each party P_i, if $y_i^* = \mathsf{Fail}$, then P_i outputs Fail; else P_i outputs y_i.

In what follows we show that Π' realizes $\mathcal{F}$ against $\mathscr{A}$ and realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}^*$. For this we present two simulators.

5.4.1 Realizing $\mathcal{F}^{gFD}$ Against $\mathscr{A}^*$

Since Π realizes $\mathcal{F}^{lFD}$ against any $A \in \mathscr{A}^*$, then there is a simulator $\mathcal{S}^{lFD}$ that communicate with $\mathcal{F}^{lFD}$ in the ideal experiment and causes the ideal and real worlds to be indistinguishable; see Functionality 2 for the local failure detection functionality and Section 4.1 for the ideal world experiment. We will design a simulator for the global failure detection functionality as described in Functionality 1. The simulator is described next:

1. **Initialization**: $\mathcal{S}^{gFD}$ is initialized the same as $\mathcal{S}^{lFD}$.

2. **Evaluation**: $\mathcal{S}^{gFD}$ executes $\mathcal{S}^{lFD}$. In the case that $\mathcal{S}^{lFD}$ sends Fail in Step 2 of Functionality 2, then in Step 4, $\mathcal{S}^{lFD}$ provides a vector $(y'_1, \ldots, y'_n)$. $\mathcal{S}^{gFD}$ does not need to provide anything in Step 4 of Functionality 1, so it saves the vector for internal use.

3. **Output**:

   (a) $\mathcal{S}^{gFD}$ computes $\{\mathsf{view}_i^{lFD}\}_{i\in\mathcal{C}}$ exactly like $\mathcal{S}^{lFD}$.

   (b) For each honest party P_i, if $y'_i = \mathsf{Fail}$, then $\mathcal{S}^{gFD}$ set $x_i^* \leftarrow \mathsf{Fail}$, else $x_i^* \leftarrow \mathsf{Success}$.

   (c) $\mathcal{S}^{gFD}$ and $\mathcal{A}$ run the ByzantineAgreementForFailures protocol with $\{x_i^*\}_{i\in[n]}$, where $\mathcal{S}^{gFD}$ simulates the honest parties by running honestly. At the end, $\mathcal{S}^{gFD}$ holds $\{\mathsf{view}'_i\}_{i\in[n]}$, the view of the execution.

   (d) $\mathcal{S}^{gFD}$ outputs $\{\mathsf{view}_i\}_{i\in\mathcal{C}} = \{\mathsf{view}_i^{lFD}||\mathsf{view}'_i\}_{i\in\mathcal{C}}$

Claim 5.8. *Let Sim^{lFD} be a simulator for Π and the functionality $\mathcal{F}^{lFD}$. Then, $\mathcal{S}^{gFD}$, as described above, is a simulator for Π' and the functionality $\mathcal{F}^{gFD}$.*

Proof. Let $\{\mathsf{view}_i^{\Pi'}\}_{i\in\mathcal{C}} = \{\mathsf{view}_i^{\Pi'_1}||\mathsf{view}_i^{\Pi'_{2-4}}\}_{i\in\mathcal{C}}$ be the adversary view in the real world execution, split to the stages in Π'. Since Π realizes $\mathcal{F}^{lFD}$ against every $\mathcal{A} \in \mathscr{A}^*$, then $(\{\mathsf{view}_i^{\Pi'_1}\}_{i\in\mathcal{C}}, \{y'_i\}_{i\in[n]\setminus\mathcal{C}})$ is statistical indistinguishable from $(\{\mathsf{view}_i^{lFD}\}_{i\in\mathcal{C}}, \{y_i\}_{i\in[n]\setminus\mathcal{C}})$. Since $\mathcal{S}^{gFD}$ knows the inputs of the ByzantineAgreementForFailures protocol and act honestly, $\{\mathsf{view}_i^{\Pi'_{2-4}}\}_{i\in\mathcal{C}}$ and $\{\mathsf{view}'_i\}_{i\in\mathcal{C}}$ are identically distributed. Also, the outputs of the ByzantineAgreementForFailures protocol are statistical indistinguishable. To see this, we consider two cases:

- **There is an honest party P_i with $y_i^* = \mathsf{Fail}$:**
  - **Real world:** By bullet 1 of Lemma 5.3, we know that every honest party will output Fail, and so $\{y_i\}_{i\in\mathcal{H}} = \{\mathsf{Fail}, \ldots, \mathsf{Fail}\}$.
  - **Ideal world:** If there is an honest party P_i with $y_i^* = \mathsf{Fail}$, then $\mathcal{S}^{lFD}$ sends Fail at step 2 of Functionality 2. Thus, $\mathcal{S}^{gFD}$ also sends Fail at step 2 of Functionality 1, and so $\{y_i\}_{i\in\mathcal{H}} = \{\mathsf{Fail}, \ldots, \mathsf{Fail}\}$.

- **Otherwise (no honest party P_i has $y_i^* = \mathsf{Fail}$):**
  - **Real world:** By bullets 1 and 2 of Lemma 5.3 this case can occur (w.h.p) only if for every honest party P_i after the first stage of Π', $y_i \neq \mathsf{Fail}$. But, since Π realizes $\mathcal{F}^{lFD}$, then $\vec{y} \in \mathcal{F}(\mathcal{C}, \{\tilde{x}_i\}_{i\in[n]})$.
  - **Ideal world:** If no honest party P_i has $y_i^* = \mathsf{Fail}$, then $\mathcal{S}^{lFD}$ sends $\mathsf{Success}$ at step 2 of Functionality 2. This means that $\mathcal{S}^{gFD}$ also sends $\mathsf{Success}$ at step 2 of Functionality 1, and so $\vec{y} \in \mathcal{F}(\mathcal{C}, \{\tilde{x}_i\}_{i\in[n]})$.

