Byzantine Agreement Decomposed: Honest Majority Asynchronous Atomic Broadcast from Reliable Broadcast

Simon Holmgaard Kamp and Jesper Buus Nielsen

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

Abstract. It is well-known that Atomic Broadcast (AB) in asynchronous networks requires randomisation and that at most $t < n/3$ out of n players are Byzantine corrupted. This is opposed to synchronous AB which can tolerate $t < n/2$ corruptions and can be deterministic. We show that these requirements can be conceptually separated by constructing an asynchronous AB protocol which tolerates $t < n/2$ corruptions from blackbox use of Common Coin and Reliable Broadcast (RB). We show the power of this conceptually simple contribution by instantiating RB under various assumptions to get AB under the same assumptions. Using this framework we obtain the first asynchronous AB with sub-quadratic communication and optimal corruption threshold $t < n/3$, and the first network agnostic AB which is optimistically responsive. The latter result is secure in a relaxed synchronous model where parties locally decide timeouts and do not have synchronized clocks. Finally, we provide asynchronous ABs with covert security and mixed adversary structures.

1 Introduction

We present a protocol for Asynchronous Atomic Broadcast (AAB) which tolerates $t < n/2$ Byzantine corruptions assuming Reliable Broadcast [Bra87] (RB) and a source of common randomness. Our main technical tool is a Gather[1] protocol based on [AW04].We use this to instantiate a protocol for Agreement on a Core Set (ACS) through a sequence of subprotocols that can be seen as a non-pipelined version of [KKNS21] which works for honest majority assuming RB and a common coin. As the common coin can be implemented with honest majority assuming a threshold setup [CKS00], this can be combined into an Atomic Broadcast Π_{AAB} which is resilient against $t < n/2$ Byzantine corruptions. It uses the RB blackbox, so one can instantiate RB in a setting where assumption A holds to get AB for the setting where $t < n/2$ and A holds. 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 AB from A. Our compiler is efficient, so this allows to construct efficient AB in various models.

Using this framework we provide the first AAB which has subquadratic communication while retaining optimal resilience against $t < n/3$ static corruptions. Starting from a ground population of n parties with $t < n/3$ we can sample a $\mathcal{O}(\kappa)$ sized committee with honest majority to run Π_{AAB}. This is in contrast to previous works such as [BKLZL20,BLZLN22] that start from an assumption of some constant fraction less than optimal resilience which allows them to sample a committee with honest supermajority. To instantiate Π_{AAB} we then construct a RB protocol for the committee assisted by the ground population. Finally we can transform Π_{AAB} into a full fledged

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

[1] Using the name of [AJM+21], but the primitive is also known as Get Core[AW04], Common Core [DG16] or Core Set Selection [MMNT19,DMM+20]

AAB for the ground population by sending extensions of the ledger out to the ground population in batches through a protocol we name Outcast. Since both RB and Outcast can be instantiated with communication that is linear in n (via accumulators and threshold signatures), the resulting construction has communication linear in n. It is additionally concretely efficient and *message length optimal* in the sense that posting a message on the AAB has no asymptotic overhead over sending it directly to all parties, for sufficiently many parties or large messages.

We then present a network agnostic AB protocol which is secure in synchronous networks assuming $t < t_{\text{S}}$ corruptions and in the asynchronous networks when $t < t_{\text{A}}$ when $2t_{\text{S}} + t_{\text{A}} < n$ and $t_{\text{A}} \leq t_{\text{S}} < n/2$. This setting was initially explored for BA in [BKL19] where they obtain the same resilience and show that it is optimal. The novelty of our construction is that it additionally achieves *optimistic responsiveness* [PS18], which means that when $t < t_{\text{A}}$ the protocols latency is proportional to the actual network delay and in particular independent of the timeouts used in the protocol. We show security of the protocol in a relaxed synchronous model that does not rely on parties having synchronous clocks and lets them individually decide on how long they want to wait for messages. The proofs assume that all honest parties use a timeout which exceeds the actual network delay, which means that an—in all other respects—honest party who fails to wait for long enough would count toward the corruption budget. We dub this relaxed synchronous model: Asymmetric Synchrony Assumptions. This implies synchronous security as a special case.

