Byzantine Agreement Decomposed: Honest Majority Asynchronous Total-Order Broadcast from Reliable Broadcast

Simon Holmgaard Kamp and Jesper Buus Nielsen

Aarhus University*
{kamp,jbn}@cs.au.dk

Abstract. It is well-known that Asynchronous Total Order Broadcast (ATOB) requires randomisation and that at most $t < n/3$ out of n players are corrupted. This is opposed to synchronous total-order broadcast (STOB) which can tolerate $t < n/2$ corruptions and can be deterministic. We show that these requirements can be conceptually separated, by constructing an ATOB protocol which tolerates $t < n/2$ corruptions from blackbox use of Common Coin and Reliable Broadcast. We show the power of this conceptually simple contribution by reproving, using simpler protocols, existing results on STOB with optimistic responsiveness and asynchronous fallback. We also use the framework to prove the first ATOB with sub-quadratic communication and optimal corruption threshold $t < n/3$, new ATOBs with covert security and mixed adversary structures, and a new STOB with asymmetric synchrony assumptions.

1 Introduction

MPC in asynchronous networks is complicated by the fact that a protocol can never wait to receive inputs from all parties in the presence of faults. [BCG93,Can95] introduce a model for Asynchronous MPC (AMPC) in which the ideal functionality is a trusted party who receives only the inputs from a *core set* of the parties and returns a description of the core set and the result of the computation using only these inputs. The core set is chosen by an adversarial scheduler but is restricted to contain at least $n - t$ parties chosen independently of any inputs of honest parties. To realise this functionality they provide an Agreement on a Core Set (ACS) protocol which is resilient against $t < n/3$ corruptions and allows the parties to agree on a set of size at least $n - t$. In more detail each party at the beginning of the protocol has a set of size $n - t$ which they send to all other parties. It keeps adding new elements to the set as they arrive and in each round of the protocol waits until $n - t$ of the sets from the previous round are included in its currently accumulated set, then sends its current set. After $\log n$ rounds it can be shown that all the accumulated sets share a common core of size $n - t$. The proof goes by induction in the number of rounds and crucially relies on the fact that in each round each party takes the union of sets from $n - t = 2t + 1$ parties to conclude that each pair of honest parties will receive at least one set from some common honest party P, and hence their set in round r will include the set sent by P in round $r - 1$. Finally each party runs a Byzantine Agreement (BA) protocol for each party with input 1 iff that party is in its accumulated set.

Several works stop just before running the BAs and instead follow the pattern of rounds of all-to-all communication while accumulating local sets with a large common core as a building block to achieve consensus. Following [AJM+21] we will refer to this design pattern as *Gather*. One notable recent example using Gather is the DAG-Rider protocol in [KKNS21], in which they build

* Simon Holmgaard Kamp and Jesper Buus Nielsen are partially funded by The Concordium Foundation.

a DAG of blocks pointing to $n-t$ blocks from the previous round, and for each 4 rounds, called a *wave*, rely on the fact that causal past of any $n-t$ blocks in the last round include $n-t$ common blocks from the first round. This allows them to conclude that a block from the first round of the wave which is randomly chosen after the wave is completed will be in the DAG of an honest party with probability at least 2/3. The reason they get away with only 4 rounds of blocks in each wave and still get a large common core, is that all blocks are sent through Reliable Broadcast[Bra87] (RB). This means that each pair of blocks will have overlap on least $n-2t \geq t+1$ of the blocks referenced from the previous round. This gives rise to an ATOB protocol by running multiple waves in sequence, flipping a common coin for each iteration and applying an appropriate commit rule.

In this paper we present an asynchronous Gather protocol tolerating $t < n/2$ corruptions assuming RB. On a high level it uses the structure of the Gather in [KKNS21] and the round complexity of [BCG93,Can95] to tolerate more corruptions and a termination condition from the Gather protocol CSS in [MMNT19,DMM+20] to get constant time complexity. Interestingly, even though the protocol follows [BCG93,Can95] and runs in $\log(n)$ rounds of communication it will terminate in time $\mathcal{O}(\Delta_{\text{NET}})$, where Δ_{NET} is the actual network delivery time during the execution. It uses the RB blackbox, so one can instantiate RB in a setting where assumption A holds to get Gather for the setting where $t < n/2$ and A holds. Using a generic transformation similar to [KKNS21] this gives a Total Order Broadcast (TOB) protocol for the same setting. We note that additional work is needed to get ATOB from Gather compared to previous works as all sub-protocols must tolerate $t < n/2$ in the asynchronous model. Typically, if A implies RB it also implies $t < n/2$, so overall we get that if we can construct RB from assumption A then we can also get TOB from A. Our compiler is efficient, so this allows to construct efficient TOB in various models. In summary we get the following results:

1. In the asynchronous setting in which we assume a ground population of size N with $T < N/3$ corruptions and a committee of n servers with honest majority we construct a protocol for RB which has communication linear in N. Using this and subset sampling to elect n servers from the ground population we get the first ATOB protocol with subquadratic communication and *optimal* resilience.
2. In the asynchronous setting we construct a RB that is covert secure against $t < n$ corruptions. This gives an ATOB protocol with covert security against $t < n/2$ corruptions.
3. We introduce a new weakly synchronous model, called the *asymmetric synchrony assumption* (ASA) model, where each party P_i has its own guess Δ^i_{GUESS} at the network delay. The guess is unknown to the other parties. The only guarantee is that the actual network delay for messages sent specifically to P_i is smaller than Δ^i_{GUESS}. A motivation for the model is *ad hoc* groups of parties which want to do TOB and might not share the same assumptions on the network, or know each other's assumptions prior to running the protocol. We give an RB for the ASA model tolerating $t < n/2$ corruptions. We also consider the weakly ASA (WASA) model, where all honest guesses at the network delay are sound for all messages sent between all honest parties.
4. We introduce the *timeout model*, which uses time weakly. Parties do not have clocks, their only access to time is that they can set a timeout and get a callback *at least* some Δ_{WAIT} seconds later. They are also not guaranteed to start the protocol at the same time, which makes it easier to compose the protocol with other protocols. This notion of synchrony is seemingly much easier to implement in practice as it does not need synchronised clocks nor very precise clocks, merely a bound on how fast the clock drift. We give an RB for the timeout model tolerating $t < n/2$ corruptions.

5. In the synchronous setting we instantiate an RB which is optimistically responsive, i.e., when the actual corruption is lower than the maximal tolerated corruption then the latency of the protocol only depends on the current network delay. Our protocol tolerates $t \leq t_{\mathrm{S}}$ corruptions and and is optimistic for $t \leq t_{\mathrm{A}} (\leq t_{\mathrm{S}})$ corruptions. It tolerates $t_{\mathrm{S}} + 2t_{\mathrm{A}} < n$ which is optimal. This gives an optimistic responsive TOB in which the optimistic condition does not include having an honest leader. The protocol is secure in the WASA model and the timeout model. To the best of our knowledge the resulting TOB is the first optimally resilient fallback TOB for the WASA model and for the timeout model.
6. We give a RB for the ASA model with asynchronous fallback which is secure for $2t_{\mathrm{S}} + t_{\mathrm{A}} < n$ and $t_{\mathrm{A}} \leq t_{\mathrm{S}}$ and when *either* there are at most $t \leq t_{\mathrm{S}}$ corruptions and the network is ASA synchronous *or* there are at most $t \leq t_{\mathrm{A}}$ corruptions and the network is fully asynchronous. To the best of our knowledge the resulting TOB is the first optimally resilient fallback TOB for the ASA model and for the timeout model. Setting $t_{\mathrm{A}} = 0$ gives the contributions claimed in Item 3 and Item 4.
7. We finally note that it is an easy corollary of our framework that you can get asynchronous TOB for a model with mixed adversaries, where there are at most $t_{\textsc{Byz}}$ fully Byzantine corruptions, at most $t_{\textsc{Crash}}$ additional crash-silent errors, and $2t_{\textsc{Crash}} + 3t_{\textsc{Byz}} < n$.

The above results show the power of a framework where ATOB is implemented with *honest majority* given RB, which can then be implemented in different ways.

1.1 Related work

We discuss related work in the context of the specific contributions.

Subquadratic ATOB with Optimal Resilience A noteworthy property of asynchronous protocols tolerating significantly more than a third corruption, is that one can securely sample a committee to perform the task while retaining the optimal resilience against $t < n/3$ corruptions. This was previously used in [ACKN23]. Previous solutions to subquadratic BA in the asynchronous [BKLL20,CKS20] or partially synchronous [GHM+17] setting rely on sampling committees with an honest supermajority from a ground population with $t < (1 - \epsilon)n/3$ corruptions. As ϵ can be arbitrarily small, the separation between optimally and "near-optimally" resilient protocols might seem purely theoretical. But the practical implications of picking a small ϵ is that any secure protocol must have a concretely very large committee. If for instance $\epsilon = 1/10$ (i.e. the corruption in the ground population is bounded by 30%) committees need 16037 parties to retain an honest supermajority except with probability 2^{-60} (cf. table 1 of [DMM+22]). A recent work [BBK+23] on the concrete security of Algorand show that a committee of 6000 parties is needed to get 56 bits of security assuming the parties are sampled from a ground population with less than 1/5 corruption. When assuming the same corruption in the ground population our protocol would need a committee of 173 parties to get 60 bits of security, while it would need 653 parties to tolerate up to a third being corrupted (cf. Table 1).

To instantiate our protocol we need to use a RB, which usually would have all-to-all communication in a committee with an honest supermajority. However, we construct a RB protocol which only depends linearly on the size of the ground population that needs honest supermajority while the remaining communication is done in a committee with honest majority. As a rough sketch: a sender sends a message to the ground population who threshold signs it and sends their shares

back to the sender who then combines the shares and sends it to the committee that will gossip it among themselves. Our actual implementation is a bit more involved in order to get not only subquadratic but concretely very efficient communication which can be amortized optimal when either the messages or the ground population is large. This approach is not secure against an adaptive adversary and it is an open problem if one can get an adaptively secure subquadratic asynchronous agreement with optimal resilience.

The DAG rider protocol [KKNS21] can achieve amortized communication overhead of $\mathcal{O}(n)$ when blocks include $\mathcal{O}(n \log n)$ values, i.e., $\mathcal{O}(n \log n \kappa)$ bits. This is done by instantiating it with the RB protocol by Cachin and Tessaro [CT05] which communicates $\mathcal{O}(n|m| + n^2 \log n \kappa)$ bits to broadcast a message of length $|m|$. An important caveat of the amortized complexity of [KKNS21] is that it only goes through in their model because there is no distinction between clients and servers. This means that each value sent from a server counts as throughput. If the protocol would be deployed in a setting in which clients send their messages to servers who broadcast them, then communication complexity increases by a factor n, as all servers could in the worst case include the same messages in each round. Our RB protocols uses similar techniques as in [CT05] to get the communication cost of broadcasting β bits from and to the ground population down to the amortized optimal $\mathcal{O}(N\beta)$ when the protocol consumes on average at least $n \log(n)$ messages per epoch and *either* the size of the ground population is at least $n^2 \log n$ or the average messages size is at least $n \log(n)\kappa$. The time complexity is expected $\mathcal{O}(1)$ network delays as in [KKNS21].

Honest Majority Asynchronous Total-Order Broadcast with Covert Security [AL07] introduces a notion of *covert adversaries* in which some parties are Byzantine corrupted but do not want to be "caught". Protocols are said to be covert secure if any breach of security will result in the honest parties detecting a specific party who did not follow the protocol. [AO12] introduce a strengthened model in which cheating parties are not only detected by an honest party with some probability, but that party also receives a certificate demonstrating that a party cheated. We informally prove that we can instantiate asynchronous RB with dishonest majority and covert security with public verifiability, which in turn gives ATOB for honest majority secure against covert adversaries. The idea is simple: senders sign their messages which are then flooded. If different messages were signed, they will eventually meet and the sender is detected as corrupted.

Optimistic Responsiveness Optimistic responsive protocols [PS17] have safety in synchronous networks when the number of corruptions is bounded by t_{S} but additionally a latency that only depends on the actual (unknown) network delay when at most $t_{\text{A}} < t_{\text{S}}$ parties are corrupted. In some cases additional optimistic conditions, e.g., a leader being honest, must be met. We provide an optimistic responsive RB protocol matching the optimal bounds for t_{A} and t_{S}, which in turn gives an optimistic responsive TOB protocol. As the resulting protocol only retroactively elects a leader, and any valid leader points to a core set of $n - t$ inputs, the resulting protocol does not impose any restrictions on the optimistic case besides having at most t_{A} corruptions. The paper [HKL20] considers asynchronous RB protocols with different corruption levels for the different security properties validity, agreement and termination. This is related, but different as we consider synchronous protocols and different running times for different thresholds.

Synchronous Consensus with Asynchronous Fallback The concept of synchronous protocols tolerating t_{S} corruptions and additionally tolerating asynchrony when $t_{\text{A}} < t_{\text{S}}$ parties are corrupted were introduced in [BKL19]. It turns out that the protocol used to achieve optimistic responsiveness has

synchronous security with asynchronous fallback when adjusting the timeout used and instantiating it with the optimal bounds for t_{A} and t_{S} in this model. Once more this immediately gives a TOB protocol with optimal thresholds in this model.

Asymmetric Synchrony Assumptions To the best of our knowledge it is new that one can do TOB for $t < n/2$ in the ASA model. The closest related work seems to be [DDFN07] where the authors consider asymmetric trust in MPC. Each party has its own assumptions on which subsets of the parties may be corrupted. However, [DDFN07] considers a fully synchronous network proceeding in rounds, i.e., all parties share assumptions on the network delay, start the protocol at the same time and have access to clocks to implement a round-based abstraction. Here we relax the model further and assume that the parties might not even share assumptions on the network delay. We show how to get TOB in this model. Our protocol has the interesting property that it is responsive in the largest *honest* Δ^i_{Guess}, i.e., the corrupted parties cannot do a denial of service attack by proposing artificially large Δ^i_{Guess}. Asymmetric trust in distributed systems and specifically total-order broadcast was also studied in [CT19,CZ21,Cac21], but again only the assumptions on corruptions are asymmetric, not assumptions on network delays.

Mixed Adversaries Mixed adversaries were studied in consensus and MPC before. In [GP92,MP91] the authors give a consensus protocols for $t_{\text{Crash}} + 3t_{\text{Byz}} < n$, but for the synchronous model. In [FHM98] the authors give information theoretic MPC protocols which tolerates mixed adversaries. For the asynchronous model $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$ is trivially optimal. An honest party can only wait to hear from $n - t$ parties without deadlocking, where $t = t_{\text{Byz}} + t_{\text{Crash}}$. If $2t_{\text{Crash}} + 3t_{\text{Byz}} = n$ then two honest parties each hearing from $n - t$ parties may only have an overlap of $n - 2t = t_{\text{Byz}}$. Therefore honest parties can be partitioned and never receive information from each other and still have to give an output. We are not aware of any concrete ATOB protocol for a mixed model with $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$. We do, however, not claim that the feasibility is new. We are highlighting the construction to show how easy it is to derive an ATOB for $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$ in our framework. Getting a RB from $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$ is trivial, see Section 6.3, and the rest of the construction does not consider mixed adversaries at all and works for $t < n/2$, i.e., $2t_{\text{Crash}} + 2t_{\text{Byz}} < n$.

Subquadratic AMPC with Optimal Resilience The final result we present is the fairly simple observation that using our subquadratic ATOB protocol as the broadcast channel in the AMPC protocol in [Coh16] which only requires honest majority assuming such a broadcast channel, we get AMPC which can be run by the honest majority committee and hence has subquadratic complexity with optimal resilience. It has previously been shown that AMPC can be linear in the circuit size[CP15]. [CHL21] focuses on adaptive security and give a protocol with $\mathcal{O}(n\,\text{poly}(\kappa))$ communication and near-optimal resilience. But using our approach the communication complexity can be sublinear in the size of the inputs and subquadratic in the size of the output for computations on many inputs. Specifically if at least $n^2 \log n$ members of the ground population give input of combined size β, and the output has size γ then the communication complexity is $\mathcal{O}(n\beta + nN\gamma)$.

2 Network Model and Technical Preliminaries

We use κ to denote the security parameter. We use σ to denote a statistical security parameter, i.e., for all fixed σ it holds that the security of a protocol is $2^{-\sigma} + \text{negl}(\kappa)$ as κ goes to ∞. In the

main analysis we assume that $\sigma = \Theta(\kappa)$ and only use κ. We revisit the distinction between the parameters in Section 4 when discussing concrete parameters.