Overall, the tuple $(\{\mathsf{view}_i\}_{i\in\mathcal{C}}, \{y_i\}_{i\in\mathcal{H}})$ is statistical indistinguishable between the real and ideal worlds. □

5.4.2 Realizing $\mathcal{F}$ Against $\mathscr{A}$

Since Π realizes $\mathcal{F}$ against an adversary $\mathcal{A} \in \mathscr{A}$, then there is a simulator $\mathcal{S}_\Pi$ that communicates with $\mathcal{F}$ in the ideal experiment and is able to generate the right view. The behavior of $\mathcal{S}$ in the ideal experiment (see Section 4.1) with $\mathcal{F}$ is described next:

1. **Initialization**: $\mathcal{S}$ simulates $\mathcal{S}_\Pi$.
2. **Evaluation**: $\mathcal{S}$ simulates $\mathcal{S}_\Pi$.
3. **Output**:
   (a) $\mathcal{S}$ computes $\{\mathsf{view}_i^{\Pi}\}_{i\in\mathcal{C}}$ by simulating $\mathcal{S}_\Pi$.
   (b) For each honest party P_i, $\mathcal{S}$ sets $x_i^* \leftarrow \mathsf{Success}$.
   (c) $\mathcal{S}$ and $\mathcal{A}$ run the ByzantineAgreementForFailures protocol with $\{x_i^*\}_{i\in[n]}$, where $\mathcal{S}$ simulates the honest parties and acts honestly. At the end, $\mathcal{S}$ holds $\{\mathsf{view}'_i\}_{i\in[n]}$, the view of the execution.
   (d) $\mathcal{S}$ outputs $\{\mathsf{view}_i\}_{i\in\mathcal{C}} = \{\mathsf{view}_i^{\Pi}||\mathsf{view}'_i\}_{i\in\mathcal{C}}$.

Claim 5.9. *Let Sim_Π be a simulator for Π and the functionality $\mathcal{F}$. Then, $\mathcal{S}$, as described above, is a simulator for Π' and the functionality $\mathcal{F}$.*

Proof. Let $\{\mathsf{view}_i^{\Pi'}\}_{i\in\mathcal{C}} = \{\mathsf{view}_i^{\Pi'_1}||\mathsf{view}_i^{\Pi'_{2-4}}\}_{i\in\mathcal{C}}$ be the adversary view in the real world execution, split to the stages in Π'. Since Π realizes $\mathcal{F}$ against $\mathcal{A}$, then $(\{\mathsf{view}_i^{\Pi'_1}\}_{i\in\mathcal{C}}, \{y_i^{real}\}_{i\in[n]\setminus\mathcal{C}})$ is statistical indistinguishable from
$(\{\mathsf{view}_i^{lFD}\}_{i\in\mathcal{C}}, \{y_i^{ideal}\}_{i\in[n]\setminus\mathcal{C}})$, and for every honest party P_i, the value of y_i^{real} after the stage 1 is not Fail. Since $\mathcal{S}$ knows the inputs of the ByzantineAgreementForFailures protocol and acts honestly, $\{\mathsf{view}_i^{\Pi'_{2-4}}\}_{i\in\mathcal{C}}$ and $\{\mathsf{view}'_i\}_{i\in\mathcal{C}}$ belong to the same distribution. This means that $\{\mathsf{view}_i^{\Pi'}\}_{i\in\mathcal{C}}$ is statistically indistinguishable from $\{\mathsf{view}_i\}_{i\in\mathcal{C}}$. From bullets 1 and 3 in Lemma 5.3 we obtain that $\{y_i^{real}\}_{i\in\mathcal{H}} \leftarrow \mathcal{F}(\mathcal{C}, \{\tilde{x}_i\}_{i\in[n]})$, which means that the tuple $(\{\mathsf{view}_i\}_{i\in\mathcal{C}}, \{y_i\}_{i\in\mathcal{H}})$ is statistical indistinguishable between the real and ideal worlds. □

6 Optimistic/Pessimistic Combiner Framework

In this section, we spell out the optimistic/pessimistic framework to obtain a full-fledged protocol implementing some given functionality. At a very high level, the starting point is an optimistic protocol for the functionality, where the protocol roughly assumes that parties behave honestly but has the property that it can recognize if some party deviated from the honest execution. We transform this protocol into a full-fledged protocol that is secure against all adversaries, even ones that do not follow the protocol specification.

In more detail, we first transform the given protocol into a protocol for the same functionality but with global failure detection (using Theorem 5.1). That is, the resulting protocol has the property that if some party notices that another party deviated from the protocol than all honest parties will know of this. Now, we execute the protocol with global failure detection; by our guarantee, if a failure was noticed by any party, all honest parties know of this, and can then fallback to some less efficient but always secure protocol.

Theorem 6.1. *Consider a functionality $\mathcal{F}$ and two attacker classes $\mathscr{A}^* \subseteq \mathscr{A}_{1/3}^{static}$ and $\mathscr{A} \subseteq \mathscr{A}_{Abort} \cap \mathscr{A}^*$. Assume that*

- *protocol* Π_{light} *realizes* $\mathcal{F}$ *against* $\mathscr{A}$ *and* $\mathcal{F}^{lFD}$ *against* $\mathscr{A}^*$*, and*
- *protocol* Π_{heavy} *realizes* $\mathcal{F}$ *against* $\mathscr{A}^*$*.*

Then, there exists a protocol Π *that realizes* $\mathcal{F}$ *against* $\mathscr{A}^*$ *with the following efficiency properties:*

1. Worst-case communication complexity*: For every party* $i \in [n]$*:*
$$\mathsf{cc}(\Pi, i) = \mathsf{cc}(\Pi_{light}, i) + \mathsf{cc}(\Pi_{heavy}, i) + \mathsf{polylog}(n).$$
2. Adversary-dependent communication complexity*: For every party* $i \in [n]$
$$\mathsf{cc}(\Pi, \mathscr{A}, i) = \mathsf{cc}(\Pi_{light}, i) + \mathsf{polylog}(n).$$
3. Worst-case round complexity*:* $\mathsf{rc}(\Pi) = \mathsf{rc}(\Pi_{light}) + \mathsf{rc}(\Pi_{heavy}) + \mathsf{polylog}(n)$*.*
4. Adversary-dependent round complexity*:* $\mathsf{rc}(\Pi, \mathscr{A}) = \mathsf{rc}(\Pi_{light}) + \mathsf{polylog}(n)$*.*

Proof. First, use Theorem 5.1 to compile Π_{light} (that only realizes $\mathcal{F}^{lFD}$ against $\mathscr{A}^*$) into Π'_{light} (that realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}^*$). Now, we describe Π.