We finally show a few implications of our framework by presenting a subquadratic asynchronous MPC protocol, as well as an asynchronous dishonest majority RB and honest majority AB protocol for Byzantine but covert corruptions and a RB and AB for mixed adversaries. In summary we get the following results:

1. In the asynchronous setting with static corruptions we get the first AAB 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 AAB protocol with covert security against $t < n/2$ corruptions.
3. 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 clocks drift. We give a RB for the timeout model tolerating $t < n/2$ corruptions.
4. 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 AB and might not share the same assumptions on the network, or know each other's assumptions prior to running the protocol. We give a 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.
5. We give a network agnostic RB for the ASA model which is secure for optimal corruption thresholds (cf. [BKL19]) $2t_{\text{S}} + t_{\text{A}} < n$ and $t_{\text{A}} \leq t_{\text{S}} < n/2$ when *either* there are at most $t \leq t_{\text{S}}$ corruptions and the network is ASA synchronous *or* there are at most $t \leq t_{\text{A}}$ corruptions and the network is fully asynchronous. The protocol is additionally optimistically responsive [PS18],

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 when $t \le t_{\text{A}}$. Setting $t_{\text{A}} = 0$ gives the contributions claimed in Item 3 and Item 4.

6. When the network is always assumed to be synchronous we can instantiate the above RB to have optimistic responsiveness with the optimal thresholds: $t_{\text{S}} + 2t_{\text{A}} < n$, synchronous security when $t \le t_{\text{S}} < n/2$, and responsiveness when $t \le t_{\text{A}}$. This gives an optimistically responsive AB in which the optimistic condition does not include having an honest leader. The protocol is secure in the WASA model and the timeout model.
7. We finally note that it is an easy corollary of our framework that you can get asynchronous AB for a model with mixed adversaries, where there are at most t_{Byz} fully Byzantine corruptions, at most t_{Crash} additional crash-silent errors, and $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$.

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

1.1 Technical Overview and Related Work

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

Honest Majority AAB from RB In Section 3 we present an honest majority AAB protocol, Π_{AAB}. At a high level Π_{AAB} consists of sequential invocations of an ACS protocol, Π_{ACS}. Π_{ACS} uses a Gather protocol, Π_{Gather}, to make sure that all parties have accumulated a set that include the input of most parties. They then exchange the accumulated sets and take union and intersection of these to get an explicit sub- and superset of the common core of the sets, and flip a coin to elect a leader with graded agreement and repeat the process until they get grade 2. The main technical insight needed to make this work with $t < n/2$ Byzantine corruptions is that the Gather protocol presented in [AW04] which is secure against $t < n/2$ crash faults is in fact secure against $t < n/2$ Byzantine faults if all messages are sent through reliable broadcast and justified by their causal past. A version of the protocol with this machinery was presented in [DG16], with the goal of adapting it to the Byzantine setting, but the proof uses a modified combinatorial argument which is incompatible with honest majority. We stress that the protocol presented in [DG16] is essentially identical to Π_{Gather} (expressed through different abstractions) and as such is also secure with honest majority assuming RB. As their protocol is the basis for the DAG-Rider protocol presented in [KKNS21] we conjecture that DAG-Rider would also be secure against $t < n/2$ Byzantine faults assuming RB.

Subquadratic AAB with Optimal Resilience In Section 4 we present an AAB with subquadratic communication and optimal resilience against static Byzantine corruptions, Π_{SQAAB}. To the best of our knowledge no previous work has considered subquadratic consensus in the asynchronous setting with optimal resilience. Other solutions to subquadratic BA in the asynchronous [BKLZL20,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. Since 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 (i.e. $\epsilon = 2/5$). However, when some task tolerates significantly more than a third of its participants being corrupted, it can securely be delegated to a randomly sampled committee without compromising on the resilience of the overall protocol. If the gap is significantly big, then the committee can even be concretely small. This was used in [ACKN23] to sample honest majority committees to recompute setups for proof of stake AAB protocols by broadcasting poly(κ) bits through the AAB while retaining optimal resilience. Π_{SQAAB} follows the [ACKN23] approach and samples a committee to run Π_{AAB}. We stress that the committee in [ACKN23] does not run an AAB protocol, but uses one blackbox to refresh the setup for one. When instantiated to tolerate strictly up to third corruptions (i.e. $\epsilon = 0$) Π_{SQAAB} would need committees of 653 parties to get 60 bits of security, but for an apples to apples comparison if one is willing to assume the corruption threshold of [BBK$^+$23] the protocol only needs a committee of 173 parties to get 60 bits of security (cf. Table 1). To instantiate Π_{AAB} we need a RB protocol, which usually would employ all-to-all communication in a committee with honest supermajority. However, we construct a RB protocol, Π_{SQRB}, that only depends linearly on the size of the ground population (which has honest supermajority) while the remaining communication is done within the committee. Concretely, the designated sender first obtains a threshold signature from $n - t$ parties in the ground population to make the message unique and then forwards the signed message to the committee who gossip among themselves to make sure that if one committee member gives output, then they forward information which makes sure that all committee members can give output. This property of RB is often referred to as *totality*. For better concrete efficiency we only implement totality for the committee, which means that Π_{SQRB} is not a RB for all parties, but only for the committee. Our actual implementation is a bit more involved in order to get not only subquadratic but message length optimal communication. Because the ground population need to know where to send their signature shares, 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.