We consider protocols for a fixed set of parties $\mathbb{G} = \{\mathsf{G}_1, \ldots, \mathsf{G}_N\}$, which we call the *ground population*. We assume that at most $T < N/3$ or these are corrupted. We also consider a small set of parties $\mathbb{S} = \{\mathsf{S}_1, \ldots, \mathsf{S}_n\}$, which we call the servers. For notational simplicity we assume that $\mathsf{S}_i = \mathsf{G}_i$. We assume that at most $t < n/2$ or these are corrupted. We offer the reader the intuition that $\mathbb{S}$ has been sampled from the set $\mathbb{G}$ large enough that $T < N/3$ implies that $t < n/2$, except with negligible probability. We therefore assume that $n = \mathcal{O}(\sigma)$. For simplicity we also assume that $n = \Theta(\kappa)$ as it allows a more concise analysis in places. In Section 4 we discuss how to set n to ensure $t < n/2$ except with probability $2^{-\sigma}$ for different security levels σ.

We assume that the parties all have pairwise authenticated channels. We assume an asynchronous communication model where the adversary schedules the delivery of the messages without any restrictions. We use a simple model of eventuality. We say that event E (like termination of a protocol) *eventually* happens (in a protocol with session identifier sid) if it holds that at any point in time if the event E did not happen for session sid, then there is still a message in transit from an honest party to an honest party with session identifier sid. Note that a protocol could hack this definition by having two honest parties P_1 and P_2 sending back and forth a PING message forever. Then by definition all events eventually happen. However, all our protocols generate an expected finite number of messages and the simple notion of *eventually* is meaningful for such protocols.

We also consider synchronous protocols. We use the above system model but assume a global clock $c \in \mathbb{N}$ incremented by the adversary. We say that the network is Δ_{NET}-synchronous if all message sent by time c_0 by an honest party to a honest party is delivered at time $c_1 \leq c_0 + \Delta_{\text{NET}}$. We do not give the parties access to the clock. Instead, when describing synchronous protocols we will use time via an explicit timeout mechanism. A party can create a timeout of some duration Δ. We say that the party calls $\mathsf{Timeout}(\mathsf{name}, \Delta)$. It is then guaranteed that the party at some point in time at least Δ time units after the call $\mathsf{Timeout}(\mathsf{name}, \Delta)$ will be activated with input name. It is not guaranteed to happen after exactly duration Δ. We do not assume that parties in a protocol start at the same time. We call this the *timeout model*.

We let $\text{Lg}(n) = \lceil \log_2(n) \rceil$. We use an erasure code $\mathsf{EC} = (\mathsf{Enc}, \mathsf{Dec})$. The encoding of $m \in \{0,1\}^{(t+1)\,\text{Lg}(n)}$ as $(m_1, \ldots, m_n)$ proceeds as follows. Encode m as $(\alpha_0, \alpha_1, \ldots, \alpha_t) \in \mathsf{GF}(2^{\text{Lg}(n)})^{t+1}$, let $f(\mathtt{x}) = \sum_{i=0}^{t} \alpha_i \mathtt{x}^i$, and let $m_j = f(j)$ for $j = 1, \ldots, n$. We can decode from $t+1$ values (j, m_j) by interpolating $f(\mathtt{x})$ and reading off $(\alpha_0, \alpha_1, \ldots, \alpha_t)$. When $n = 2t+1$ then clearly $|m_j| \leq |m|/n$.

We assume a collision resistant hash function Hash. Among other uses we use it to compute a Merkle-tree $h = \mathsf{HashTree}(m_1, \ldots, m_n)$ for messages m_i. In this context we use $\mathsf{Path}_i = \mathsf{Path}(m_1, \ldots, m_n, i)$ to denote the usual path used to prove that m_i is consistent with being message number i in the tree with root h. We let $\mathsf{VerPath}$ be the algorithm verifying that Path_i proves that m_i is in h.

We assume a signature scheme $(\mathsf{Gen}, \mathsf{Sig}, \mathsf{Ver})$ which is EUF-CMA secure [GMR88] and we assume a PKI. In some initial synchronous round all parties $\mathsf{G}_i \in \mathbb{G}$ sample $(\mathsf{vk}_i, \mathsf{sk}_i) \leftarrow \mathsf{Gen}(1^\kappa)$, sends vk_i to a trusted third party which makes public $(\mathsf{vk}_1, \ldots, \mathsf{vk}_N)$. The adversary gets to see vk_i for all honest G_i before picking its own keys, and it does not have to pick its own keys at random, it can use any vk_j for a corrupted G_j.

All our definitions and protocols tacitly assumes that a session identifier sid is given to distinguish different runs of a protocol. When signing a message in a session with session identifier sid we tacitly add sid to the signed message to avoid replay attacks.

We assume some common setup among $\mathbb{G}$. Initial some PPT algorithm $(\mathsf{pv}, \mathsf{sv}_1, \ldots, \mathsf{sv}_N) \leftarrow \mathsf{Setup}(1^\kappa)$ is run and $(\mathsf{pv}, \mathsf{sv}_i)$ is given to G_i. The secret values can be correlated, for instance a sharing of a secret key. We do not assume special setup among the servers $\mathbb{S}$. The reason is that we think of $\mathbb{S}$ as having been sampled from $\mathbb{G}$, possibly on the fly. Therefore any specialized setup for $\mathbb{S}$ would reasonably have to be computed on the fly too, and we do not want to hide the cost of this.

We use a threshold signature scheme with unique signatures $(\mathsf{Setup}, \mathsf{Sig}, \mathsf{Ver}, \mathsf{Combine})$, where $(\mathsf{vk}, \mathsf{sk}_1, \ldots, \mathsf{sk}_N) \leftarrow \mathsf{Setup}(1^\kappa)$ generates a verification key vk and a signing share sk_i for $\mathbb{G}_i$, $\mathsf{Sig}_{\mathsf{sk}_i}$ partially signs m, Ver can verify a partial signature, and $\mathsf{Combine}$ computes $\sigma = \mathsf{Sig}_{\mathsf{sk}}(m)$ from $T+1$ verified shares, where $T < N/3$. Given only T of the signing keys sk_i the scheme is still EUF-CMA. A detailed definition can be found in for instance [CKS00].

For one result we assume a threshold fully homomorphic encryption scheme $(\mathsf{Setup}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec}, \mathsf{Ver}, \mathsf{Combine})$, where $(\mathsf{ek}, \mathsf{dk}_1, \ldots, \mathsf{dk}_N) \leftarrow \mathsf{Setup}(1^\kappa)$ generates en encryption key ek and a decryption share dk_i for $\mathbb{G}_i$, $\mathsf{Enc}_{\mathsf{ek}}$ decrypts, $\mathsf{Eval}_{\mathsf{ek}}(f, \cdot)$ applies f to the messages in ciphertexts, $\mathsf{Dec}_{\mathsf{dk}_j}(c) = y_j$ produces a partial decryption, Ver verifies a decryption share, and $\mathsf{Combine}$ compute $\mathsf{Dec}_{\mathsf{dk}}(c)$ from $T+1$ verified shares, where $T < N/3$. Given only T decryption keys the scheme is IND-CPA. A detailed security definition can be found in [Coh16].

2.1 Eventual Justifiers

In our protocols all messages will have a message identifier mid specifying which protocol it belongs to, what round of the protocol it comes form, sent by whom and so on. Each message identifier mid specifies a party $\mathsf{P}^{\mathsf{mid}}$, which we think of as the party which is to send the message identified by mid. Here P is either a server S or a ground member G. Each mid also specifies a so-called justifier J^{mid}, which is a predicate depending on the message m and the local state of a party. When we write pseudo-code then we write $J^{\mathsf{mid}}(m)$ to denote that the party P executing the code computes J^{mid} on m using its current state. In definitions and proofs we write $J^{\mathsf{mid}}(m, \mathsf{P}, t)$ to denote that we apply J^{mid} to m and the local state of P at time t. The following definition is adopted from a similar definition in [DMM+20].

Definition 1 (Justifier). *For a message identifier* mid *we say that* J^{mid} *is a justifier if the following properties hold.*

Monotone: *If for an honest* P *and some time* t *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, t) = \top$ *then at all* $t' \geq t$ *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, t') = \top$.

Propagating: *If for honest* P *and some point in time* t *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, t) = \top$, *then eventually the execution will reach a time* t' *such that* $J^{\mathsf{mid}}(m, \mathsf{P}', t') = \top$ *for all honest parties* P'.

Definition 2 (Justified Protocol). *We will work with justified protocols* Π. *If a protocol has an input justifier* J_{IN} *it means that a message identifier* mid *is associated with the input,* $J_{IN} = J^{\mathsf{mid}}$, *and it is* $\mathsf{P}^{\mathsf{mid}}$ *which gets the input. Furthermore, if* $\mathsf{P}^{\mathsf{mid}}$ *is honest then it is guaranteed that* $J_{IN}(m) = \top$ *whenever* m *is input to* $\mathsf{P}^{\mathsf{mid}}$. *If a protocol has an output justifier* J_{OUT} *it means that a message identifier* mid *is associated with the output,* $J_{OUT} = J^{\mathsf{mid}}$, *it is* $\mathsf{P}^{\mathsf{mid}}$ *which gives the output, and when it gives the output it sends it to all parties with message identifier* mid. *Furthermore, if* $\mathsf{P}^{\mathsf{mid}}$ *is honest then it is a security property of the protocol that* $J_{OUT}(m) = \top$ *whenever* m *is output by* $\mathsf{P}^{\mathsf{mid}}$.

We will sometimes talk about a property holding for all *possible justified outputs* of a justified protocol, by which we mean that it holds also for adversarial outputs, as long as these are justified. These outputs might not have been the output of any party in the protocol, the adversary is free to cook them up. However, all outputs will be sent, and an output is only considered an output if it meets J_{OUT}. Therefore we can identify adversarial outputs by messages that honest parties would accept as justified.

Definition 3 (Possible Justified Outputs). *Let Π be a protocol with output justifier J. When we say that an ℓ-ary predicate P holds* for all possible justified outputs *we mean: Run the protocol Π under attack by the adversary. At some point the adversary may output a sequence $(\mathsf{P}^1, \mathsf{mid}^1, m^1), \ldots, (\mathsf{P}^\ell, \mathsf{mid}^\ell, m^\ell)$. We say that the adversary wins if $\mathsf{mid}^1, \ldots, \mathsf{mid}^\ell$ are identified with outputs of Π, $\mathsf{P}^1, \ldots, \mathsf{P}^\ell$ are honest, for $j = 1, \ldots, \ell$ it holds that $J^{\mathsf{mid}^j}(m^j) = \top$ at P^j, and $P(m^1, \ldots, m^\ell) = \bot$. Otherwise the adversary looses the game. Any PPT adversary should win with negligible probability.*

Note that since the outputs of a justified protocol are sent to all parties they will become known by the adversary, and we assume the output m_i of all honest parties P_i are justified, i.e., $J^{\mathsf{mid}_i}(m_i) = \top$ at $\mathsf{P}^{\mathsf{mid}_i}$ at the time of output, we have as a special case that $P(m^1, \ldots, m^\ell) = \top$ for all honest outputs $m^1, \ldots, m^\ell$, as the adversary can let $\mathsf{P}_j = \mathsf{P}^{\mathsf{mid}_j}$ in the game. However, *for all possible justified outputs* implies in addition that the adversary cannot even cook up outputs which look justified to some honest parties and which does not have the property P.

2.2 Justified Reliable Subcast

We use the notion of reliable broadcast (RB) (cf. [Bra87]). For a message identifier mid we can force a possible corrupt sender G_s to send a message m associated to mid such that all servers which receive m will receive the same m. Furthermore, if any honest server receives m then they all receive m. We look at a version where all parties in $\mathbb{G}$ can broadcast but only the subset of parties $\mathbb{S} \subseteq \mathbb{G}$ learn the output, so we call it a reliable subcast. Allowing only $\mathbb{S}$ to broadcast gives a normal RB among $\mathbb{S}$. Setting $\mathbb{S} = \mathbb{G}$ gives a normal reliable broadcast among $\mathbb{G}$.

Definition 4 (Justified Reliable Subcast). *A protocol Π for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ and N ground members $\mathsf{G}_1, \ldots, \mathsf{G}_N$, where all parties have input mid. The message identifier mid contains the identity of a sender G_s along with the description of a justifier J_{mid}. The sender additionally has input $m \in \{0,1\}^*$ for which $J_{\mathsf{mid}}(m) = \top$ at G_s at the time of input.*

Validity: *If honest G_s has input (mid, m) and an honest S_i has output (mid, m') then $m' = m$.*
Agreement: *For all possible justified outputs (mid, m) and (mid, m') it holds that $m = m'$.*
Eventual Output 1: *If G_s is honest and has input $(\mathsf{mid}, \cdot)$, and all honest G_j start running the protocol, then eventually all honest S_i have output $(\mathsf{mid}, \cdot)$.*
Eventual Output 2: *If an honest S_j has output $(\mathsf{mid}, \cdot)$, and all honest servers start running the protocol then eventually all honest S_i have output $(\mathsf{mid}, \cdot)$.*

If only servers S_i are allowed to give inputs then we call Π a RB (for the servers).

We will later give many different implementations of reliable subcast. We will also show how to get total-order broadcast among $\mathbb{S}$ from reliable subcast secure against $t < n/2$ servers. We will also need that the servers can reliably broadcast a message "out" to the ground population. In this case we assume that all servers agree on the message m to be sent. We call this primitive outcast.

Definition 5 (Reliable Outcast). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ and N ground members $\mathsf{G}_1, \ldots, \mathsf{G}_N$, where all parties have input* mid*. The servers additionally have an already agreed upon input $m \in \{0,1\}^*$.*

Validity: *If all honest S_i have input (mid, m) and an honest G_i has output (mid, m') then $m' = m$.*
Eventual Output: *If all honest S_i have input (mid, m) and start running the protocol, then eventually all honest G_i which start running the protocol have an output $(\mathsf{mid}, \cdot)$.*

Outcasting is trivial when at most $t < n/2$ servers are corrupt. First each server takes m and uses an erasure code to encode it as $(m_1, \ldots, m_n)$ such that it can be decoded using $n - t$ of the values m_i. Using standard techniques and assuming $|m| \geq n \log n$, this can be done with $\sum_{i=1}^{n} |m_i| = \mathcal{O}(|m|)$. Then each server computes a Merkle-Damgård hash tree on $(m_1, \ldots, m_n)$ with root h. Server S_i sends to each G_j the root h and m_i and a path h_i showing that m_i is in leaf i of the tree with root h. Each ground member G_j waits for $n - t$ identical reports of h and adopts this value. It was sent by at least one honest server, so it must be correct. Then it waits for $n - t$ messages m_i along with proofs that they are in h. From these $n - t$ values it computes m. We call this protocol Π_{OUTCAST}. This costs communication $\mathcal{O}(nN \log(n)\kappa + N|m|)$. For $|m| \geq n \log(n)\kappa$ and $n = \Theta(\kappa)$ this is $\mathcal{O}(N|m|)$, which is asymptotically optimal as each ground member has to receive m. More details are given in Section 4.

2.3 Justified Leader Election

In the following definitions we will often drop the explicit mentioning of message identifiers. They are tacitly identified by variable names and adding explicit message identifiers gives no more insight. In addition, in each of the following protocols each party has an additional input sid, the session identifier, allowing to identify different runs. We often let it be tacit. All security properties are defined relative to the same sid and all protocol messages tacitly contains sid.

Definition 6 (Justified Leader Election). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ where the input of the parties is a fixed symbol* ELECT*. The output is $j \in [n]$. There is an output justifier J_{OUT} specified by the protocol.*

Liveness: *If all honest servers start running the protocol with input* ELECT*, then eventually all honest parties S_i have an output.*
Validity: *For all possible justified outputs j it holds that $j \in \{1, \ldots, n\}$.*
Agreement: *For all possible justified outputs j and j' it holds that $j = j'$.*
Unpredictable: *If no honest party had input* ELECT *yet, then the adversary cannot guess j negligibly better than at random. In particular, consider the game where the adversary can run the protocol and at any point in time where no honest party has input* ELECT *yet—and at most once—can output j'. It wins the game if it can continue the execution of the protocol and make an honest party output j'. No PPT adversary should win this game with probability better than $1/n + \text{negl}$.*

We note that the leader election protocol from [CKS00] works for the asynchronous model with $t < n/2$ corruptions if a threshold signature scheme with unique signatures has been set up among the servers. On $(\text{ELECT}, \mathsf{sid})$ server S_i sends its signature share on sid. Given $n - t$ signature shares a server reconstructs $\sigma_{\mathsf{sid}} = \mathsf{Sig}_{\mathsf{sk}}(\mathsf{sid})$, computes $c_{\mathsf{sid}} = \mathsf{Hash}(\sigma_{\mathsf{sid}})$. Modelling Hash as a random

oracle this gives a uniformly random c_{sid} unknown until σ_{sid} is known, in particular until the first honest parties gets input ELECT. This costs communication $\mathcal{O}(n^2\kappa)$.