Protocol Π:

1. Run Π'_{light}.
2. For each (honest) party P_i, if $y_i = \mathsf{Fail}$, then participant in Π_{heavy}; Otherwise, output the output of Π'_{light} and terminate.

Efficiency. In the worst-case, $\mathsf{cc}(\Pi, i) = \mathsf{cc}(\Pi'_{light}, i) + \mathsf{cc}(\Pi_{heavy}, i)$ and $\mathsf{rc}(\Pi, \mathscr{A}) = \mathsf{rc}(\Pi'_{light}, \mathscr{A}) + \mathsf{rc}(\Pi_{heavy}, \mathscr{A})$. Now, by Theorem 5.1, $\mathsf{cc}(\Pi'_{light}, i) = \mathsf{cc}(\Pi_{light}, i) + \mathsf{polylog}(n)$ and $\mathsf{rc}(\Pi'_{light}, \mathscr{A}) = \mathsf{rc}(\Pi_{light}) + \mathsf{polylog}(n)$. This proves bullets 1 and 3. If the adversary is in $\mathscr{A}$ (meaning it does not deviate from the protocol except by possibly dropping messages), then by the guarantee of Π'_{light}, there will not be an honest party P_i with $y_i = \mathsf{Fail}$, and so all honest parties will not participant in Π_{heavy}. Thus, in this case, the complexity of the protocol is: $\mathsf{cc}(\Pi, \mathscr{A}, i) = \mathsf{cc}(\Pi'_{light}, i) = \mathsf{cc}(\Pi_{light}, i) + \mathsf{polylog}(n)$ and $\mathsf{rc}(\Pi, \mathscr{A}) = \mathsf{rc}(\Pi'_{light}) = \mathsf{rc}(\Pi_{light}) + \mathsf{polylog}(n)$. This proves bullets 2 and 4.

Security. Since Π'_{light} realizes $\mathcal{F}^{gFD}$ against $\mathscr{A}^*$, then by definition if an honest party P_i has $y_i = \mathsf{Fail}$, then we are guaranteed that all honest parties have $y = \mathsf{Fail}$. Thus, all honest parties will participate in Π_{heavy}, and since Π_{heavy} is secure against $\mathscr{A}^*$, then Π is also secure against $\mathscr{A}^*$. Formally, we define a simulator for Π and functionality $\mathcal{F}$ as follows: First run the simulator of Π'_{light} and then if all honest parties have $y = \mathsf{Fail}$, then run the simulator of Π_{heavy}. The proof that this simulator generated an indistinguishable view is straightforward given the guarantees of the underlying two simulators. □

7 Applications

In this section we present several instantiations of our framework. First, in Section 7.1, we show how to build a quorum, i.e., a well-formed representative committee for each party in the protocol. This is a useful abstraction on its own because (as we show in Section 7.2) it can be used to directly obtain several other applications, essentially without added complexity. For example, we show how to use it to directly obtain a Byzantine Agreement protocol, a broadcast protocol, and a committee election protocol. All of these have super efficient "optimistic" complexity and the best known worst-case overhead as fallback. As another (less immediate) application, in Section 7.3, we build a scalable secure multiparty computation (SMPC) protocol with almost no overhead in an optimistic execution beyond the circuit size.

7.1 A Quorum

We start by defining the properties of a good quorum. We implement the construction of a quorum via a distributed protocol and so it is convenient to specify the properties of a quorum via a functionality. At a very high level, a quorum is a collection of n poly-logarithmic size subsets where (1) each subset contains roughly the same fraction of bad parties as in the total population, (2) Each party is a member in poly-logarithmically-many committees. We parametrize a quorum functionality with $\epsilon > 0$, where the adversary controls constant $< \epsilon$ fraction of parties in the total population.

Functionality 3: $\mathcal{F}_{\epsilon\text{-quorum}}$ - Quorum Building

1. $\mathcal{S}$ specifies a function $f\colon [n] \to [n]^d$, where $d \in \tilde{O}(1)$, such that:
   (a) For all $i \in [n]$, $\frac{|f(i) \cap \mathcal{C}|}{|f(i)|} < \epsilon$.
   (b) Let $f^{-1}(j) = \{i \mid j \in f(i)\}$. Then $|f^{-1}(j)| \in O(d)$ for all $j \in [n]$.
2. $\mathcal{F}$ outputs f to every party P_i.

"Heavyweight" quorum building. A protocol for generating a 1/3-quorum in the presence of attackers that statically corrupts $1/3 - \epsilon$ fraction of parties for a constant $\epsilon > 0$ is known [GK24]. The protocol consists of $\tilde{O}(1)$ rounds and $\tilde{\Theta}(\sqrt{n})$ communication per party.

Lemma 7.1 (Heavyweight quorum generation [GK24]). *There is a protocol Π_{heavy} realizing $\mathcal{F}_{1/3\text{-quorum}}$ against $\mathscr{A}^{static}_{1/3}$. The protocol consists of $\tilde{O}(1)$ rounds and $\tilde{\Theta}(\sqrt{n})$ communication per party.*

"Lightweight" quorum building. For the lightweight protocol, we observe that it was actually obtained in ByzantineAgreementForFailures (Section 5.3). Specifically, we run this protocol up until the end of the Agreement Phase, where the inputs of all parties are $\mathsf{Success}$, and the output of P_i is Fail if $y_i = \mathsf{Fail}$. If not, the output is $f_{\mathsf{str}_i}(\star) \triangleq H(\star, \mathsf{str}_i)$, where H is the function from Theorem 5.2.