In a setting where we do not want to assume special setup among the servers we can share sk among the ground population with reconstruction threshold $T+1$. On input $(\text{ELECT}, \mathsf{sid})$ a server S_i sends an authenticated $(\text{ELECT}, \mathsf{sid})$ to the ground population. On seeing $t+1$ such messages a ground member sends its signature share to each of the n servers. Given $T+1$ of the signature shares the server can compute the coin as above. This costs communication $\mathcal{O}(Nn\kappa)$. Below we let $\text{ELECT} = Nn\kappa$.

2.4 Causal Cast

We now present a framework for describing protocols for DAG-style protocols (cf. [KKNS21]) in a modular way and in combination with non-DAG style protocols. In a DAG-style protocol all messages m are reliably broadcast and they point to the other reliably broadcast messages they were computed from. Therefore the receiver can recompute and check the message m. In fact, the message m never has to be sent, the receiver can compute it itself. This allows to compress the communication complexity. All that needs to be reliably broadcast are the pointers to the messages used to compute m. This implements reliable, causal communication against a Byzantine adversary. We will therefore call our system below causal cast (CauCast). Each message m to be CauCast will have a message identifier mid. The set of message identifiers is divided into four disjoint sets of free-choice identifiers, computed-message identifiers, leader-election identifiers, and constant identifiers. We assume that the type of mid can be determined efficiently. Each free-choice, computed message, and constant identifier mid specifies a sender $\mathsf{S}^{\mathsf{mid}}$. Each free-choice identifier mid specifies an input justifier $J_{\text{IN}}^{\mathsf{mid}}$. Each computed message identifier specifies a next message function $\mathsf{NextMessage}^{\mathsf{mid}}$. This function is PPT and takes as input a set of pairs $M = \{(\mathsf{mid}_j, m_j)\}_{j=1}^{\ell}$ and outputs $m = \mathsf{NextMessage}^{\mathsf{mid}}(M)$, where $m = \perp$ indicates that M is not a valid set of inputs for computing the message for mid. For leader-election identifiers mid we have that $\mathsf{mid} = \mathsf{sid}$ for a session identifier sid for a justified leader-election protocol Π_{ELECT} with output justifier $\Pi_{\text{ELECT}}.J_{\text{OUT}}$. The constant identifiers are just a convenient tool to define that some values on which the servers already agree have been "delivered", for instance hardwired values of the protocol.

The system guarantees liveness, agreement on all messages, and that all messages are valid inputs, valid outputs of a leader election, or computed correctly from other valid messages.

Definition 7 (Causal Cast). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ is called a causal cast (CauCast) if it has the following properties.*

Free-Choice Send: *A server S_i can have input $(\textit{CAUCAST-SEND}, \mathsf{mid}, m)$ where mid is a free choice identifier $\mathsf{S}_i = \mathsf{S}^{\mathsf{mid}}$ and $J_{IN}^{\mathsf{mid}}(m) = \top$ at $\mathsf{S}^{\mathsf{mid}}$ at the time of input.*

Computed-Message Send: *A server S_i can have input $(\textit{CAUCAST-SEND}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$, where mid is a computed-message identifier, $\mathsf{S}_i = \mathsf{S}^{\mathsf{mid}}$, S_i earlier gave outputs $(\textit{CAUCAST-DEL}, \mathsf{mid}_j, m_j)$ for $j = 1, \ldots, \ell$, and $\perp \neq m = \mathsf{NextMessage}^{\mathsf{mid}}((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$.*

Constant Send: *A server S_i can have input $(\textit{CAUCAST-SEND}, \mathsf{mid}, m)$ where mid is a constant identifier. In that case it is guaranteed that all servers eventually have the same input $(\textit{CAUCAST-SEND}, \mathsf{mid}, m)$.*

Free-Choice Validity: *A server* S_i *can have output* $(\textsc{caucast-del}, \mathsf{mid}, m)$*, where* mid *is a free-choice identifier. It then holds that* $J_{IN}^{\mathsf{mid}}(m) = \top$ *at* S_i *at the time of output. Furthermore, if* $\mathsf{S}_j = \mathsf{S}^{\mathsf{mid}}$ *is honest, then* S_j *had input* $(\textsc{caucast-send}, \mathsf{mid}, m)$*.*

Leader Election Validity: *A server* S_i *may output* $(\textsc{caucast-del}, \mathsf{mid}, m)$ *where* mid *is a leader-election identifier. In that case* $\mathsf{mid} = \mathsf{sid}$ *is a session identifier for a justified leader election and* $\Pi_{\textsc{Elect}}.J_{OUT}^{\mathsf{sid}}(m)$ *at* S_i *at the time of output.*

Computed-Message Validity: *A server* S_i *can have output* $(\textsc{caucast-del}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$*, where* mid *is a computed-message identifier. In that case* S_i *earlier gave outputs* $(\textsc{caucast-del}, \mathsf{mid}_j, m_j, \ldots)$ *for* $j = 1, \ldots, \ell$*, and* $\bot \neq m = \mathsf{NextMessage}^{\mathsf{mid}}((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$*.*

Constant Validity: *A server* S_i *can have output* $(\textsc{caucast-del}, \mathsf{mid}, m)$*. In that case it immediately before had input* $(\textsc{caucast-send}, \mathsf{mid}, m)$*.*

Liveness: *If an honest server* S_i *had input* $(\textsc{caucast-send}, \mathsf{mid}, \ldots)$ *or some honest server had output* $(\textsc{caucast-del}, \mathsf{mid}, \ldots)$ *and all honest servers are running the system, then eventually all honest servers have output* $(\textsc{caucast-del}, \mathsf{mid}, \ldots)$*.*

Agreement: *For all possible justified outputs* $(\textsc{caucast-del}, \mathsf{mid}, m, \ldots)$ *and* $(\textsc{caucast-del}, \mathsf{mid}, m', \ldots)$ *it holds that* $m' = m$*.*

Below is a protocol implementing CauCast. It uses a sub-protocol for reliable broadcast (RB) among the servers and a sub-protocol $\Pi_{\textsc{Elect}}$ for justified leader election. The protocol $\Pi_{\textsc{Elect}}$ may be initialised by other protocols. The CauCast merely reports it outputs—this allows the elections to be inputs for computed messages in the causal cast system.

Free-Choice Send: On input $(\textsc{caucast-send}, \mathsf{mid}, m)$ at S_i where mid is a free choice S_i will RB m with message identifier mid.

Computed-Message Send: On input $(\textsc{caucast-send}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ at S_i, where mid is a computed-message identifier, S_i will RB $(\mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ with message identifier mid.

Free-Choice Deliver: On RB m from with message identifier mid from $\mathsf{S}^{\mathsf{mid}}$ where mid is a free-choice identifier, wait until $J_{\text{IN}}^{\mathsf{mid}}(m) = \top$ (possibly waiting forever if this never happens) and then output $(\textsc{caucast-del}, \mathsf{mid}, m)$.

Computed-Message Deliver: On RB $(\mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ with message identifier mid from $\mathsf{S}^{\mathsf{mid}}$ wait until outputs $(\textsc{caucast-del}, \mathsf{mid}_j, m_j)$ were given for $j = 1, \ldots, \ell$ (possibly waiting forever if this never happens), compute $m = \mathsf{NextMessage}^{\mathsf{mid}}((\mathsf{mid}_1, m_1), \ldots, (\mathsf{mid}_\ell, m_\ell))$, and output $(\textsc{caucast-del}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ if $m \neq \bot$.

Leader-Election Deliver: On output $(\textsc{elect}, \mathsf{sid}, K)$ from $\Pi_{\textsc{Elect}}$ output $(\textsc{caucast-del}, \mathsf{sid}, K)$.

Constant Deliver: On input $(\textsc{caucast-send}, \mathsf{mid}, m)$ where mid is a constant identifier, output $(\textsc{caucast-del}, \mathsf{mid}, m)$.

Definition 8 (CauCast Conventions). *When using CauCast we let the justifier* J^{mid} *of a message* m *at* S_i *be that* $(\textsc{caucast-del}, \mathsf{mid}, m, \ldots)$ *was output by* S_i*. When describing protocols using CauCast we assume that all outputs were CauCast and this is how they are justified. Specifically, the session identifier* sid *of the protocol specifies a message identifier* mid *and* $J_{OUT}^{\mathsf{sid}}(m) = J^{\mathsf{mid}}(m)$*. When describing protocols using CauCast we will always as the first instruction CauCast the inputs of the protocol as a free-choice message. In the typical case where the input* x *is the output* y *of some previous protocol we will by convention have that* $y = m$ *for some message with message identifier* mid*, where* m *was by convention already CauCast. In that case we do not reCauCast the*

input x. All servers will by convention know mid *and that $x = m$, so getting input for the protocol just means waiting for* $(\textsc{caucast-del}, \mathsf{mid}, m)$ *and then letting $x = m$.*

We address communication complexity. We can represent a session identifier sid with κ bits as we can always hash the session identifiers. We can let all servers G_i number the messages they send using a counter $c_i = 1, 2, \ldots$. Then message identifiers can be of the form (sid, i, c_i). We do not need to send sid along with all mid as it is the same for all mid in the protocol. Using that $n, N \in \text{poly}(\kappa)$ and that c_i can become at most polynomially large in a poly-time run of the system each mid can therefore be represented in $\log(\kappa)$ extra bits. We can therefore represent $(\mathsf{mid}, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$ as $\kappa + \ell \log(\kappa)$ bits. Therefore the communication complexity is that of reliably broadcasting all free-choice messages m, the leader elections, plus the complexity of reliably broadcasting $\kappa + \ell \log(\kappa)$ per computed message. In many cases a computed message is computed from $n - t$ outputs of n possible messages with session identifiers $\mathsf{sid}_1, \ldots, \mathsf{sid}_n$ known by all servers. In these cases we can send an n-bit vector indicating the $n - t$ session identifiers to use. Then the communication is only $\mathcal{O}(\kappa + n) = \mathcal{O}(\kappa)$ bits per computed message. We return to this when analysing the complexity of concrete protocols below.

Definition 9 (Complexity). *We say that a protocol Π using CauCast has (expected communication) complexity*

$$\mathcal{O}(c_1 \, \textsc{in} + c_2 \, \textsc{rs} + c_3 \, \textsc{rs}_\# + c_4 \textsc{elect} + c_5)$$

if the following holds in expectation, using the above methods for compression, and under $\mathcal{O}$: c_1 is the total number of bits that the protocol needs to reliably subcast[1] *its inputs, c_2 is the total number of bits that the protocol needs to reliably subcast intermediary values and outputs, c_3 is the* number *of reliable subcast instances run, c_4 is the number of calls to the reliable leader election primitive, and c_5 is the total number of bits sent otherwise. Notice that we count all messages sent by all servers. By time complexity of a protocol we mean how long wall clock time it takes to run, measured in units of the longest it takes to deliver a message during the execution, called $\Delta_{\textsc{net}}$.*

The reason why we single out the complexity of inputs is that when we compose protocols, c_1 IN falls away if the inputs are outputs of a previous protocol, as their cost is included in RS of that protocol.

3 Total Order Broadcast from RB and $t < n/2$

We now present a protocol for total order broadcast. We will here make a distinction between ground population which can broadcast and the servers holding the ledger.

Definition 10 (Total Order Subcast). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ and ground population $\mathsf{G}_1, \ldots, \mathsf{G}_N$. There is an input justifier $J_{\textsc{in}}$. Each S_i holds a list Ledger_i.*

Input: G_i *get inputs m where $J_{\textsc{in}}(m) = \top$. We use* $\mathsf{Scheduled}$ *to denote the set of (i, m) such that m was input at an honest G_i. We use $\mathsf{Scheduled}^t$ to denote the value of this set at time t.*
Liveness: *For all honest S_i and all times t it holds eventually that $\mathsf{Scheduled}^t \subseteq \mathsf{Ledger}_i$.*
Monotone: *For all S_i and $t' \geq t$ it holds that $\mathsf{Ledger}_i^t \sqsubset \mathsf{Ledger}_i^{t'}$.*
Agreement: *For all S_i and S_j it holds that $\mathsf{Ledger}_i^t \sqsubset \mathsf{Ledger}_j^t$ or $\mathsf{Ledger}_j^t \sqsubset \mathsf{Ledger}_i^t$.*

[1] I.e., the sum of the length of messages input to a reliable subcast.

If the above holds but with each G_i *holding* Ledger_i *and Liveness, Monotone and Agreement holding for all honest* G_i *and* G_j *instead of* S_i *and* S_j*, then we call the protocol an Total Order Broadcast (for the ground population).*

In this section we build a Total Order Subcast which has communication complexity $\mathcal{O}(\beta \cdot \text{RS})$ when there is enough traffic, as discussed in detail later. By letting the servers outcast the ledger to the ground population the complexity becomes $\mathcal{O}(\beta \cdot \text{RS} + \beta N)$ as we can outcast β bits with complexity βN when outcasting in large enough blocks. The term βN is clearly optimal as all N ground members need to receive all β bits in Total Order Broadcast. In the following we therefore focus on presenting and proving the Total Order Subcast. It will be build up via a sequence of increasingly powerful primitives.

3.1 Justified Gather

We describe and analyse our Justified Gather protocol. We consider protocols where each server has an input $B_i \in \{0,1\}^*$, which we will call a block below.

Definition 11 (block set). *A* block set *is a set of pairs* $U = \{(\mathsf{S}_j, B_j)\}_{\mathsf{S}_j \in P}$*, where* $P \subset \mathbb{S}$ *and* $|P| \geq n - t$*.*

We think of $(\mathsf{S}_j, B_j) \in U$ as S_j having input B_j. The Gather primitive says that each server S_i has a block set U_i as output and there is a large common core, i.e., a set U of size $n - t$ which is a subset of each U_i. So all servers to some extend agree on a large set U, but some S_i might have extra elements in U_i and they do not know which are the extra ones.

Definition 12 (Justified Gather). *A protocol for* n *servers* $\mathsf{S}_1, \ldots, \mathsf{S}_n$*. There is an input justifier* J_{IN} *and an output justifier* J_{OUT} *specified by the protocol. All honest* S_i *have an input* B_i *for which* $J_{IN}(B_i) = \top$ *at* S_i *at the time the input is given.*

Liveness: *If all honest parties start running the protocol with a* J_{IN}*-justified input then eventually all honest servers have a* J_{OUT}*-justified output.*

Justified Blocks: *For all possible justified outputs* U *and all (potentially corrupt)* S_i *and all* $(\mathsf{S}_i, B_i) \in U$ *it holds that* $J_{IN}(B_i) = \top$*.*

Validity: *For all possible justified outputs* U *and all honest* S_i *and all* $(\mathsf{S}_i, B_i) \in U$ *it holds that* S_i *had input* B_i*.*

Agreement: *For all possible justified outputs* U *and* U' *and all* $(\mathsf{S}_i, B_i) \in U$ *and* $(\mathsf{S}_i, B_i') \in U'$ *it holds that* $B_i = B_i'$*.*

Large Core: *For all possible justified outputs* $(U^1, \ldots, U^m)$ *it holds that* $|\bigcap_{k=1}^m U^k| \geq n - t$*.*

The protocol works for $t < n/2$ corruptions and is inspired by the ACS protocol in [Can95] which worked for $t < n/3$ corruptions. Making a variant of this protocol work for $t < n/2$ is central to the results in this paper. The main observation is that if the sending of sets in [Can95] is replaced by CauCast then it works for $t < n/2$. The protocol from [Can95] proceeds via a $\log(n)$-round gather structure. We add to this gather structure that late original blocks are added when they arrive, this is the X_i^r sets below. This means that in constant time all honest parties have their blocks enter the gather structure, which will build a large core in constant time. We then use a termination condition from the Gather protocol CSS in [MMNT19,DMM+20]. This termination condition allows detecting that a large core will be build in the ensuing round. This allows terminating no later than

when all honest inputs entered the gather structure. This makes the overall protocol constant time. Note that the protocol might still run in $\log(n)$ rounds of communication even though it runs in constant time. But this requires that the messages of the fastest honest parties are delivered with delay $\Delta_{\text{NET}}/\log(n)$ and the slowest honest parties have their messages delivered with delay Δ_{NET}, so there must be an asymptotic gap in the delays. The protocol is presented in Fig. 1.

1. The input of S_i is B_i with $J_{\text{IN}}(B_i) = \top$. Server S_i CauCasts $B_i^0 = B_i$. The message is justified by being RB'ed by S_i and $J_{\text{IN}}(B_i) = \top$.
2. Server S_i collects incoming B_j^0 from servers S_j and lets P_i^1 be the set of S_j it heard from. It waits until $|P_i^1| \geq n - t$, lets $X_i^1 = P_i^1$, lets
$$U_i^1 = \{(\mathsf{S}_j, B_j^0)\}_{\mathsf{S}_j \in P_i^1},$$
and CauCasts U_i^1. The set is justified by being computed as above from justified sets.[a]
3. For rounds $r = 2, \ldots$ server S_i proceeds as follows:[b]
   (a) Collects incoming U_j^{r-1} from servers S_j and lets P_i^r be the set of S_j it heard from. It waits until $|P_i^r| \geq n - t$.
   (b) Lets $\mathsf{dist}^{r-1}(\mathsf{S}_k)$ be the number of U_j^{r-1} such that $(\mathsf{S}_k, \cdot) \in U_j^{r-1}$, called the *distribution* of S_k.
   (c) Lets X_i^r be the set of servers S_j from whom S_i by now received a justified input B_j^0.
   (d) Lets
$$U_i^r = \bigcup_{\mathsf{S}_j \in P_i^r} U_j^{r-1} \cup \{(\mathsf{S}_j, B_j^0)\}_{\mathsf{S}_j \in X_i^r}$$
   and CauCasts U_i^r. The set is justified by being computed as above from justified values.
4. If in any round r it happens that there are $n-t$ servers S_k with $\mathsf{dist}^{r-1}(\mathsf{S}_k) \geq n-t$, then output $U_i = U_i^r$, reliable broadcast the message STOP, and run for one more round and terminate after sending U_i^{r+1}. The output $U_i = U_i^r$ is justified by being computed as above from justified sets which have $\mathsf{dist}^{r-1}(\mathsf{S}_k) \geq n-t$ for at least $n - t$ servers S_k.
5. If in any round r it happens that $n - t$ servers RB'ed STOP and S_i already gave output U_i, then it terminates.

[a] Note that U_i^r being a CauCast *computed message* means that sending bit vectors of X_i^r and P_i^r through reliable broadcast allows the receiver to uniquely compute U_i^i.
[b] We later show that this process terminates in at most $\log n$ rounds.

Fig. 1. Protocol Π_{GATHER}.

Theorem 1. *If $t < n/2$ then Π_{GATHER} is a Justified Gather. If $\beta = \sum_{i=1}^{n} |B_i|$ then it has complexity*
$$\mathcal{O}(\beta \, IN + n \log(n) \, RS_{\#} + n^2 \log(n) \, RS) \ .$$
Let Δ_{RB} be the actual delivery time of an RB while the protocol is run. Then all honest servers give output within time $4\Delta_{RB}$. In particular, if the RB has constant time complexity then Π_{GATHER} has constant time complexity.

Proof. We argue Justified Blocks. The message B_i is justified by being RB'ed by S_i and $J_{\text{IN}}(B_i) = \top$. All later messages are justified by justified messages from the previous rounds, and therefore ultimately each B_i is justified by being RB'ed by S_i and $J_{\text{IN}}(B_i) = \top$. This also shows Validity. Agreement follows similarly. For B_i and B_i' to appear as output blocks, they must be justified, which implies that they were both reliably broadcast by S_i. And then they are identical. To argue liveness and large core we need a helper lemma.

We need to analyse the development of the support of the core. A core C^r for round r would be a set of size at least $n-t$ which is a subset of all justified U_i^r. This in particular means it is a subset of $\bigcap_{\mathsf{S}_j \in P_i^r} U_j^r$ for all justified P_i^r, where we call P_i^r justified if all $\mathsf{S}_j \in P_i^r$ sent a justified U_j^r. We say that the core has support σ in round r if it holds for all P_i^r of *size at most* σ that $\bigcap_{\mathsf{S}_j \in P_i^r} U_j^r$ has size at least $n-t$. We let the support of a round r be the largest such σ, i.e.,

$$\Sigma^r = \max\left\{ \sigma \geq 1 \middle| \left(\min_{\text{justified } P_i^r : |P_i^r| \leq \sigma} \left| \bigcap_{\mathsf{S}_j \in P_i^r} U_j^r \right| \right) \geq n-t \right\} ,$$

where we let $\Sigma^r = 0$ if the set is empty.

Lemma 1 (growing core support). $\Sigma^r \geq 2^{r-1}$.

Proof. We prove the lemma by induction. The basis is $r=1$ where $U_i^1 = \{(\mathsf{S}_j, B_j^0)\}_{\mathsf{S}_j \in P_i^1}$. Note that it has size $n-t$ by construction and we only consider intersections between sets of size $\sigma = 2^{r-1} = 1$, so the basis case is trivial. We then assume the induction hypothesis for r and prove it for $r+1$. We have to prove support $\sigma = 2^r$. Consider a justified set P_i^{r+1} with $|P_i^{r+1}| \leq \sigma$. It is enough to show that

$$\left| \bigcap_{\mathsf{S}_j \in P_i^{r+1}} U_j^{r+1} \right| \geq n-t .$$

We can pair the 2^r servers [2] in P_i^{r+1} into $(\mathsf{S}_{j_{k,1}}, \mathsf{S}_{j_{k,2}})$ for $k = 1, \ldots, 2^{r-1}$. For each of these pairs there exists a server S_{j_k} such that $U_{j_k}^r \subseteq U_{j_{k,1}}^{r+1}$ and $U_{j_k}^r \subseteq U_{j_{k,2}}^{r+1}$. This is because both $U_{j_{k,1}}^{r+1}$ and $U_{j_{k,2}}^{r+1}$ to be justified contain the union of $n-t \geq t+1$ justified sets U_j^r, and these sets being justified means that they are reliably broadcast, so since $(n-t)+(t+1) > n$ one identical U_j^r will be a subset of both $U_{j_{k,1}}^{r+1}$ and $U_{j_{k,2}}^{r+1}$. This implies that $U_{j_k}^r \subseteq U_{j_{k,1}}^{r+1} \cap U_{j_{k,2}}^{r+1}$, which implies that

$$\bigcap_{k=1}^{2^{r-1}} U_{j_k}^r \subseteq \bigcap_{k=1}^{2^{r-1}} \left(U_{j_{k,1}}^{r+1} \cap U_{j_{k,2}}^{r+1} \right) .$$

By the induction hypothesis that round r had support 2^{r-1} we have that

$$\left| \bigcap_{k=1}^{2^{r-1}} U_{j_k}^r \right| \geq n-t .$$

By construction

$$\bigcap_{k=1}^{2^{r-1}} \left(U_{j_{k,1}}^{r+1} \cap U_{j_{k,2}}^{r+1} \right) = \bigcap_{\mathsf{S}_j \in P_i^{r+1}} U_j^{r+1} .$$

This combines to give the desired conclusion. □

Now let $\ell = \lceil \log_2(n) \rceil + 1$. Then $\Sigma^\ell \geq 2^{\ell-1} \geq n$, which implies that

$$\left| \bigcap_{j=1}^{n} U_j^\ell \right| \geq n-t .$$

[2] In case $|P_i^{r+1}| < \sigma$ one can consider inserting copies of one server to pad up to σ for the argument to go through.

We first argue Liveness. We do this in two steps. We argue that if no honest server terminates, then at least one honest server terminates. We then argue that if one honest server terminates, then all honest servers terminate. Assume for the sake of contradiction that no honest server terminates. So they all reach round $\ell+1$. Now, clearly for all $\mathsf{S}_k \in \bigcap_{j=1}^{n} U_j^\ell$ we will have that $\mathsf{dist}^\ell(\mathsf{S}_k) \geq n-t$ as it will be in all received U_j^ℓ in round $\ell+1$. Therefore, in round $\ell+1$ all honest servers will have $n-t$ servers S_k with $\mathsf{dist}(\mathsf{S}_k) \geq n-t$ and will output $U_i^{\ell+1}$ if they did not already give output. And then they all run the next round, and then terminate, a contradiction. Assume then that some honest S_i terminates. Let S_i be the first honest server to terminate and let r_i be the round in which it happened. Then in round r_i-1 server S_i had $n-t$ servers S_k with $\mathsf{dist}^{r_i-2}(\mathsf{S}_k) \geq n-t$. Each such S_k was in $n-t \geq t+1$ of the sets $U_j^{r_i-2}$. Since all servers in the next round collect $n-t$ sets $U_j^{r_i-2}$ all servers (corrupted and honest) must collect at least one $U_j^{r_i-2}$ which contains S_k. Therefore S_k will be in *all* justified $U_j^{r_i-1}$. Note that all honest servers run at least until round r_i. In that round they will then all have $\mathsf{dist}^{r_i-1}(\mathsf{S}_k) \geq n-t$ for all $n-t$ servers S_k for which S_i had $\mathsf{dist}^{r_i-2}(\mathsf{S}_k) \geq n-t$ in round r_i-1. Therefore they give output and send STOP. Therefore they will all eventually see $n-t$ STOP messages and then they terminate.

We then argue Large Core. We argued above that all honest servers terminate. It is clear by inspection that if an honest server terminates it also gave an output U_i. We have that $U_i = U_i^{r_i}$ for some r_i. We argued as part of termination that when the first S_i terminated with output $U_i = U_i^{r_i}$ then in round r_i-1 the server S_i had $n-t$ servers S_k with $\mathsf{dist}^{r_i-2}(\mathsf{S}_k) \geq n-t$ and all these servers ended up in all $U_j^{r_i}$. Therefore these S_k servers will be part of all justified outputs U_j.

We then address the communication complexity. In the first round all the blocks are CauCast, for a total of β bits. In the next $\mathcal{O}(\log(n))$ rounds there are no further free choice inputs. The servers only need to CauCast P_i^r and X_i^r, which can be represented as two n-bit vectors. Sending the stop signal is asymptotically negligible. We then consider the time complexity. Let Δ_{RB} be the actual maximum delivery time of an RB while the protocol is run. Within time $2\Delta_{\text{RB}}$ all honest B_i will via some X_j^r have entered U_j^r at all honest S_j. Let r be the round in which this happens. In the next round $(r+1)$, B_i will then be in all justified U_k^{r+1} as all S_k must add in their union one of the $n-t \geq t+1$ sets U_j^r containing B_i. Therefore, in the next round $(r+2)$ we will have $\mathsf{dist}^{r+1}(\mathsf{S}_i) \geq n-t$ for all honest S_i. And then all honest servers give output. Each round consists of one round of reliable broadcast, so it takes time at most Δ_{RB}. Therefore, all in all, within $4\Delta_{\text{RB}}$ time all honest servers give output. □

3.2 Justified Graded Gather

We describe and analyse our Justified Graded Gather protocol. This is just a Justified Gather, where each server also has knowledge about the common core. Each S_i will output a set T_i of size at least $n-t$ which is guaranteed to be a subset of the common core U.

Definition 13 (Justified Graded Gather). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest S_i have an input B_i for which $J_{IN}(B_i) = \top$ at S_i at the time the input is given.*

Liveness: *If all honest servers start running the protocol with a J_{IN}-justified input then eventually all honest servers have a J_{OUT}-justified output.*

Justified Blocks: *For all possible justified outputs (U,T) and all (potentially corrupt) S_i and all $(\mathsf{S}_i, B_i) \in U$ it holds that $J_{IN}(B_i) = \top$.*

Sub Core: *For all possible justified outputs* $((U^1,T^1),\ldots,(U^m,T^m))$ *it holds that* $T^1 \subseteq \bigcap_{k=1}^m U^k$.
Validity: *For all possible justified outputs* (U,T) *and all honest* S_i *and all* $(\mathsf{S}_i,B_i) \in U$ *it holds that* S_i *had input* B_i.
Agreement: *For all possible justified outputs* (U,T) *and* (U',T') *and all* $(\mathsf{S}_i,B_i) \in S$ *and* $(\mathsf{S}_i,B_i') \in U'$ *it holds that* $B_i = B_i'$.
Large Sub Core: *For all possible justified outputs* $((U^1,T^1),\ldots,(U^m,T^m))$ *it holds that* $|\bigcap_{k=1}^m T^k| \geq n-t$.

1. The input of S_i is B_i with $J_{\text{IN}}(B_i) = \top$. All servers run Π_{GATHER} with S_i inputting B_i justified by J_{IN}.
2. Server S_i waits for output U_i' and CauCasts U_i'.
3. Server S_i collects U_j' from servers S_j, lets P_i be the set of S_j it heard from and waits until $|P_i| \geq n-t$.
4. Server S_i lets
$$U_i = \bigcup_{\mathsf{S}_j \in P_i} U_j' \ , \quad T_i = \bigcap_{\mathsf{S}_j \in P_i} U_j'$$
CauCasts (U_i,T_i) and outputs (U_i,T_i). The outputs are justified by being computed as above from justified sets.

Fig. 2. $\Pi_{\text{GRADEDGATHER}}$

Theorem 2. *If* $t < n/2$ *then* $\Pi_{\text{GRADEDGATHER}}$ *is a Justified Graded Gather. If* $\beta = \sum_{i=1}^n |B_i|$ *and* $\Pi_{\text{GRADEDGATHER}}$ *uses* Π_{GATHER} *from Fig. 1 as sub-protocol, then it has complexity*

$$\mathcal{O}(\beta\, \text{IN} + n^2 \log(n)\, \text{RS} + n \log(n)\, \text{RS}_{\#}) \ .$$

The time complexity is $\mathcal{O}(1)$ *times that of the RB used.*

Proof. Complexities, Liveness, Justified Blocks, Validity and Agreement follow from the same properties of Justified Gather. We have that $\bigcap_{k=1}^n T_k = \bigcap_{k=1}^n \bigcap_{\mathsf{S}_j \in P_i} U_j'$, so Large Sub Core follows from Large Core of Justified Gather. Sub Core follows from the below lemma.

Lemma 2 (Sub Core). *Let* $(\cdot,T_i)$ *and* $(U_\iota,\cdot)$ *be any possible justified outputs. Then* $T_i \subseteq U_\iota$.

Proof. It is enough to argue that if $(\mathsf{S}_k,B_k) \in T_i$ then $(\mathsf{S}_k,B_k) \in U_\iota$. If $(\mathsf{S}_k,B_k) \in T_i$ then $\mathsf{S}_k \in U_j'$ for all $\mathsf{S}_j \in P_i$. Since $|P_i| \geq n-t \geq t+1$ it follows that any server receiving $n-t$ justified sets U_j' will also receive a set U_j' with $\mathsf{S}_k \in U_j'$. Namely, the sets are reliably broadcast so if two servers receive justified U_j' and $\hat{U}_j'$ then $U_j' = \hat{U}_j'$. Since all servers collect $n-t$ justified sets $U_\cdot^\ell$ to justify $U_\iota = \bigcup_{\mathsf{S}_j \in P_\iota^\ell} U_j^\ell$ it follows that $(\mathsf{S}_j,B_k) \in U_\iota$. □

3.3 Justified Graded Block Selection

We now present our graded block selection protocol. Here each server has as input a block B_i and as output a block C_i. The goal is to let C_i be one of the inputs and to agree on C_i. Since corrupted servers can pick their own input and we allow that $C_i = B_i$ for a corrupt S_i we simply define validity by saying that the output should be some justified input. Note that this implies that if there is only

1. The input of S_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. The servers run $\Pi_{\text{GradedGather}}$ with input B_i and input justifier J_{IN}. Let the output of S_i be (U_i, T_i).
3. The servers run a justified leader election to elect a justified king S_k. Each server starts the leader election only after getting output from the Graded Gather.
4. Server S_i outputs
$$(C_i, g_i) = \begin{cases} (B_k, 2) & \text{if } \exists(\mathsf{S}_k, B_k) \in T_i \\ (B_k, 1) & \text{if } \exists(\mathsf{S}_k, B_k) \in U_i \setminus T_i \\ (B_i, 0) & \text{if } \nexists(\mathsf{S}_k, \cdot) \in U_i \ . \end{cases}$$
and CauCasts (C_i, g_i) and outputs (C_i, g_i). The output is justified by being computed as above from justified values.