See formal description in Section 7.1.1. The proof that we obtain a realization of the 1/3-quorum functionality readily follows from Claims 5.4 and 5.7. We thus have the following lemma whose proof can be found in Section 7.1.1.

Lemma 7.2 (Lightweight quorum generation). *There is a protocol Π_{light} (described above) that realizes $\mathcal{F}_{1/3\text{-quorum}}$ against $\mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}$ and realizes $\mathcal{F}^{lFD}_{1/3\text{-quorum}}$ against $\mathscr{A}^{static}_{1/3}$.*

We apply our optimistic-pessimistic framework from Theorem 6.1 using Lemmas 7.1 and 7.2. Thus, We get the following result.

Theorem 7.3 (Quorum building). *There is a protocol Π that realizes $\mathcal{F}_{1/3\text{-quorum}}$ against adversaries family $\mathscr{A}^{static}_{1/3}$. Furthermore, Π has the following efficiency properties:*

1. $\mathsf{rc}(\Pi) = \tilde{O}(1)$. *(Round complexity)*
2. *For each party P_i,* $\mathsf{cc}(\Pi, \mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}, i) \in \tilde{O}(1)$. *(Optimistic complexity)*
3. *For each party P_i,* $\mathsf{cc}(\Pi, i) \in \tilde{O}(\sqrt{n})$. *(Pessimistic complexity)*

7.1.1 Lightweight Quorum Building Protocol (Proof of Lemma 7.2)

In this protocol, we associate Fail with 1 and $\mathsf{Success}$ with 0, so logical Boolean operations are well defined.

Lightweight Quorum Building Protocol (Π_{light})

- **Setup Phase:** Each party P_i sets $y_i \leftarrow \mathsf{Success}$.
- **Almost-Everywhere Phase:** All the parties run together the almost-every agreement protocol from [KLST11]. At the end, each party P_i holds a string str_i of size $\Theta(\log^3 n)$.
- **Agreement Phase:**
  1. Each party P_i samples a set of $\log^2 n$ parties (denoted S_i), and sends them "*ping*".
     Filtering: Each party accepts messages of size $|$"*ping*"$|$, and at most $2 \cdot \log^2 n$ of them.
  2. For each party P_j: if it received less that $2 \cdot \log^2 n$ "*ping*"s, P_j answers with str_j; else, P_j set $y_j \leftarrow \mathsf{Fail}$.
     Filtering: P_i accepts messages of size at most $\log^4 n$ and only from parties in S_i.
  3. For each party P_i, if there is a string str^* that was received more than $|S_i|/2$ times, P_i sets $\mathsf{str}_i \leftarrow \mathsf{str}^*$; else, P_i sets $y_i \leftarrow \mathsf{Fail}$.
- **Output Phase:** For each party P_i if $y_i = \mathsf{Fail}$ then output Fail; else, output $y_i = f_{\mathsf{str}_i}(\star) \triangleq H(\star, \mathsf{str}_i)$, where H is the $\frac{1}{3}$-quorum building function from Theorem 5.2.

We proceed with the proof of correctness. Since the inputs are fixed in the protocol description, the simulator for an adversary $\mathcal{A}$ simulates the honest parties to obtain the view of $\mathcal{A}$. By Claim 5.4, it follows that the output of each honest party is either f_{str_i} (the same function for all honest parties) or Fail. By Claim 5.2 (see also [KLST11]) with all but negligible probability of error, for str with $\Omega(\log^2 n)$ (min-)entropy, $H(\star, \mathsf{str})$ defines a good quorum. Thus, $f_{\mathsf{str}} \in \mathcal{F}_{\frac{1}{3}\text{-quorum}}(\mathcal{C})$, and so Π_{light} realizes $\mathcal{F}^{lFD}_{\frac{1}{3}\text{-quorum}}$ against $\mathcal{A}$.

By Claim 5.7 it follows that if $\mathcal{A} \in \mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}$, then the output of all honest parties is f_{str} (nobody will output Fail). This means that Π_{light} realizes $\mathcal{F}_{\frac{1}{3}\text{-quorum}}$ against $\mathcal{A}$.

7.2 Byzantine Agreement, Broadcast, and Committee Election

Once we have a quorum (as in Section 7.1), it is relatively easy to obtain other useful abstractions with essentially no extra complexity. Specifically, we obtain Byzantine Agreement, Broadcast, and Committee Election (formal definitions in Section 3.2), and more with essentially optimal optimistic complexity and otherwise the best-known worst-case complexity.

Theorem 7.4. *There are $\tilde{O}(1)$-round Byzantine agreement, broadcast, quorum/committee/leader election protocols, each with negligible (in n) probability of failure, and with the following complexity:*

- *For each party, if the adversary is in $\mathscr{A}^{static}_{1/3}$, the communication complexity is $\tilde{O}(\sqrt{n})$.*
- *For each party, if the adversary is in $\mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}$, the communication complexity is $\tilde{O}(1)$.*

It is rather straightforward to obtain the above theorem given the quorum functionality realization in Section 7.1. We sketch the details next. Consider the Byzantine Agreement problem in which every party has an input bit and the goal is to reach an agreement on an input which is held by one of the honest parties. We first generate a quorum using Theorem 7.3. Then, each party sends its bit to its associated committee members. The committees now perform agreement in a tree-like fashion by say computing the majority value of their input bits (recall that the committees can be viewed as honestly-behaving parties so we need not worry about Byzantine behavior). Finally, all committees distribute the agreed upon bit to their respective associated parties. Broadcast is implemented in a similar fashion by performing broadcast in a tree-like fashion between the committees defined via the quorum (again, since these committees are behaving honestly, this is easy). Committee election is a special case of quorum[9] (the first committee in the quorum is a good committee). Leader election can be implemented by running a naive (heavy-weight) protocol among a chosen committee.

7.3 Secure Multiparty Computation

In MPC there are n parties, each having an input x_i. The goal is to jointly evaluate a known functionality described by a circuit C without revealing unnecessary information about the private inputs.

This can be captured within our general real-ideal simulation given in Section 4.1 using the following functionality.

Functionality 4: $\mathcal{F}^C$ - MPC for a circuit C

1. Every party P_i submits $\tilde{x}_i$, where for honest parties $\tilde{x}_i = x_i$.
2. Compute $(y_1, \dots, y_n) \leftarrow C(\tilde{x}_1, \dots, \tilde{x}_n)$.
3. For every party P_i, output y_i.

The size of a circuit C is defined as the total number of wires in C, and is denoted $|C|$. We further denote the depth of C by $\mathit{Depth}(C)$ and define it as the length of the longest path from an input wire to an output wire. We shall assume that each gate in C has a constant number of input and output wires; say two inputs and two outputs.

Recall that our communication model assumes only point-to-point synchronous communication. The communication is authenticated and private. (This is in contrast to the rest of the paper where communication is only authenticated but not private.) We obtain an MPC protocol that has worst case complexity $\tilde{O}(|C|/n + \sqrt{n})$ per party when computing C. If the adversary is honest, then the complexity per party is $\tilde{O}(|C|/n)$. In either case, the round complexity is $\tilde{O}(\mathit{Depth}(C))$. Notice that if we are considering a linear size circuit, then the complexity of an honest execution is essentially the best possible. The protocol uses techniques similar to those in [DKM$^+$17], but our protocol appears to be a simpler. Additionally, we employ our scalable optimistic/pessimistic protocol for quorum generation. We refer to Appendix B for full details.

Theorem 7.5 (Optimistic/Pessimistic Scalable MPC). *There is an MPC protocol* Π *such that for every circuit* C*, it has the following complexity:*

- $\mathsf{rc}(\Pi) \in \tilde{O}(\mathit{Depth}(C))$. *(Round complexity)*

[9] Usually, we want committees of size poly-logarithmic and have the property that the number of corrupted parties in the committee be roughly the same as their fraction in the total population.

- *For each party P_i,* $\mathsf{cc}(\Pi^C_{MPC}, \mathscr{A}_{Abort} \cap \mathscr{A}^{static}_{1/3}, i) \in \tilde{O}(|C|/n)$. *(Optimistic complexity)*
- *For each party P_i,* $\mathsf{cc}(\Pi^C_{MPC}, i) \in \tilde{O}(|C|/n + \sqrt{n})$. *(Pessimistic complexity)*

Acknowledgements

Ilan Komargodski is the incumbent of the Harry & Abe Sherman Senior Lectureship at the School of Computer Science and Engineering at the Hebrew University. 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).

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.

[AL17] Gilad Asharov and Yehuda Lindell. A full proof of the bgw protocol for perfectly secure multiparty computation. *Journal of Cryptology*, 30(1):58–151, 2017.

[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.

[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.

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

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS*, pages 136–145. IEEE Computer Society, 2001.

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

[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.

[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.

[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.

[DKMS14] Varsha Dani, Valerie King, Mahnush Movahedi, and Jared Saia. Quorums quicken queries: Efficient asynchronous secure multiparty computation. In *Distributed Computing and Networking - ICDCN*, pages 242–256, 2014.

[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.

[FGK24] Rex Fernando, Yuval Gelles, and Ilan Komargodski. Scalable distributed agreement from LWE: byzantine agreement, broadcast, and leader election. In *ITCS*, pages 46:1–46:23, 2024.

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

[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.

[GK24] Yuval Gelles and Ilan Komargodski. Optimal load-balanced scalable distributed agreement. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing*, pages 411–422, 2024.

[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.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications.* Cambridge University Press, 2004.

[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 $\tilde{\mathrm{o}}(\mathrm{n}^{3/2})$ bits. In *Distributed Computing, 23rd International Symposium, DISC*, pages 464–478, 2009.

[KS10] Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC*, pages 420–429, 2010.

[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.

[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.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, pages 612–613, 1979.

[WB86] Lloyd R. Welch and Elwyn R. Berlekamp. Error correction for algebraic block codes, U.S. Patent US4633470A, 1986.

A The Distribute Sub-Protocol

For convenience, we assume that party and committee IDs go from 0 to $n-1$. In this protocol we assume a quorum, meaning that there are n committees ($Q_0 \dots Q_{n-1}$), where in each committee there is a majority of honest parties that have knowledge about the quorum. When we say that a committee sends a message, we mean that all the parties in the committee send this message, and when we say that a message was received by a committee, we mean that it was received by a majority of the parties in the committee.

The protocol is very simple, all the committees communicate their values upstream in tree-like communication pattern, calculating a logical OR along the way, and then broadcast the results in reverse order, it takes $O(\log n)$ rounds. Since the communication pattern and messages sizes are fixed (given n), we avoid writing the filtering rules.

The Distribute Protocol

Input: Each committee Q_i holds a bit x_i, indicating that P_i sent a message to Q_i.

Output: Each committee Q_i holds a bit x, which is the logical OR of all the indicators.