Fig. 3. $\Pi_{\text{GradedSelectBlock}}$

one possible justified input, then that will become the only justifiable output. We will not always be able to perfectly agree on the output, instead the output will have a grade $g \in \{0, 1, 2\}$. The grades are never more than 1 apart and if the grade is 2 then there was agreement on C_i. Finally, we want that with some non-zero probability the grade will be 2.

Definition 14 (Justified Graded Block Selection). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest S_i have an input B_i for which $J_{\text{IN}}(B_i) = \top$ at the time the input B_i is given. The output of the protocol is a block C_i justified by J_{OUT}.*

Liveness: *If all honest servers start running the protocol with a J_{IN}-justified input then eventually all honest servers have a J_{OUT}-justified output.*
Justified Output: $J_{IN}(C_i) = \top$ *holds for all possible J_{OUT}-justified outputs C_i.*
Graded Agreement: *For all possible justified outputs (C_i, g_i) and (C_j, g_j) it holds that $|g_i - g_j| \leq 1$. Furthermore, if both $g_i, g_j > 0$ then $C_i = C_j$.*
Positive Agreement: *There exists $\alpha > 0$ such that with probability at least $\alpha - \text{negl}$ some honest S_i will have output (C_i, g_i) with $g_i = 2$.*
Stability: *If there are possible justified outputs C_i and C_j with $C_i \neq C_j$ then there exist two justified inputs B_i and B_j with $B_i \neq B_j$.*

Theorem 3. *If $t < n/2$ then $\Pi_{\text{SelectBlock}}$ is a Justified Graded Block Selection. When $\beta = \sum_{i=1}^{n} |B_i|$ and when using $\Pi_{\text{GradedGather}}$ from Fig. 2 as sub-protocol the complexity is*

$$\mathcal{O}(\beta \, IN + n^2 \log(n) \, RS + n \log(n) \, RS_{\#} + ELECT) \ .$$

Proof. We start with the complexity. The protocol in Fig. 2 has complexity

$$\mathcal{O}(\beta \, \text{IN} + n^2 \log(n) \, \text{RS} + n \log(n) \, \text{RS}_{\#}) \ .$$

In addition to this $\Pi_{\text{SelectBlock}}$ only does one leader election. It has to send no more PUJM information as the justifier for (C_i, g_i) is the justified B_i, the justified (U_i, T_i), and the justified S_k, which have all been CauCast already. Liveness is straight forward. We argue Justified Output. Let (C_i, g_i) be any justified output. If $g_i = 0$ then by definition $C_i = B_i$ is a justified input. If $g_i > 0$ then $B_i = B_k$ for $(\mathsf{S}_k, C_k) \in U_i$ and therefore C_k is a justified input to the Graded Gather which

also used J_{IN} as input justifier. Then use the Justified Blocks property. To argue Graded Agreement let (C_i, g_i) and (C_j, g_j) be any justified outputs. To argue that $|g_i - g_j| \leq 1$ it is sufficient to prove that if $g_i = 2$ then $g_j \neq 0$. So assume that $g_i = 2$. Then $(\mathsf{S}_k, C_i) \in T_k$ for some justified T_k. Therefore, by Sub Core, $(\mathsf{S}_k, C_i) \in U_j$, and therefore $g_j \geq 1$. Assume then that $g_i, g_j > 0$. In that case $(\mathsf{S}_k, C_i) \in U_i$ and $(\mathsf{S}_k, C_j) \in U_j$, so by Agreement of the Graded Gather it follows that $C_i = C_j$. We then argue Positive Agreement for $\alpha = 1/2$. It is sufficient to argue that with probability $1/2$ it holds for some honest S_i that $\mathsf{S}_k \in T_i$. Consider the first honest S_i to start running the leader election. When this happens T_i is already defined, and S_k is unpredictable. Since $|T_i| = n - t > n/2$ it follows that $\mathsf{S}_k \in T_i$ with probability at least $1/2 - \text{negl}$. To argue Stability just note that it holds for both C_i and C_j that they are justified inputs of $\Pi_{\text{SelectBlock}}$. □

3.4 Justified Block Selection

We now present our (ungraded) block selection protocol. The difference from graded block selection is that all possible justified outputs C_i should be identical.

Definition 15 (Justified Block Selection). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$. There is an input justifier J_{IN} and an output justifier J_{OUT}. All honest S_i have an input B_i for which $J_{\text{IN}}(B_i) = \top$ at the time the input was given. The output of the protocol is a block C_i justified by J_{OUT}.*

Liveness: *If all honest servers start running the protocol with a J_{IN}-justified input then eventually all honest servers have a J_{OUT}-justified output.*
Justified Output: *$J_{\text{IN}}(C_i) = \top$ holds for all possible J_{OUT}-justified outputs C_i.*
Agreement: *For all possible justified outputs C_i and C_j it holds that $C_i = C_j$.*

1. The input of S_i is B_i with $J_{\text{IN}}(B_i) = \top$. It initialises $\mathsf{GaveOutput}_i = \bot$.
2. Let $B_i^0 = B_i$ and $g_i^0 = 0$ and CauCast (B_i^0, g_i^0), which is justified if $J_{\text{IN}}(B_i^0) = \top$ and $g_i^0 = 0$.
3. For rounds $r = 1, \ldots$ the servers run $\Pi_{\text{GradedSelectBlock}}$ where:
   (a) S_i has input B_i^{r-1}.
   (b) The input of S_i is justified by a justified (B_i^{r-1}, g_i^{r-1}).
   (c) S_i eventually gets justified output (B_i^r, g_i^r).
4. In addition to the above loop each S_i runs the following *echo rules*:
   – In the first round r where $\mathsf{GaveOutput}_i = \bot$ and $g_i^r = 2$, set $\mathsf{GaveOutput}_i = \top$ and output $C_i = B_i^r$. The output justifier is the justifier for (B_i^r, g_i^r).
   – In the first round r where $\mathsf{GaveOutput}_i = \bot$ and where some justified (B_j^ρ, g_j^ρ) propagated from $\mathsf{S}_j \neq \mathsf{S}_i$ with $g_j^\rho = 2$, set $\mathsf{GaveOutput}_i = \top$, and output $C_i = B_j^\rho$. The output justifier is the justifier for (B_j^ρ, g_j^ρ).

Fig. 4. $\Pi_{\text{SelectBlock}}$

Theorem 4. *If $t < n/2$ then $\Pi_{\text{SelectBlock}}$ is a Justified Select Block Protocol. When $\beta = \sum_{i=1}^{n} |B_i|$ and using the protocol $\Pi_{\text{GradedSelectBlock}}$ from Fig. 3 as sub-protocol the complexity is*

$$\mathcal{O}(\beta \, \mathit{IN} + n^2 \log(n) \, \mathit{RS} + n \log(n) \, \mathit{RS}_{\#} + \mathit{ELECT}) \ .$$

Proof. We start with the complexity. In Step 2 the servers need only CauCast the inputs B_i as B_i^0 and g_i^0 can be computed from B_i. This gives β IN. Each run of $\Pi_{\text{GradedSelectBlock}}$ from Fig. 3 has complexity $\mathcal{O}(n^2 \log(n)\,\text{RS} + n\log(n)\,\text{RS}_{\#} + \text{ELECT})$, where we ignore the IN component as we are CauCasting computed values which have known message identifiers. Besides this the protocol only CauCasts computed values for which the receiver knows the message identifier, so there is no more information to CauCast. The protocol terminates in expected $\mathcal{O}(1)$ rounds as argued below. This gives the desired complexity. Liveness follows from Positive Agreement: at some point some honest server will have $g_i^r = 2$ and then the protocol will eventually terminate by construction of the echo rules. Justified Outputs is clear by the Justified Output rule of $\Pi_{\text{GradedSelectBlock}}$ which maintains that $J_{\text{IN}}(B_i^r) = \top$ for all r. We then argue Agreement. Assume that some S_i outputs C_i. Then it saw a justified $(B_j^r = C_i, 2)$. Let r be the smallest r for which a justified $(B_j^r, 2)$ was seen by an honest server. Then by graded agreement all justified (B_j^r, g) for round r will have $B_j^r = C_i$. Therefore, by Stability, it holds for all justified (B_j^ρ, g) for rounds $\rho \geq r$ that $B_j^\rho = C_i$. Now consider any other honest server S_k which outputs C_k. Then it saw some justified $(B_j^{r'} = C_k, 2)$. Since we picked r to be minimal we have that $r' \geq r$. From this it follows that $B_j^{r'} = C_i$. Ergo $C_j = C_i$. □

3.5 Justified Agreement on a Core Set

We then present a protocol for Justified Agreement on a Core Set (JACS). It just lets each server propose a set and then picks $n - t$ of them.

Definition 16 (Justified Agreement on a Core Set). *A protocol for n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ with input and output justifiers J_{IN} and J_{OUT}. All honest S_i have an input B_i for which $J_{IN}(B_i) = \top$ at the time of input.*

Liveness: *If all honest servers start running the protocol with a J_{IN}-justified input then eventually all honest servers have a J_{OUT}-justified output.*

Validity: *For all possible J_{OUT}-justified outputs U and all honest S_i and all $(\mathsf{S}_i, B_i) \in U$ it holds that S_i had input B_i.*

Justified Blocks: *For all possible justified outputs U and all (potentially corrupt) S_i and all $(\mathsf{S}_i, B_i) \in U$ it holds that $J_{IN}(B_i) = \top$.*

Agreement: *For all possible justified outputs U_i and U_j it holds that $U_i = U_j$.*

Large Core: *For all possible justified outputs U it holds that $|S| \geq n - t$.*

1. The input of S_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. Server S_i CauCasts B_i. This message is justified by $J_{\text{IN}}(B_i) = \top$ and B_i having been reliably broadcast by S_i.
3. Server S_i collects at least $n - t$ justified B_j from servers $\mathsf{S}_j \in P_i$ and lets $U_i = \{(\mathsf{S}_j, B_j)\}_{\mathsf{S}_j \in P_i}$. This value is justified by each B_j being justified and $|U_i| \geq n - t$. CauCast U_i.
4. Run $\Pi_{\text{SelectBlock}}$ where S_i inputs U_i. The input justifier of $\Pi_{\text{SelectBlock}}$ is to check that U_i is justifiable as defined in the above step.
5. Server S_i gets output C_i from $\Pi_{\text{SelectBlock}}$ and outputs C_i. The output justifier is that C_i is a justified output from the above $\Pi_{\text{SelectBlock}}$.

Fig. 5. Protocol to Agree on a Core Set Π_{ACS}

Theorem 5. *If $t < n/2$ then Π_{ACS} is a JACS protocol. When $\beta = \sum_{i=1}^{n} |B_i|$ and when using $\Pi_{\text{SELECTBLOCK}}$ from Fig. 4 as sub-protocol the complexity is*

$$\mathcal{O}(\beta \, \text{IN} + n^2 \log(n) \, \text{RS} + n \log(n) \, \text{RS}_{\#} + \text{ELECT}) \ .$$

Proof. Safety and liveness properties follows directly from those of $\Pi_{\text{SELECTBLOCK}}$. We address the complexity. The CauCast of B_i costs β IN. CauCasting U_i costs n^2 RS to specify the sets P_i. Running $\Pi_{\text{SELECTBLOCK}}$ from Fig. 4 costs expected $\mathcal{O}(n^2 \log(n) \, \text{RS} + n \log(n) \, \text{RS}_{\#} + \text{ELECT})$, where we ignore the IN component as it is run on computed messages. There are no further costs. □

3.6 Total Order Subcast

We now give a protocol for total order subcast assuming reliable subcast. The protocol uses an erasure code with reconstruction from $n - t$ code words out of n. It also uses a PKI for $\mathbb{G}$. We first discuss some hairy details of how transactions are collected to not clutter the protocol description with these.

Definition 17 (Transaction identifiers, Blocks, Message Sets). *The first item establishes a FIFO delivery on reliable broadcast of transactions. The second item establishes a causal order on transaction sets using vector clocks. The last item collects all messages in the causal past of a transaction set.*

- *In the Π_{TOS} protocol the servers will send around so-called* transaction identifier (j, c_j) *identifying transaction number c_j by G_j. We say that (j, c_j) is justified at S_i if it saw that some (j, c_j, m_j) was reliable subcast by G_j and $c_j = 1$ or $(j, c_j - 1)$ is similarly justified at S_i. This means that the transaction identifiers become justified in order $c_j = 1, 2, \ldots$ and that when (j, c_j) is justified, the message m_j is known. Let $\mathsf{Msg}(j, c_j) = m_j$.*
- *Servers S_i will send out (and relay)* blocks (B_i^e, w_i^e)*, where e is an* epoch number*, B_i^e is a set of transaction identifiers (j, c_j), and $w_i^e = (v_i^1, \ldots, v_i^n)$ is a vector clock. The set B_i^e is justified at S_k if all $(j, c_j) \in B_i^e$ are justified and S_k already received previous blocks $((B_1^{v_i^1}, w_1^{v_i^1}), \ldots, (B_n^{v_i^n}, w_n^{v_i^n}))$ from $(\mathsf{S}_1, \ldots, \mathsf{S}_n)$ which are similarly justified.*[3]
- *Justified blocks define message sets as follows. For justified B_i^e let $\mathsf{Msg}(B_i^e) = \{\mathsf{Msg}(j, c_j) | (j, c_j) \in B_i^e\}$. Let $\mathsf{Msg}(B_i^0, w_i^0) = \emptyset$ and for $e > 0$ and justified (B_i^e, w_i^e) let $\mathsf{Msg}(B_i^e, w_i^e) = \mathsf{Msg}(B_i^e) \cup \bigcup_{j=1}^{n} \mathsf{Msg}(B_j^{v_i^j}, w_j^{v_i^j})$.*

Theorem 6. *If $t < n/2$ then protocol Π_{TOS} is a total order subcast. When β is the total bit-length of all inputs, ι is the number of inputs, and ρ is the number of epochs in the protocol, and when using Π_{ACS} from Fig. 5 as sub-protocol, and assuming that on average inputs have length at least $\kappa \log(\kappa)$, the communication complexity is*

$$\mathcal{O}\left(\beta \, \text{RS} + \iota \cdot \text{RS}_{\#} + \rho \cdot \left(n^2 \log(n) \, \text{RS} + n \log(n) \, \text{RS}_{\#} + \text{ELECT}\right)\right) \ .$$

Proof. The ledger is clearly monotone. Consider Agreement. By Agreement of Π_{ACS} there is agreement on U^e. As Msg is a function this implies agreement on M^e, and thus agreement on Ledger

[3] The blocks with $e = 0$ are empty and always justified, so the recursion ends at $e = 0$.

Init: All S_i lets $\mathsf{Ledger}_i = ()$, $\mathsf{Scheduled}_i = \emptyset$, $\mathsf{OnGoing}_i = \bot$, and $e_i = 0$. Define $B_i^0 = \emptyset$, $w_i^0 = (0, \ldots, 0)$ and that (B_i^0, w_i^0) has been CauCast received from S_i by all S_j already. For all $\mathsf{G}_j \in \mathbb{G}$ let $c_j = 0$.

Input: G_i: On input m at G_i where $J_{\text{IN}}(m) = \top$ let $c_i \leftarrow c_i + 1$ and reliably subcast (c_i, m).

Schedule: S_i: On arrival of RB of (c_j, m) from G_j where $c_j = 1$ or $(j, c_j - 1) \in \mathsf{Scheduled}_i$, add (j, c_j) to $\mathsf{Scheduled}_i$.

Keep vector clock: S_i: Throughout the protocol, for each S_j let v_i^j be the largest integer such that $B_j^{v_i^j}$ was received via CauCast, in particular $v_i^i = e_i - 1$. Let $w_i \coloneqq (v_i^1, \ldots, v_i^n)$. Let $\mathsf{InBlocks}_i \coloneqq \bigcup_j \mathsf{Msg}\left(B_j^{v_i^j}\right)$.