The Protocol:

```
(a) For $r$ in $1 \dots \lceil \log n \rceil + 1$:
    i. For each committee $Q_i$ and if $i \bmod 2^{r-1} = 0$:
        A. If $r \neq 1 \wedge i + 2^{r-2} < n$, $Q_i$ set $y_i$ to be the value that was received most times
           from $Q_{i+2^{r-2}}$ in previous round (0 if there is a tie), else, $Q_i$ set $y_i \leftarrow 0$.
        B. $Q_i$ set $x_i \leftarrow x_i \vee y_i$.
        C. If $i \bmod 2^r \neq 0$, $Q_i$ sends $(i, x_i)$ to each party in $Q_{i-2^{r-1}}$.
(b) $Q_0$ set $x \leftarrow x_0$, and all other committees set $x \leftarrow \perp$.
(c) For $r$ in $\lceil \log n \rceil \dots 0$:
    i. For each committee $Q_i$ and if $i \bmod 2^r = 0$:
        A. If $i \neq 0 \wedge i \bmod 2^{r+1} \neq 0$, $Q_i$ set $x$ to be the value that was received most times
           from parties in $Q_{i-2^r}$ in previous round (0 if there is a tie).
        B. If $r \neq 0$, $Q_i$ sends $x_0$ to each party in $Q_{i+2^{r-1}}$.
```

B Secure Multiparty Computation

The BGW protocol and secret sharing. Our protocol will rely on the generic MPC protocol of Ben-Or et al. [BGW88]. This protocol gives a *perfectly* secure general purpose protocol. Security holds for a static malicious adversary corrupting $t < n/3$ of the parties, where n is the number of parties, and even with concurrent composition. The protocol supports all computations and assumes they are given as circuits. Given a circuit of depth d and size s (where size is measured as the number of wires), the protocol requires $O(d)$ rounds and each party sends $s \cdot \mathsf{poly}(n)$ bits overall.

Looking forward, we will use this MPC protocol only among poly-logarithmic size groups of parties and only for computing (poly-)logarithmic size circuits. Thus, the communication and round overhead per invocation of these protocols will be $\tilde{O}(1)$. (Recall that even though we use a polynomial-time broadcast algorithm [DS83], we execute it among poly-logarithmic number of parties).

Theorem B.1 (Perfectly secure heavyweight MPC e.g., [AL17])**.** *Let f be an n-ary functionality and C its arithmetic circuit representation. There exists an explicit protocol Π^{MPC}_{Heavy} realizing $\mathcal{F}^C$ with perfect universally composable security against static adversaries that corrupt $< n/3$ parties. Furthermore, $\mathsf{rc}(\Pi^{MPC}_{Heavy}) \in O(Depth(C))$ and $\mathsf{cc}(\Pi^{MPC}_{Heavy}) \in \mathsf{poly}(|C|, n)$.*

Furthermore, we will rely on a robust secret sharing scheme, a technique for splitting a secret s among n parties in such a way that every coalition of t parties can reconstruct the secret and every coalition of $t-1$ parties learn nothing about the secret. A well-known method achieving the above was suggested by Shamir [Sha79]. We will need a strong notion of correctness called *robustness*: large enough sets of parties that try to reconstruct the secret will succeed even if some of the parties try to cheat in the reconstruction phase. Concretely, we will need that the n parties always succeed in reconstructing the secret, even if there are $t-1$ parties that try to cheat. This property is known to be achievable for Shamir's scheme if $t < n/3$. Note that the sharing and reconstruction procedures are efficient, namely, there is a polynomial time algorithm (in n) that implements them.

The MPC Protocol. We denote the circuit corresponding to $n/3$-out-of-n Shamir's secret sharing scheme [Sha79] by $\mathsf{share}_n(x, r)$. For robust reconstruction, we use the (efficient, polynomial time) procedure of [WB86]. When we say that committee is associated with outputs of the share circuit, we mean that the ith share is associated with the ith member of the committee in lexicographic order.

Secure Multiparty Computation Protocol

- **Setup Phase (Quorum Building):**
  1. All parties run the quorum building protocol from Theorem 7.3. At the end, each party holds a function f as described in Functionality 3. Denote the ith committee ($f(i)$) by Q_i, and let q_i be the size of each committee.
  2. Associate each committee Q_i with $|C|/n$ wires.
- **Share Phase:** Each party P_i and committee Q_i perform a "heavy" MPC protocol (Theorem B.1) on $\mathsf{share}_{q_i}(x_i, r_i)$, where r_i is fresh randomness.
- **Evaluation Phase:**
  1. For layer $\ell \in 1 \ldots Depth(C)$:
     - For each gate G in layer ℓ, do (in parallel):
       * All the committees that are associated with wires that are connected to G perform an MPC protocol (Theorem B.1) on $C^{q_i}_{\mathsf{G}}$, where $C^{q_i}_{\mathsf{G}}$ is a circuit that wraps the functionality of the gate by first reconstructing the input wire values, then evaluating the gates, and finally resharing the output with the committee members of associated with the output wires. This circuit is formally defined in Appendix B.3.
- **Output Phase (Reconstruct):** Each party P_i reconstructs the output value by pulling shares from its committee members Q_i.

B.1 Security

We specify the simulator $\mathcal{S}$ for the ideal experiment with $\mathcal{F}^C$ (Functionality 4). Note that we will get statistical security due to the statistical error in the quorum building protocol from Theorem 7.3.

1. **Initialization**: $\mathcal{S}$ chooses the same set of corrupted parties $\mathcal{C}$ as $\mathcal{A}$.
2. **Evaluation**: $\mathcal{S}$ run the setup and share phases with $\mathcal{A}$, where $\mathcal{S}$ simulates every honest party P_i with $x_i = 0$. At the end, $\mathcal{S}$ extracts $\{\tilde{x}_i, r_i\}_{i \in \mathcal{C}}$ using $q_i/3$ shares of honest parties. Then, it sends $\{\tilde{x}_i\}_{i \in \mathcal{C}}$ to $\mathcal{F}^C$, and receives back $\{y_i\}_{i \in \mathcal{C}}$.
3. **Output**: $\mathcal{S}$ run the protocol with $\mathcal{A}$ with the same randomness as before, with the following changes:
   (a) **Setup Phase**: $\mathcal{S}$ invokes the simulator of the quorum building protocol from Theorem 7.3; at the end $\mathcal{S}$ holds $f_{\mathcal{S}}$.
   (b) **Share Phase**: $\mathcal{S}$ invokes the simulator of n concurrent executions of the "heavy" MPC protocol, where $\mathcal{S}$ simulates $\mathcal{F}^C$ as follows:
      - For $P_i \in \mathcal{C}$: $\mathcal{S}$ outputs $\mathsf{share}_{q_i}(\tilde{x}_i, r_i)$ using the values that were extracted previously.
      - For $P_i \notin \mathcal{C}$: $\mathcal{S}$ outputs $\mathsf{share}_{q_i}(0, r)$ using fresh randomness.
   (c) **Evaluation Phase**: For layer $\ell \in 1 \dots \mathit{Depth}(C)$, let k be the number of gates in layer ℓ. $\mathcal{S}$ invokes the simulator of k concurrent executions of the "heavy" MPC protocols, where for each such execution $\mathcal{S}$ simulates the functionality $\mathcal{F}^{C^{q_i}_{\mathsf{G}}}$ as follows:
      - If the output of G corresponds to an output y_i:
        $\mathcal{S}$ outputs $(\mathsf{share}_{q_i}(y_i, r_1), \mathsf{share}_{q_i}(0, r_2))$ using fresh randomness.
      - Else, $\mathcal{S}$ outputs $(\mathsf{share}_{q_i}(0, r_1), \mathsf{share}_{q_i}(0, r_2))$ using fresh randomness.