Propose Next Block: Server S_i: If $\mathsf{OnGoing}_i = \bot$ and $\mathsf{Scheduled}_i \setminus \mathsf{InBlocks}_i \neq \emptyset$, let $\mathsf{OnGoing}_i = \top$ and atomically do the following:[a]

1. Let $e_i \leftarrow e_i + 1$.
2. Let $B_i^{e_i} = \mathsf{Scheduled}_i \setminus \mathsf{InBlocks}_i$, $w_i^{e_i} = w_i$, and CauCast $(B_i^{e_i}, w_i^{e_i})$ justified as explained in Definition 17. [b]
3. Start $\Pi_{\text{ACS}}^{e_i}$ with input $(B_i^{e_i}, w_i^{e_i})$ with the input justifier being that the value was CauCast.

Extend Ledger: If $\mathsf{OnGoing}_i = \top$ and $\Pi_{\text{ACS}}^{e_i}$ produces output $U^{e_i} = \{(\mathsf{S}_j, (B_j^{e_i}, w_j^{e_i}))\}_{\mathsf{S}_j \in P}$, then let $M^{e_i} = \cup_{\mathsf{S}_j \in P} \mathsf{Msg}(B_j^{e_i}, w_j^{e_i})$, sort $M^{e_i} \setminus \mathsf{Ledger}_i$ using some deterministic rule to get a list M, and let $\mathsf{Ledger}_i \leftarrow \mathsf{Ledger}_i \| M$. Then let $\mathsf{OnGoing}_i = \bot$.

[a] Here any non-deadlocking condition Wait_i can be added to let $\mathsf{Scheduled}_i \setminus \mathsf{InBlocks}_i$ grow to some bigger size.

[b] Note that this involves only RB'ing identifiers of the values justifying $(B_i^{e_i}, w_i^{e_i})$. We discuss as part of analysing communication complexity exactly which values need to be sent.

Fig. 6. Totally Ordered Subcast Protocol Π_{TOS}.

by a simple induction. We then look at liveness. Liveness of subcast means that (j, c_j) eventually ends up in $B_i^{e_i}$ at all honest S_i. This $B_i^{e_i}$ will eventually reach all honest S_j which will set $v_j^i \geq e_i$. After that m is in $\mathsf{Msg}(B_j^{e_j}, w_j^{e_j})$ for all honest S_j. In the next run of Π_{ACS} message m_j will then be in $\mathsf{Msg}(B_j^{e_j}, w_j^{e_j})$ for all honest $\mathsf{S}_j \in P$. Since $|P| = n - t \geq t + 1$ there is an honest S_j in P, so m will be in M^{e_i} and then Ledger_i. We count complexity. Each input is reliably subcast. This is $\mathcal{O}(\beta \,\text{RS} + \iota \,\text{RS}_{\#})$. To CauCast the blocks we only need to CauCast the transaction identifiers and w_i, as it is clear from w_i which other values are needed to justify the block. We count the total length of the sets in all blocks, i.e., $\sum_{i,e} |\,\text{enc}(B_i^e)|$ for an asymptotically optimal encoding of the set of transaction identifiers. For each input we add (j, c_j) to at most n blocks. Since j and c_j are counters they will be $\mathcal{O}(\text{poly}(\kappa))$ in any poly-time run, so we represent them with $\mathcal{O}(\log(\text{poly}(\kappa))) = \mathcal{O}(\log(\kappa))$ bits. Therefore $\sum_{i,e} |\,\text{enc}(B_i^e)| = \mathcal{O}(\iota n \log(\kappa))$. This overall adds $\mathcal{O}(\iota n \log(\kappa)\,\text{RS})$. We assumed that inputs have length at least $\kappa \log(\kappa)$ and that $n = \Theta(\kappa)$. Therefore $\iota n \log(\kappa) = \mathcal{O}(\beta)$, so the contribution is $\mathcal{O}(\beta\,\text{RS})$. In each epoch S_i CauCasts B_i along with the vector clock which also has size $n \log(\kappa)$. Ignoring the size of the blocks, which we already accounted for, this adds up to $\mathcal{O}(\rho n^2 \log(\kappa)\,\text{RS} + \rho n\,\text{RS}_{\#})$. In each of ρ epochs Π_{ACS} contributes an extra $\mathcal{O}(n^2 \log(n)\,\text{RS} + n \log(n)\,\text{RS}_{\#} + \text{ELECT})$. Sum and use that $n = \Theta(\kappa)$. □

3.7 Total Order Broadcast

We finally note that from TOS we can get TOB. We will use the protocol Π_{TOS} in a white-box manner. After each epoch we will outcast the new part of the ledger to the ground population to let them learn the update. The protocol is given in Fig. 7.

Init: All $\mathsf{G}_i \in \mathbb{S}$ keeps an ordered list Ledger_i as part of Π_{TOS}. All $\mathsf{G}_i \in \mathbb{G} \setminus \mathbb{S}$ will keep their own Ledger_i, initially empty.
Input: On input m at G_i input m to Π_{TOS}.
Outcast: $\mathsf{S}_i \in \mathbb{S}$: When computing $\mathsf{Ledger}_i \leftarrow \mathsf{Ledger}_i \| M$ in Π_{TOS} in epoch e_i input M to $\Pi_{\text{ReliableOutcast}}$ with session identifier e_i.
Extend Ledger: $\mathsf{G}_j \in \mathbb{G} \setminus \mathbb{S}$: On output M from $\Pi^{e_i}_{\text{ReliableOutcast}}$ let $\mathsf{Ledger}_i \leftarrow \mathsf{Ledger}_i \| M$.

Fig. 7. Total-Order Broadcast Π_{TOB}

Theorem 7. *If $t < n/2$ then protocol Π_{TOB} is a total order broadcast. When β is the total bit-length of all inputs, ι is the number of inputs, and ρ is the number of epoch in Π_{TOS}, and when using Π_{TOS} from Fig. 6 and $\Pi_{OUTCAST}$ from Section 2.2 as sub-protocol, and assuming that on average inputs have length at least $\kappa \log(\kappa)$, the communication complexity is*

$$\mathcal{O}\left(\beta(RS+N) + \iota \cdot RS_{\#} + \rho \cdot \left(n^2 \log(n)(RS+N) + n \log(n)\, RS_{\#} + ELECT\right)\right) .$$

Proof. The sub-protocol Π_{TOS} has communication complexity

$$\mathcal{O}\left(\beta\, \text{RS} + \iota \cdot \text{RS}_{\#} + \rho \cdot \left(n^2 \log(n)\, \text{RS} + n \log(n)\, \text{RS}_{\#} + \text{ELECT}\right)\right)$$

and Π_{Outcast} has communication complexity $\mathcal{O}(nN \log(n)\kappa + N|M|)$ for each M. We run Π_{Outcast} once each epoch and on M's of total length β. This gives a total contribution of $\mathcal{O}(\rho n N \log(n)\kappa + N\beta)$ from the outcasting. Then use that $\rho n N \log(n) = \Theta(\rho N n^2 \log(n))$ and sum. □

4 RB with Subsampling

We now present a reliable subcast protocol for N ground members and n servers. We assume that at most $T < N/3$ ground members are corrupt and at most $t < n/2$ servers are corrupt.

As demonstrated in [ACKN23] this setting can be used to get sub-quadratic communication with good constants in a setting with static security from the assumption that less that $N/3$ ground members are corrupt. The basic idea is that from the N ground members one samples a uniformly random subset $\mathbb{S}$ of size $n = \Theta(\kappa)$. Since we are sampling from a set with a supermajority of honest servers and only need that $\mathbb{S}$ has a majority of honest servers the size of $\mathbb{S}$ can be practical. In our setting the set $\mathbb{S}$ could be constructed simply by running $n = |\mathbb{S}|$ justified leader elections in parallel and wait for all of them and let $\mathbb{S}$ be the n winners. A party elected multiple times can be handled by proceeding with a smaller $\mathbb{S}$ or letting the party run multiple servers. There are, however, many different ways in the literature for doing sub-sampling of committees. We therefore leave the concrete subsampling out of the description and just assume that it has been done and that there are at most $t < n/2$ corruptions in $\mathbb{S}$. For now we just remark that if committees are sampled with replacement from the ground population with equal probability, the amount of honest parties in the committee follows the binomial distribution. In [ACKN23] this is used to show that 653 is the minimal committee size needed to get honest majority with 60 bits of statistical security, when assuming less than a third corrupted parties in the ground population. Using the same method—checking that the cumulative distribution function of the honest parties getting elected for any number $\leq n/2$ of seats on the committee is negligible—we compute (cf. Appendix A) the minimal secure committee sizes for various choices of statistical security parameter (σ) and maximal fraction

Setup: We assume a setup for a threshold signature scheme with verification key vk being public and each G_i holding key share sk_i. The reconstruction threshold is $N-T$. The protocol also use a Merkle-tree scheme $(\mathsf{HashTree}, \mathsf{Path}, \mathsf{VerPath})$ and uses an erasure code $\mathsf{EC} = (\mathsf{Enc}, \mathsf{Dec})$ with n code words and reconstruction threshold $n-t$.

Input: G_j: On input (mid, m) with $\mathsf{G}^{\mathsf{mid}} = \mathsf{G}_j$ (i.e., G_j is the designated sender of mid) and $J^{\mathsf{mid}}(m) = \top$ let $(m_1, \ldots, m_n) = \mathsf{EC.Enc}(m)$, let $h = \mathsf{HashTree}(m_1, \ldots, m_n)$, and send (mid, h) to all G_k.

SubSign G_k: On receiving (mid, h) from $\mathsf{G}_j = \mathsf{G}^{\mathsf{mid}}$, where no $(\mathsf{mid}, \cdot)$ was received from G_j before and $J^{\mathsf{mid}}(m) = \top$, send $\sigma_k = \mathsf{Sig}_{\mathsf{sk}_k}((\mathsf{mid}, h))$ to G_j.

Send Shards: G_j: On having received $N-T$ valid signature shares compute $\sigma = \mathsf{Sig}_{\mathsf{sk}}((\mathsf{mid}, h))$ using $\mathsf{Combine}$ and send $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i = \mathsf{Path}(m_1, \ldots, m_n, i))$ to S_i.

Echo and Record own Shard: S_i: On having received $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ with $\mathsf{Ver}_{\mathsf{vk}}((\mathsf{mid}, h), \sigma) = \top$ and $\mathsf{VerPath}(h, i, m_i, \mathsf{Path}_i) = \top$ from $\mathsf{G}_j = \mathsf{G}^{\mathsf{mid}}$ or some server $\mathsf{S}_j \in \mathbb{S}$ send $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ to all other servers and record $(\mathsf{mid}, h, i, m_i)$.

Record other Shards: S_j: On having received $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ with $\mathsf{Ver}_{\mathsf{vk}}((\mathsf{mid}, h), \sigma) = \top$ and $\mathsf{VerPath}(h, i, m_i, \mathsf{Path}_i) = \top$ from S_i record $(\mathsf{mid}, h, i, m_i)$. Note that we deliberately do not echo here.

Combine Shards: S_j: On having recorded $(\mathsf{mid}, h, i, m_i)$ for $n-t$ servers S_i for the same mid, call the set of these servers S, compute $m = \mathsf{EC.Dec}(\{(i, m_i)\}_{\mathsf{S}_i \in S})$ and $(m_1, \ldots, m_n) = \mathsf{EC.Enc}(m)$ and $h' = \mathsf{HashTree}(m_1, \ldots, m_n)$. If $h' = h$ then output (mid, m) and for each $\mathsf{S}_i \in \mathbb{S}$ send $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ to S_i.

Fig. 8. RSS: A protocol for reliable subcast with N ground members, n servers, at most $T < N/3$ corrupt ground members, and at most $t < n/2$ corrupt servers.

of corruption tolerated in the ground population (c). The results are shown in Table 1. In particular it shows that a committee of 173 parties would be resilient against a 1/5 of the ground population being corrupt with 60 bits of security. This compares very favourably against the Algorand setting in which less than a 1/5 of the ground population is assumed to be corrupted in order to get 56 bits of security with committees of 6000 parties (cf. [BBK+23]).

σ	30			40			60			80		
c	1/5	1/4	1/3	1/5	1/4	1/3	1/5	1/4	1/3	1/5	1/4	1/3
n	81	127	307	111	173	423	173	269	653	235	363	887

Table 1. Minimal committee sizes (n) that guarantee an honest majority (except with probability $2^{-\sigma}$) when parties are sampled according to the binomial distribution from a ground population in which less than a fraction c of the parties are corrupted.

Theorem 8. *If $t < n/2$ and $T < N/3$, then* RSS *is a reliable subcast. If messages have length $\beta \geq \kappa \log(\kappa)$, then the complexity is $\mathcal{O}(\beta n + N\kappa + n^2 \log(n)\kappa$, which is of the form $\mathcal{O}(\beta\, RS + RS_{\#})$ for $RS = n$ and $RS_{\#} = N\kappa + n^2 \log(n)\kappa$. For large populations $N \geq n^2 \log(n)$ or large messages $\beta \geq n \log(n)\kappa$ the complexity is $\mathcal{O}(\beta n + N\kappa)$ such that $RS_{\#} = N\kappa$.*

Proof. We argue agreement. If a server accepts $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ then it was signed by $N-T$ ground members and therefore at least $T+1$ honest ground members. Therefore h is unique for mid. And if S_j outputs (mid, m) then $h = \mathsf{HashTree}(\mathsf{EC.Enc}(m))$ and therefore they all output the same m if they output something. Eventual Output 1 is trivial. We argue eventual Output 2. Assume some honest S_j outputs (mid, m). Then it sends $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ to each S_i and therefore each honest S_i sends $(\mathsf{mid}, h, \sigma, m_i, \mathsf{Path}_i)$ to all servers in **Echo and Record own Shard**. There are $n-t$ honest

servers doing this, so all honest will end up recording $n-t$ shards and output (mid, m) in **Combine Shards**. We count complexity. Getting the signature shares from N ground members G_j costs $N\kappa$. Each m_i and Path_i it sent to and from S_i at most $\mathcal{O}(n)$ times which contributes $\mathcal{O}(n|m_i|+n\log(n)\kappa)$. Summing over S_i this gives $\mathcal{O}(n|m| + n^2\log(n)\kappa)$. For large ground populations $N \geq n^2\log(n)$ we have that $\mathcal{O}(\beta n + N\kappa + n^2\log(n)\kappa) = \mathcal{O}(\beta n + N\kappa + N\kappa) = \mathcal{O}(\beta n + N\kappa)$. For large messages $\beta \geq n\log(n)\kappa$ we have that $\mathcal{O}(\beta n + N\kappa + n^2\log(n)\kappa) = \mathcal{O}(\beta n + N\kappa + n\beta) = \mathcal{O}(\beta n + N\kappa + n\beta)$. □

Corollary 1. *Protocol Π_{TOS} when using Π_{ACS} for ACS, $\Pi_{\textsc{Constantine}}$ for leader election, and* RSS *for RB is a total order subcast. Let β be the total length of inputs and let ι be the number of inputs. Then the communication complexity is*

$$\beta n + \iota \cdot (N\kappa + n^2\log(n)\kappa) + \rho \cdot \left(Nn\log(n)\kappa + n^3\log^2(n)\kappa\right) \ . \tag{1}$$

If there is a large ground population $N \geq n^2\log(n)$ or messages which are large on average (i.e. $\beta/\iota \geq n\log(n)\kappa$), and if the protocol on average consumes at least $n\log(n)$ messages per epoch, then the complexity is

$$\beta n + \iota N\kappa \ . \tag{2}$$

Proof. The complexity is $\beta\,\textsc{rs} + \iota \cdot \textsc{rs}_{\#} + \rho \cdot (n^2\log(n)\,\textsc{rs} + \textsc{elect} + n\log(n)\,\textsc{rs}_{\#})$. Plug in $\textsc{rs} = n$, $\textsc{rs}_{\#} = N\kappa + n^2\log(n)\kappa$, and $\textsc{elect} = Nn\kappa$ to get

$$\beta n + \iota \cdot (N\kappa + n^2\log(n)\kappa) + \rho \cdot \left(Nn\log(n)\kappa + n^3\log^2(n)\kappa\right) \ .$$

If on average we consume $n\log(n)$ messages per epoch then $\rho n\log(n) \leq \iota$, so we get

$$\rho \cdot \left(Nn\log(n)\kappa + n^3\log^2(n)\kappa\right) \leq \iota \cdot (N\kappa + n^2\log(n)\kappa) \ .$$

So we can drop the LHS asymptotically. If $N \geq n^2\log(n)$ then $N\kappa \geq n^2\log(n)\kappa$. Equivalently, if $\beta/\iota \geq n\log(n)\kappa$ then $\beta n \geq \iota \cdot n^2\log(n)\kappa$. In both cases we can drop $\iota \cdot n^2\log(n)\kappa$ asymptotically. □

Corollary 2. *Protocol Π_{TOB} when using Π_{TOS} from Corollary 1 for TOS and $\Pi_{\textsc{Outcast}}$ for outcasting is a TOB. Let β be the total length of inputs and let ι be the number of inputs. Then the communication complexity is*

$$\beta N + \iota \cdot (N\kappa + n^2\log(n)\kappa) + \rho \cdot \left(Nn\log(n)\kappa + n^3\log^2(n)\kappa\right) \ . \tag{3}$$

For a large ground population $N \geq n^2\log(n)$ or messages which are large on average (i.e., $\beta/\iota \geq n\log(n)\kappa$), and if the protocol on average consumes at least $n\log(n)$ messages per epoch, then the complexity is

$$\beta N \ . \tag{4}$$

Proof. The contribution of outcasting is $\mathcal{O}(\beta N + \rho Nn\log(n)\kappa)$. The term $\rho Nn\log(n)\kappa$ is already dominated by Eq. (1). Adding βN to Eq. (1) gives Eq. (3). Adding βN to Eq. (2) gives $\beta N + \iota N\kappa$. Then use that $\iota\kappa \leq \beta$ to get Eq. (4). □

5 RB with Dual Threshold and Asymmetric Synchrony Assumptions

We now present a RB for the servers only. We will not focus on communication complexity but resilience. The protocol can be turned into a communication efficient reliable subcast using threshold signatures and the sharding technique from RSS, but we leave this out of the description to focus on the main contribution. The protocol will have two corruption thresholds t_{A} and t_{S} where $t_{\mathrm{A}} \leq t_{\mathrm{S}} \leq n/2$ and where there are at most t_{S} corruptions.

The protocol uses one timeout per server—server S_i waits $\Delta^i_{\textsc{Wait}}$ seconds. The protocol has the property that if there are at most t_{A} corruptions then the running time of protocol does not depend on the $\Delta^i_{\textsc{Wait}}$. However, if the actual number of corruptions t is in the interval $t_{\mathrm{A}} < t \leq t_{\mathrm{S}}$ then the running time of the protocol does depend on the $\Delta^i_{\textsc{Wait}}$.

Input We assume a PKI for a signature scheme. The input of the designated sender S_s is m with $J_{\textsc{In}}(m) = \top$. In response to this input S_S sends $(m, \sigma_s = \mathsf{Sig}_{\mathsf{sk}_s}(m))$ to all servers. Create initially empty sets SignedAsync and SignedSync.

Asynchronous Echo S_i: On receiving (m, σ_s) from S_s, where $J_{\textsc{In}}(m) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_s}(m, \sigma_s) = \top$ and where no such message was received before and there is no $(\mathsf{S}_j, m_j, \sigma_j) \in \mathsf{SignedAsync}$ with $m_j \neq m$, proceed as follows:

1. Let $\sigma_i = \mathsf{Sig}_{\mathsf{sk}_i}((\textsc{Async}, m))$ and send (m, σ_s, σ_i) to all servers.
2. Add $(\mathsf{S}_i, m, \sigma_i)$ to SignedAsync.
3. Set a timeout $\mathsf{Timeout}(\textsc{CollectFromHonest}, \Delta^i_{\textsc{Wait}})$.

Collect Asynchronous Echos All servers: On receiving $(m_j, \sigma_s, \sigma_j)$ from S_j, where $\mathsf{Ver}_{\mathsf{vk}_s}(m_j, \sigma_s) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_j}((\textsc{Async}, m_j), \sigma_j) = \top$ and no such value was received from S_j before, add $(\mathsf{S}_j, m_j, \sigma_j)$ to SignedAsync.

Synchronous Echo S_i: If CollectFromHonest occurred and there exists m such that there are at least $n - t_{\mathrm{S}}$ values $(\mathsf{S}_j, m, \cdot) \in \mathsf{SignedAsync}$ and there does not exist $(\mathsf{S}_k, m', \cdot) \in \mathsf{SignedAsync}$ where $m' \neq m$, then let $\sigma_i = \mathsf{Sig}_{\mathsf{sk}_i}((\textsc{Sync}, m))$ and send (m, σ_i) to all servers.

Collect Synchronous Echos All servers: On receiving (m_j, σ_j) from S_j, where $\mathsf{Ver}_{\mathsf{vk}_j}((\textsc{Sync}, m_j), \sigma_j) = \top$ and no such value was received from S_j before, add $(\mathsf{S}_j, m_j, \sigma_j)$ to SignedSync.

Asynchronous Output All servers: If there exists m such that there are $n - t_{\mathrm{A}}$ values $(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedAsync}$ then let $\Sigma = \{(\mathsf{S}_j, \sigma_j)\}_{(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedAsync}}$, output m, send (m, Σ) to all servers, and terminate.

Synchronous Output All servers: If there exists m such that there are $n - t_{\mathrm{S}}$ values $(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedSync}$ then let $\Sigma = \{(\mathsf{S}_j, \sigma_j)\}_{(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedSync}}$, output m, send (m, Σ) to all servers, and terminate.

Output by Relay On receiving (m, Σ) from any server where either

- Σ contains $n - t_{\mathrm{S}}$ values $(\mathsf{S}_j, m, \sigma_j)$ for distinct S_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\textsc{Sync}, m), \sigma_j) = \top$ (call such a value synchronous-valid), or
- Σ contains $n - t_{\mathrm{A}}$ values $(\mathsf{S}_j, m, \sigma_j)$ for distinct S_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\textsc{Async}, m), \sigma_j) = \top$ (call such a value asynchronous-valid),

output m, send (m, Σ) to all servers, and terminate.

Fig. 9. RBD: A protocol for RB with dual thresholds t_{S} and t_{A}. For conciseness we do not explicitly mentioning the messages identifier mid and we let $\mathsf{S}_s = \mathsf{S}^{\mathsf{mid}}$ and let $J_{\textsc{In}} = J^{\mathsf{mid}}$.

The protocol can be proven secure in two settings. One is a setting where the network is always synchronous and where $t_{\mathrm{S}} + 2t_{\mathrm{A}} < n$. In this setting we need that $\Delta^i_{\textsc{Wait}} \geq 2\Delta_{\textsc{Net}}$. We call this the *optimistic model*. In the other setting we can tolerate that the network is sometimes asynchronous. Here we only need $\Delta^i_{\textsc{Wait}} \geq \Delta_{\textsc{Net}}$. However, we need that $2t_{\mathrm{S}} + t_{\mathrm{A}} < n$. As a strengthening we can here tolerate that either the network is synchronous and there are at most $t \leq t_{\mathrm{S}}$ corruptions or the

network is asynchronous and there are at most $t \leq t_{\text{A}}$ corruptions. We call this the *fallback model* below.

5.1 Asymmetric Synchrony Assumptions

We construct our TOBs for a model with asymmetric synchrony assumptions, which is meant as a model making it easier to implement the needed notion of synchrony in practice. Consider a setting where a group of servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$ have just been thrown together, maybe sampled at random from a larger ground population. They want to run a synchronous protocol to be able to tolerate $t < n/2$. Assume that each S_i can set a sound timeout length Δ^i_{Guess}, i.e., all messages sent to S_i are received within time Δ^i_{Guess}. This still opens the question of what timeout length to use in the protocol if a common timeout is needed. Note that we cannot broadcast the values Δ^i_{Guess} to help us pick a common value, as broadcast is the problem we are trying to solve. This motivates implementing TOB in the following model where the parties do not agree on a common timeout value.

Definition 18 (Asymmetric Synchrony Assumption (ASA) Model). *The ASA model considers n servers $\mathsf{S}_1, \ldots, \mathsf{S}_n$. Each S_i gets its own Δ^i_{Guess} as private input, i.e., the other servers are not given Δ^i_{Guess}. The adversary schedules messages, but it is guaranteed that all messages sent at time t from an honest server to an honest server S_i are delivered no later than at time $t + \Delta^i_{\text{Guess}}$. Note the only messages to S_i are delivered in time Δ^i_{Guess}. Messages to other honest servers may be slower. Round complexity is measured in units of $\Delta^{\text{Max}}_{\text{Guess}} := \max_{\text{honest } \mathsf{S}_i} \Delta^i_{\text{Guess}}$ and Δ_{Net}, where Δ_{Net} is the longest it took to send a message from an honest server to an honest server. In the weakly asymmetric synchrony assumption (WASA) model we make the stronger assumption that all messages between honest servers are delivered faster than any honest Δ^i_{Guess}, i.e., $\Delta_{\text{Net}} \leq \Delta^{\text{Min}}_{\text{Guess}} := \min_{\text{honest } \mathsf{S}_i} \Delta^i_{\text{Guess}}$.*

Definition 19. *Let RBD^{OPT} be the protocol RBD in Fig. 9 with $\Delta^i_{\text{Wait}} = 2\Delta^i_{\text{Guess}}$ and $t_A \leq t_S < n/2$ and $t_S + 2t_A < n$. Let $\mathsf{RBD}^{FALLBACK}$ be the protocol RBD with $\Delta^i_{\text{Wait}} = \Delta^i_{\text{Guess}}$ and $t_A \leq t_S < n/2$ and $2t_S + t_A < n$.*

5.2 Synchronous Security with Asynchronous Fallback

We now analyse $\mathsf{RBD}^{\text{FALLBACK}}$ in the ASA model.

Theorem 9 (Fallback). *$\mathsf{RBD}^{FALLBACK}$ is a justified RB protocol for the model where either the network is ASA synchronous and there are at most t_S corruptions or the network is asynchronous and there are at most t_A corruptions. All servers terminate within time $2\Delta_{\text{Net}} + \Delta^{\text{Max}}_{\text{Guess}}$. Furthermore, if $t \leq t_A$ and the sender is honest then all honest servers terminate within time $2\Delta_{\text{Net}}$.*

It is not hard to see that the protocol has eventual output and validity. The running time is also straight forward. The main observation is that when $t \leq t_{\text{A}}$ then within time Δ_{Net} the $n - t_{\text{A}}$ honest servers trigger **Asynchronous Echo** and then within Δ_{Net} all servers have a asynchronous-valid output. We sketch why the protocol has agreement. The pivotal property which the protocol has by design is the following.

Lemma 3 (Synchronous Echo Agreement). *If two honest S_i and S_j send (m_i, σ_i) and (m_j, σ_j) in **Synchronous Echo** then $m_i = m_j$.*

Proof. Consider S_i and S_j sending (m_i, σ_i) and (m_j, σ_j) in **Synchronous Echo**. Then obviously they sent some $(m_i, \sigma_s, \sigma_i')$ and $(m_j, \sigma_s', \sigma_j')$ in **Asynchronous Echo**. Assume S_i sent $(m_i, \sigma_s, \sigma_i')$ first. Then S_j started $\mathsf{Timeout}(\textsc{CollectFromHonest}, \Delta^j_{\textsc{Wait}})$ for $\Delta^j_{\textsc{Wait}} = \Delta^j_{\textsc{Guess}}$ after $(m_i, \sigma_s, \sigma_i')$ was sent. Assume that the network is synchronous. Since we are in the ASA model S_j thus saw $(m_i, \sigma_s, \sigma_i')$ before $\textsc{CollectFromHonest}$ occurred and (m_j, σ_j) was sent. Therefore $m_j = m_i$, or $(m_i, \sigma_s, \sigma_i')$ would have blocked the sending of (m_j, σ_j). Assume then that the network is asynchronous. Then by assumption $t \leq t_{\text{A}}$. Recall that $2t_{\text{S}} + t_{\text{A}} < n$. If S_i sent (m_i, σ_i) then it saw $n - t_{\text{S}}$ values $(\mathsf{S}_k, m_i, \cdot) \in \mathsf{SignedAsync}$. If S_j sent (m_j, σ_j) then it saw $n - t_{\text{S}}$ values $(\mathsf{S}_k, m_j, \cdot) \in \mathsf{SignedAsync}$. This means they saw values from $n - 2t_{\text{S}} > t_{\text{A}}$ common parties. So they saw $(\mathsf{S}_k, m_i, \cdot)$ and $(\mathsf{S}_k, m_j, \cdot)$ from at least one joint honest S_k. Therefore $m_i = m_j$. □

Lemma 4 (Fallback Agreement). *If S_i and S_j are honest and output m_i and m_j then $m_i = m_j$.*

Proof. If S_i outputs m_i then it saw a valid (m_i, Σ_i) and if S_j outputs m_j then it saw a valid (m_j, Σ_j). If any of the servers saw a synchronous-valid value, then rename the servers such that S_i saw one. This gives three cases on the validity flavour of (m_i, Σ_i)-(m_j, Σ_j): synchronous-synchronous, synchronous-asynchronous, and asynchronous-asynchronous. Assume first they both are synchronous-valid. Recall that $2t_{\text{S}} - t_{\text{A}} < n$. Among the $n - t_{\text{S}}$ servers in Σ_i there is at least one honest server as $n - t_{\text{S}} > t$, where t is the actual number of corruptions. Similarly, among the $n - t_{\text{S}}$ servers in Σ_j there is at least one honest server. Agreement then follows from Lemma 3. Assume then that both (m_i, Σ_i) and (m_j, Σ_j) are asynchronous-valid. Then among the $n - t_{\text{A}}$ servers in Σ_i and the $n - t_{\text{A}}$ servers in Σ_j there are at least $n - t_{\text{A}} - t_{\text{A}} > 2t_{\text{S}} - t_{\text{A}} > t_{\text{S}} \geq t$ common servers. Therefore there is at least one common honest server. Honest servers sign at most one message m. Assume then that (m_i, Σ_i) is synchronous-valid and (m_j, Σ_j) is asynchronous-valid. Among the $n - t_{\text{S}}$ servers in Σ_i and the $n - t_{\text{A}}$ servers in Σ_j there are at least $n - t_{\text{S}} - t_{\text{A}} > t_{\text{S}}$ common servers. Since $t_{\text{S}} \geq t$, where t is the actual number of corruptions, it follows that there is at least one honest server in common among Σ_i and Σ_j. Clearly, if an honest server signs both $(\textsc{Sync}, m)$ and $(\textsc{Async}, m')$ then $m' = m$. Therefore $m_i = m_j$. □

5.3 Synchronous Security with Optimistic Responsiveness

Theorem 10 (Optimistic). *RBD^{OPT} is a justified RB protocol for the WASA model with at most t_{S} corruptions. All servers terminate within time $2\Delta_{\textsc{Net}} + \Delta^{\textsc{Max}}_{\textsc{Guess}}$. Furthermore, if $t \leq t_A$ and the sender is honest then all honest servers terminate within time $2\Delta_{\textsc{Net}}$.*

It is not hard to see that the protocol has eventual output, validity, and the stated time complexity. We sketch why the protocol has agreement. Note that Lemma 3 still holds. Namely, we now wait $2\Delta^i_{\textsc{Guess}}$ instead of $\Delta^i_{\textsc{Guess}}$, we are in the WASA model which gives stronger guarantees, and the network is alway synchronous, so the preconditions used for proving Lemma 3 are all stronger.

Lemma 5 (Optimistic Agreement). *If S_i and S_j are honest and output m_i and m_j then $m_i = m_j$.*

Proof. As in the proof of Lemma 4 we break into cases. The proofs of the cases synchronous-synchronous and asynchronous-asynchronous can basically be repeated verbatim, so we skip them. This leaves us with the case where (m_i, Σ_i) is synchronous-valid and (m_j, Σ_j) is asynchronous-valid. We have that $\Delta^i_{\textsc{Wait}} \geq 2\Delta^i_{\textsc{Guess}} \geq 2\Delta_{\textsc{Net}}$, as we are in the WASA model. Since $n - t_{\text{S}} > t$, clearly,

at least one honest S_k signed $(\textsc{Sync}, m_i)$ which in turn implies that it earlier signed $(\textsc{Async}, m_i)$, say at time t_i. Furthermore, at least one honest S_ℓ signed $(\textsc{Async}, m_j)$, say at time t_j. Assume for the sake of contradiction that $m_i \neq m_j$. If $t_i \leq t_j - \Delta_{\textsc{Net}}$ then $(\textsc{Async}, m_i)$ would by definition of $\Delta_{\textsc{Net}}$ have reached S_ℓ before t_j and then S_ℓ would by construction not have signed $(\textsc{Async}, m_j)$. So we can assume that $t_j < t_i + \Delta_{\textsc{Net}}$. This means that the $(\textsc{Async}, m_j)$ signed by S_ℓ reached S_k by $t_j + \Delta_{\textsc{Net}} < t_i + 2\Delta_{\textsc{Net}}$. When S_k signed $(\textsc{Async}, m_j)$ by time t_i then it did not sign $(\textsc{Sync}, m_i)$ until time $t_i + 2\Delta_{\textsc{Net}}$. But by $t_i + 2\Delta_{\textsc{Net}}$ it received $(\textsc{Async}, m_j)$. And then by construction of **Synchronous Echo** and $m_i \neq m_j$ it will not sign $(\textsc{Sync}, m_i)$, a contradiction. □

6 Corollaries

In this section we mention a few easy corollaries of our result.