To show that the above simulator succeeds in simulating the view of the real world, we proceed by considering five intermediate hybrids. The first hybrid corresponds to the ideal experiment, the last hybrid corresponds to the real experiment, and in each intermediate hybrid, we specify one component that we change.

- **Hybrid 0:** This worlds corresponds to the ideal experiment with $\mathcal{S}$ as above.
- **Hybrid 1:** In this world $\mathcal{S}$ behave as in Hybrid 0, except that in the Setup Phase, $\mathcal{S}$ behaves like in the real world execution by using f instead of the simulated ones $f_{\mathcal{S}}$.
- **Hybrid 2:** In this world $\mathcal{S}$ behave as in Hybrid 1, except that $\mathcal{S}$ knows the honest parties inputs, and in the Share Phase, when $\mathcal{S}$ simulates $\mathcal{F}^C$, for $P_i \notin \mathcal{C}$, $\mathcal{S}$ outputs $\mathsf{share}_{q_i}(x_i, r)$ using fresh randomness.
- **Hybrid 3:** In this world, $\mathcal{S}$ behaves as in Hybrid 2, except that in the Share Phase, $\mathcal{S}$ behaves as in the real world execution.
- **Hybrid 4:** In this world, $\mathcal{S}$ behaves as in Hybrid 3, except that in the Evaluation Phase, $\mathcal{S}$ behaves as in the real world execution. This hybrid is identical to the real world.

Let $(\mathsf{view}_i^{(k)}, y_i^{(k)})$ be the view and output of P_i in Hybrid k. Denote $\mathcal{H} = [n] \setminus \mathcal{C}$, the set of honest parties.

Claim B.2. *The distributions* $(\{\mathsf{view}_i^{(0)}\}_{i \in \mathcal{C}}, \{y_i^{(0)}\}_{i \in \mathcal{H}})$ *and* $(\{\mathsf{view}_i^{(1)}\}_{i \in \mathcal{C}}, \{y_i^{(1)}\}_{i \in \mathcal{H}})$ *are statistically close.*

Proof. Follows directly from the security of the protocol from Theorem 7.3. ☐

Claim B.3. *The distributions* $(\{\mathsf{view}_i^{(1)}\}_{i\in\mathcal{C}}, \{y_i^{(1)}\}_{i\in\mathcal{H}})$ *and* $(\{\mathsf{view}_i^{(2)}\}_{i\in\mathcal{C}}, \{y_i^{(2)}\}_{i\in\mathcal{H}})$ *are identical.*

Proof. By Theorem B.1 it follows that the adversary learns nothing about $\mathcal{S}$'s inputs from the execution of the heavy MPC protocol. From the security of the secret sharing scheme, it follows that the adversary's outputs are independent of the inputs. Overall, the distribution of the view that is generated by the heavy MPC simulator is identical for all possible inputs. ☐

Claim B.4. *The distributions* $(\{\mathsf{view}_i^{(3)}\}_{i\in\mathcal{C}}, \{y_i^{(2)}\}_{i\in\mathcal{H}})$ *and* $(\{\mathsf{view}_i^{(3)}\}_{i\in\mathcal{C}}, \{y_i^{(3)}\}_{i\in\mathcal{H}})$ *are identical.*

Proof. For each party $P_i \notin \mathcal{C}$ the claim is true by Theorem B.1. For each party $P_i \in \mathcal{C}$, we use the fact that $\mathcal{S}$ extracts the right $\tilde{x}_i$ and r_i. Indeed, by Theorem B.1, $\mathcal{S}$ receives at least $\frac{2q_i}{3}$ shares of $\tilde{x}_i$, and by the parameters of the secret sharing scheme, $\mathcal{S}$ correctly extracts both $\tilde{x}_i$ and r_i, as needed. ☐

To prove that Hybrid 3 and Hybrid 4 are identical we define additional sub-hybrids as follows:

- **Hybrid** $(3, \ell)$: In this world, $\mathcal{S}$ behaves as in Hybrid 3, except that in the first ℓ layers of the Evaluation Phase, $\mathcal{S}$ behave as in the real world execution.

Clearly, Hybrid $(3, 0)$ is identical to Hybrid 3, and Hybrid $(3, \mathit{Depth}(C))$ is identical to Hybrid 4. Also, for convenience and without loss of generality, we assume that the gates connected to output wires are in layer $\mathit{Depth}(C)$.