6.1 Sub-Quadratic Asynchronous MPC with $T < N/3$

As shown in Lemma 6.1 in [Coh16], given threshold fully homomorphic encryption and total order broadcast and $t < n/2$ corruptions one can implement asynchronous multiparty computation for $t < n/2$. Since we implement total order broadcast among the servers for $t < n/2$ from RB we can directly run [Coh16] in our framework and get AMPC from RB and $t < n/2$. This gives MPC for the fallback model and the optimistic model. We can also run the protocol in the sub-sampling setting with N ground members and n servers, with corruption threshold $t < n/2$ and $T < N/3$. To avoid having specialised setup among the servers $\mathbb{S}$ we can use that [Coh16] uses threshold decryption as a blackbox: it only uses that if all parties agree on a ciphertext c and that it should be decrypted, then they can eventually learn $y = \mathsf{Dec}_{\mathsf{sk}}(c)$. We can therefore secret share sk among $\mathbb{G}$ with reconstruction threshold $T + 1$ and let them provide decryption as a service for the servers. The servers outcast the encryption c of the output and the servers send a decryption share to each party. If we have a large ground population $N \geq n^2 \log(n)$ and they all give one input we can enforce that they give inputs in the same rounds, and then our total order subcast has communication complexity $\mathcal{O}(\beta n)$, where β is the total length of the encrypted input. And outcasting y has complexity $\mathcal{O}(N|y|)$ and returning the decryptions shares has complexity $Nn|y|$. This gives sub-quadratic AMPC for $T < N/3$ corruptions.

Theorem 11 (informal). *Assume $t < n/2$ corruption in $\mathbb{S}$ and $T < N/3$ corruptions in $\mathbb{G}$, assume $N \geq n^2 \log(n)$, let f be an N-party function, let β be the total length of the inputs x_i and let γ be the length of the output $y = f(x_1, \ldots, n_N)$. Then there is an AMPC protocol for $\mathbb{G}$ with communication complexity $\mathcal{O}(n\beta + nN\gamma)$.*

Note that we can use the expensive $\Pi_{\textsc{Constantine}}$ with communication $\mathcal{O}(Nn\kappa)$ to implement AMPC and then use the AMPC to do distributed key generation for $\Pi_{\textsc{Constantine}}$ among the n servers and thereafter do get communication $\mathcal{O}(n^2\kappa)$. It is an interesting open problem to propose concretely efficient asynchronous distributed key distributions for the setting with $t < n/2$.

6.2 Asynchronous Covert TOB with $t < n/2$

We can also get total order broadcast for the model with covert security [AL07]. We assume that every G_i and S_i can be Byzantine but does not do anything which will lead to an eventual common detection, where all honest S_j output a proof that S_i was corrupted. We only

need to assume that at most $t < n$ servers are corrupted for the RB to work. Our protocol works as follows. When S_i sends (mid, m) it sends along a signature σ_i on $(\mathsf{mid}, \mathsf{Hash}(m))$. On receiving $(\mathsf{mid}, m, \sigma_i)$ with a valid signature S_j outputs (mid, m) and forwards $(\mathsf{mid}, h, \sigma_i)$. On receiving two valid $(\mathsf{mid}, h', \sigma_i')$ and $(\mathsf{mid}, h, \sigma_i)$ with $h' \neq h$ a server forwards them and outputs (S_i IS CORRUPT: $(\mathsf{mid}, h', \sigma_i'), (\mathsf{mid}, h, \sigma_i)$). The message m can be relayed with linear complexity using an erasure code EC as in RSS.

Theorem 12 (informal). *Assume that $t < n$ and the servers are Byzantine but covert. Then CRSS is a secure subcast protocol. If messages have length $\beta \geq \kappa \log(\kappa)$, then the complexity is $\mathcal{O}(\beta n + n^2 \log(n)\kappa)$, which is of the form $\mathcal{O}(\beta\, RS + RS_{\#})$ for $RS = n$ and $RS_{\#} = n^2 \log(n)\kappa$.*

We can use this to get a total order subcast protocol secure against $t < n/2$ Byzantine corruptions in the covert model with communication $\beta n + \iota \cdot (n^2 \log(n)\kappa) + \rho \cdot \left(n^3 \log^2(n)\kappa\right)$.

Input The input of the designated sender S_s is m with $J_{\text{IN}}(m) = \top$. In response to this input S_s sends $(m, \sigma_s = \mathsf{Sig}_{\mathsf{sk}_s}(m))$ to all servers. Create initially empty set $\mathsf{SignedAsync}$.

Asynchronous Echo S_i: On receiving (m, σ_s) from S_s, where $J_{\text{IN}}(m) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_s}(m, \sigma_s) = \top$ and where there are no $(\mathsf{S}_j, m_j, \sigma_j) \in \mathsf{SignedAsync}$ with $m_j \neq m$ proceed as below, let $\sigma_i = \mathsf{Sig}_{\mathsf{sk}_i}((\text{ASYNC}, m))$, send (m, σ_s, σ_i) to all servers, and add $(\mathsf{S}_i, m, \sigma_i)$ to $\mathsf{SignedAsync}$.

Collect Asynchronous Echos All servers: On receiving $(m_j, \sigma_s, \sigma_j)$ from S_j, where $\mathsf{Ver}_{\mathsf{vk}_j}((\text{ASYNC}, m_j), \sigma_j) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_s}(m_j, \sigma_s) = \top$ and no such value was received from S_j before, add $(\mathsf{S}_j, m_j, \sigma_j)$ to $\mathsf{SignedAsync}$.

Output All servers: If there exists m such that there are $n - t$ values $(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedAsync}$ then let $\Sigma = \{(\mathsf{S}_j, \sigma_j)\}_{(\mathsf{S}_j, m, \sigma_j) \in \mathsf{SignedAsync}}$, output m, send (m, Σ) to all servers, and terminate.

Output by Relay On receiving (m, Σ) from any server where Σ contains $n-t$ values $(\mathsf{S}_j, m, \sigma_j)$ for distinct S_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\text{ASYNC}, m), \sigma_j) = \top$, output m, send (m, Σ) to all servers, and terminate.

Fig. 10. RBMA: A protocol for RB against mixed adversaries. For conciseness we do not explicitly mentioning the messages identifier mid and we let $\mathsf{S}_s = \mathsf{S}^{\mathsf{mid}}$ and let $J_{\text{IN}} = J^{\mathsf{mid}}$.

6.3 Mixed Adversary ATOB

We can also get ATOB for the model with mixed adversaries. We assume servers can either silently crash or be fully Byzantine corrupted. We assume there are at most t_{BYZ} fully Byzantine parties and t_{CRASH} additional crash-silent corruptions. We let $t = t_{\text{BYZ}} + t_{\text{CRASH}}$ in the protocol. The protocol is given in Fig. 10.

Theorem 13. *Assume that $2t_{CRASH} + 3t_{BYZ} < n$. Then RBMA is a justified reliable broadcast protocol.*

Proof. Eventual output is trivial as there are $n - t$ honest parties. Agreement follows from the fact that any two sets Σ and Σ' will overlap on at least $n - t - t$ parties and from $2t_{\text{CRASH}} + 3t_{\text{BYZ}} < n$ and $t = t_{\text{BYZ}} + t_{\text{CRASH}}$ we have that $n - t - t > t_{\text{BYZ}}$. Validity follows using similar arguments. □

References

ACKN23. Orestis Alpos, Christian Cachin, Simon Holmgaard Kamp, and Jesper Buus Nielsen. Practical large-scale proof-of-stake asynchronous total-order broadcast. Cryptology ePrint Archive, Paper 2023/1103, 2023. https://eprint.iacr.org/2023/1103.

AJM+21. Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation. *CoRR*, abs/2102.09041, 2021.

AL07. Yonatan Aumann and Yehuda Lindell. Security against covert adversaries: Efficient protocols for realistic adversaries. In Salil P. Vadhan, editor, *TCC 2007*, volume 4392 of *LNCS*, pages 137–156, Amsterdam, The Netherlands, February 21–24, 2007. Springer, Heidelberg, Germany.

AO12. Gilad Asharov and Claudio Orlandi. Calling out cheaters: Covert security with public verifiability. In Xiaoyun Wang and Kazue Sako, editors, *ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 681–698, Beijing, China, December 2–6, 2012. Springer, Heidelberg, Germany.

BBK+23. Fabrice Benhamouda, Erica Blum, Jonathan Katz, Derek Leung, Julian Loss, and Tal Rabin. Analyzing the real-world security of the algorand blockchain. Cryptology ePrint Archive, Paper 2023/1344, 2023. `https://eprint.iacr.org/2023/1344`.

BCG93. Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In *25th ACM STOC*, pages 52–61, San Diego, CA, USA, May 16–18, 1993. ACM Press.

BKL19. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I*, volume 11891 of *Lecture Notes in Computer Science*, pages 131–150. Springer, 2019.

BKLL20. Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I*, volume 12550 of *Lecture Notes in Computer Science*, pages 353–380. Springer, 2020.

Bra87. Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, nov 1987.

Cac21. Christian Cachin. Asymmetric distributed trust. In *ICDCN '21: International Conference on Distributed Computing and Networking, Virtual Event, Nara, Japan, January 5-8, 2021*, page 3. ACM, 2021.

Can95. Ran Canetti. Studies in secure multiparty computation and applications. 1995.

CHL21. Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. *IACR Cryptol. ePrint Arch.*, page 1174, 2021.

CKS00. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing, July 16-19, 2000, Portland, Oregon, USA*, pages 123–132. ACM, 2000.

CKS20. Shir Cohen, Idit Keidar, and Alexander Spiegelman. Not a coincidence: Sub-quadratic asynchronous byzantine agreement WHP. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 25:1–25:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

Coh16. Ran Cohen. Asynchronous secure multiparty computation in constant time. In Chen-Mou Cheng, Kai-Min Chung, Giuseppe Persiano, and Bo-Yin Yang, editors, *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*, volume 9615 of *Lecture Notes in Computer Science*, pages 183–207. Springer, 2016.

CP15. Ashish Choudhury and Arpita Patra. Optimally resilient asynchronous MPC with linear communication complexity. In Sajal K. Das, Dilip Krishnaswamy, Santonu Karkar, Amos Korman, Mohan J. Kumar, Marius Portmann, and Srikanth Sastry, editors, *Proceedings of the 2015 International Conference on Distributed Computing and Networking, ICDCN 2015, Goa, India, January 4-7, 2015*, pages 5:1–5:10. ACM, 2015.

CT05. Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In Pierre Fraigniaud, editor, *Distributed Computing, 19th International Conference, DISC 2005, Cracow, Poland, September 26-29, 2005, Proceedings*, volume 3724 of *Lecture Notes in Computer Science*, pages 503–504. Springer, 2005.

CT19. Christian Cachin and Björn Tackmann. Asymmetric distributed trust. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, *23rd International Conference on Principles of Distributed Systems, OPODIS 2019, December 17-19, 2019, Neuchâtel, Switzerland*, volume 153 of *LIPIcs*, pages 7:1–7:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

CZ21. Christian Cachin and Luca Zanolini. Asymmetric asynchronous byzantine consensus. In Joaquín García-Alfaro, Jose Luis Muñoz-Tapia, Guillermo Navarro-Arribas, and Miguel Soriano, editors, *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2021 International Workshops,*

DPM 2021 and CBT 2021, Darmstadt, Germany, October 8, 2021, Revised Selected Papers, volume 13140 of *Lecture Notes in Computer Science*, pages 192–207. Springer, 2021.

DDFN07. Ivan Damgård, Yvo Desmedt, Matthias Fitzi, and Jesper Buus Nielsen. Secure protocols with asymmetric trust. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2007.

DMM+20. Thomas Dinsdale-Young, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort: A partially synchronous finality layer for blockchains. In Clemente Galdi and Vladimir Kolesnikov, editors, *Security and Cryptography for Networks - 12th International Conference, SCN 2020, Amalfi, Italy, September 14-16, 2020, Proceedings*, volume 12238 of *Lecture Notes in Computer Science*, pages 24–44. Springer, 2020.

DMM+22. Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 683–696. ACM, 2022.

FHM98. Matthias Fitzi, Martin Hirt, and Ueli M. Maurer. Trading correctness for privacy in unconditional multi-party computation (extended abstract). In Hugo Krawczyk, editor, *Advances in Cryptology - CRYPTO '98, 18th Annual International Cryptology Conference, Santa Barbara, California, USA, August 23-27, 1998, Proceedings*, volume 1462 of *Lecture Notes in Computer Science*, pages 121–136. Springer, 1998.

GHM+17. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. Cryptology ePrint Archive, Paper 2017/454, 2017. `https://eprint.iacr.org/2017/454`.

GMR88. Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

GP92. Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In Adrian Segall and Shmuel Zaks, editors, *Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings*, volume 647 of *Lecture Notes in Computer Science*, pages 153–165. Springer, 1992.

HKL20. Martin Hirt, Ard Kastrati, and Chen-Da Liu-Zhang. Brief announcement: Multi-threshold asynchronous reliable broadcast and consensus. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*, volume 179 of *LIPIcs*, pages 48:1–48:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

KKNS21. Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is DAG. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 165–175. ACM, 2021.

MMNT19. Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. Afgjort - A semi-synchronous finality layer for blockchains. *IACR Cryptol. ePrint Arch.*, page 504, 2019.

MP91. F.J. Meyer and D.K. Pradhan. Consensus with dual failure modes. *IEEE Transactions on Parallel and Distributed Systems*, 2(2):214–222, 1991.

PS17. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In Andréa W. Richa, editor, *31st International Symposium on Distributed Computing, DISC 2017, October 16-20, 2017, Vienna, Austria*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

A Script for calculating minimal committee sizes

The following python script computes the minimal secure committee for various security parameters and corruption thresholds.

```
from scipy.stats import binom
import math
from fractions import Fraction
```

```
# Check if a committee of size n has an honest majority
# with probability at least 1-2^-secparam
# when sampled from a ground population with at most t corruptions
def check_committee(n, secparam, t = 1/3):
    negl_prob = Fraction(1, pow(2, secparam))
    least_majority = math.ceil((n + 1) / 2)
    # probability that each member is honest
    p_honest = 1-t
    # cdf over honest parties having 0,1,...,least_majority-1 spots
    p_corrupt_majority = binom(n, p_honest).cdf(least_majority-1)
    return p_corrupt_majority < negl_prob

def smallest_safe_committee(secparam, t = 1/3):
    lower_bound = 1
    while not check_committee(lower_bound * 2, secparam, t):
        lower_bound *= 2
    upper_bound = 2 * lower_bound
    while lower_bound < upper_bound:
        test = (lower_bound + upper_bound) // 2
        if check_committee(test, secparam, t):
            upper_bound = test
        else:
            lower_bound = test
        if upper_bound - lower_bound < 2:
            break
    # Heuristically subtract 10 from lower bound and try each possibility
    # because the predicate is not monotone.
    # The relative difference between odd and even size committees is
    # larger for smaller committees and we overshoot by < 4 for the
    # examples in main, so 10 should suffice for realistic security parameters
    for i in range(lower_bound - 10, upper_bound + 1):
        if check_committee(i, secparam, t):
            print(
                i, "in committee and up to", t,
                "of ground population corrupted gives honest majority with",
                secparam, "bits security")
            return i


if __name__ == '__main__':
    security_parameters = [30, 40, 60, 80]
    # Get numbers for t = 1/3 optimal resiliency
    print("Committee size with optimal resilience:")
    for secpar in security_parameters:
        smallest_safe_committee(secpar)
    # t = 0.25
    print("Committee size with <1/4 of GP corrupted:")
    for secpar in security_parameters:
        smallest_safe_committee(secpar, 1/4)
    # algorand setting, t = 0.2
    print("Committee size -<1/4 of GP corrupted (Algorand setting):")
    for secpar in security_parameters:
        smallest_safe_committee(secpar, 1/5)
```