Claim B.5. *For every* $\ell \in [\mathit{Depth}(C) - 1]$*, the distributions* $(\{\mathsf{view}_i^{(3,\ell)}\}_{i\in\mathcal{C}}, \{y_i^{(3,\ell)}\}_{i\in\mathcal{H}})$ *and* $(\{\mathsf{view}_i^{(3,\ell-1)}\}_{i\in\mathcal{C}}, \{y_i^{(3,\ell-1)}\}_{i\in\mathcal{H}})$ *are identical.*

Proof. By the construction of the $C_{\mathsf{G}}^{q_i}$ circuit, it follows that each output is computed by $\mathsf{share}_{q_i}(\star, r^*)$, where r^* is XOR of random values submitted by committee members. From Theorem B.1, it follows that the adversary learns nothing about the inputs from the execution of the heavy MPC protocol, and so since the XOR contains at least one unknown uniformly random value, r^* is also uniformly random and unknown to the adversary. From the security of the secret sharing scheme, it follows that the outputs of the adversary are independent of the private inputs. Overall, the distribution of the view that is generated by the heavy MPC simulator is the same for all inputs. ☐

Note that, up to this point the outputs of the adversary are generated by the ideal functionality. In the following claim, we prove the "correctness" of the protocol.

Claim B.6. *In the real execution, assuming the quorum building protocol succeeds, for every input vector* $\vec{x}$ *and gate* $\mathsf{G} \in C$*, the result of applying the reconstruction scheme on the outputs of* share *in the wrapped circuit is equal to the value of output of* G *in* C *(without wrapping).*

Proof. We prove this by induction on the layers of the circuit.

- Base case $(\ell = 0)$: For each party P_i, by the security of the heavy MPC protocol, the output of each honest party $P_j \in Q_i$ at the end of the Share Phase is the output of $\mathsf{share}_{q_i}(x_i, r_i)$. Thus, more than $2q_i/3$ honest parties hold the right shares of x_i, and so from the robustness property, the reconstruction scheme will output x_i.

- Inductive step: Assume that the claim holds for all $\ell \in [l]$. For layer $l+1$, Let $C_{\mathsf{G}}^{q_i}$ be a circuit on layer $l+1$. In the construction of $C_{\mathsf{G}}^{q_i}$, at first the reconstruction scheme is applied for each G input to obtain x_1 and x_2, then $\mathsf{G}(x_1, x_2)$ calculated, and lastly, the share scheme is applied for each output. From the security of the heavy MPC it follows that the result of applying the reconstruction scheme on each share circuit output will be same as G's output.

□

Claim B.7. *The distributions* $(\{\mathsf{view}_i^{(3,Depth(C))}\}_{i\in\mathcal{C}}, \{y_i^{(3,Depth(C))}\}_{i\in\mathcal{H}})$ *and* $(\{\mathsf{view}_i^{(3,Depth(C)-1)}\}_{i\in\mathcal{C}}, \{y_i^{(3,Depth(C)-1)}\}_{i\in\mathcal{H}})$ *are identical.*

Proof. The proof of this claim is analogous to the proof of Claim B.5. By Claim B.6 with $\ell = Depth(C)$, it follows that the outputs in the real execution are equal to $C(\tilde{x}_1, \ldots, \tilde{x}_n)$, i.e., the output of the functionality $\mathcal{F}^C$. □

B.2 Efficiency

By Claim B.8, each $C_{\mathsf{G}}^{q_i}$ is of size polynomial in the committee size and the latter is poly-logarithmic in n (Functionality 3). That is, $|C_{\mathsf{G}}^{q_i}| = \mathsf{poly}(q_i) = \mathsf{polylog}(n)$. Together with Theorem B.1, each heavy MPC on $C_{\mathsf{G}}^{q_i}$ costs $\mathsf{polylog}(n)$ rounds and communication; the same goes for the share and output phases. Since for each layer in the circuit C we compute all the gates in parallel for the share, evaluation, and output phases, we need $\tilde{O}(Depth(C))$ rounds, and $\tilde{O}(|C|/n)$ communication per party. Together with Theorem 7.3, we got the following complexities:

- $\mathsf{rc}(\Pi_{MPC}^{C}) \in \tilde{O}(Depth(C))$.
- For each party P_i, $\mathsf{cc}(\Pi_{MPC}^{C}, i) \in \tilde{O}(|C|/n + \sqrt{n})$.
- For each party P_i, $\mathsf{cc}(\Pi_{MPC}^{C}, \mathscr{A}_{Abort} \cap \mathscr{A}_{1/3}^{static}, i) \in \tilde{O}(|C|/n)$.

B.3 The Gate Wrapper

Given a gate G with two inputs and two outputs, and a pair of circuits share_n and rec_n that represent the sharing and reconstruction mechanism of the robust variant of Shamir's scheme [Sha79] (i.e., using the reconstruction of [WB86]), we build a circuit C_{G}^{n} with $4n$ inputs and $2n$ outputs as follows, and as depicted in Figure 1:

- Connect each input of G to the output of a new rec_n circuit.
- Connect each output of G to the first (value) input of a new share_n circuit.
- Connect the second (randomness) input of each share_n circuit to a new XOR_n circuit, where XOR_n is implemented in a tree-like manner.

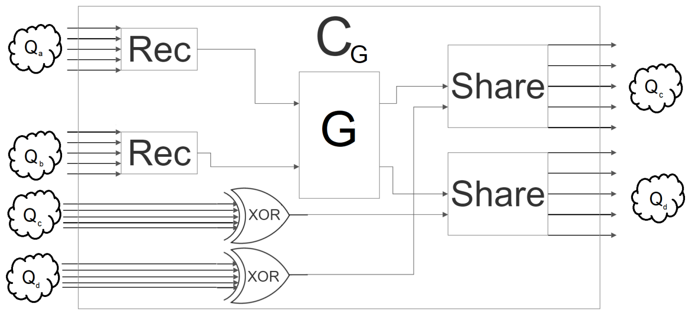

Figure 1: The wrapper of a gate G.

Since the sharing and reconstruction procedures can be implemented in polynomial time and since all other computations are efficient, we conclude that the wrapper can be written as a polynomial size circuit.

Corollary B.8. *For each gate* G*,* $|C^n_{\mathsf{G}}| \in \mathsf{poly}(n)$*.*