Message length optimality A wide variety of works have considered message length optimal protocols that have no amortized communication overhead over sending the message if the message is sufficiently long. Typically this implies that messages must have size $\Omega(n)$ to get amortized linear cost as most consensus protocols have quadratic communication. The work of Banghale et al. [BLZLN22] achieves both message length optimality and subquadratic communication for Byzantine Agreement on long messages in the [BKLZL20] paradigm with $t < (1 - \epsilon)n/3$ corruptions. We are not aware of any previous work that achieves message length optimality for message sizes sublinear in n with optimal resilience. The DAG rider protocol [KKNS21] achieves communication of $\mathcal{O}(n^3 \log(n)\kappa)$ which is optimal when the messages from each party includes $\mathcal{O}(n \log(n)\kappa)$ bits.[2] 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 $|m|$. The same asymptotic communication complexity is achieved in [GLL$^+$22]. In comparison our AAB protocol Π_{SQAAB} combines committee sampling, accumulators and erasure codes to get the communication cost of ordering messages of β bits down to the amortized optimal $\mathcal{O}(n\beta)$ when the protocol consumes on average at least κ messages per epoch and *either* the average messages size is at least κ^2 bits or $n = \Omega(\kappa^2)$ and the average messages size is at least $\kappa \log \kappa$ bits.

[2] Assuming the messages do not contain an asymptotically significant amount of duplicate information.

Optimistic Responsiveness Optimistically responsive protocols [PS18] 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 $t_A < t_S$ parties are corrupted. In some cases, additional optimistic conditions, e.g. a leader being honest, must be met. In Section 5 we provide an optimistically responsive RB protocol matching the optimal bounds for t_A and t_S (i.e. $t_A \leq t_S < n/2$ and $t_A + 2t_S < n$), which in turn also gives an optimistically responsive AB protocol. As the resulting protocol only retroactively elects a leader, and any justified 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.

Network Agnostic Consensus The concept of network agnostic protocols tolerating t_S corruptions under synchrony and additionally tolerating $t_A < t_S$ corruptions under asynchrony was introduced in [BKL19]. They give a network agnostic BA protocol assuming $0 < t_A < n/3 \leq t_S < n/2$ and $t_A + 2t_S < n$ and show that this is optimal. It turns out that the RB protocol we use to achieve optimistic responsiveness is network agnostic when instantiated with these thresholds. However, it retains optimistic responsiveness when $t < t_A$. This is a result of the RB simultaneously executing a synchronous and asynchronous path to termination, instead of first attempting to reach agreement synchronously with $n - t_S$ parties and only afterwards attempting asynchronous agreement with $n - t_A$ parties. Once more this immediately gives an AB protocol with the same thresholds, which is network agnostic and optimistically responsive. A network agnostic AB was first given by Blum et al. in [BKL21], in which they also adapt the lower bound on the corruption thresholds from [BKL19] to the problem of AB. Our AB construction additionally implies that the corruption thresholds are optimal for network agnostic RB, as any RB that would improve on these thresholds would violate the lower bound for AB.[3]

Asymmetric Synchrony Assumptions To the best of our knowledge it is new that one can do AB 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 AB 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 atomic broadcast was also studied in [CT19,CZ21,Cac21], but again only the assumptions on corruptions are asymmetric, not assumptions on network delays.

Honest Majority Asynchronous Atomic 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

[3] Note that the network agnostic model is only defined for honest majority, so security of Π_{AAB} follows from the security of the RB and a common coin which can be instantiated with honest majority from a threshold setup.

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 where the honest party also receives a certificate demonstrating that a party cheated. We informally prove in Section 6.1 that we can instantiate asynchronous RB with dishonest majority and covert security with public verifiability, which in turn gives AAB 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.

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 AAB 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 AAB 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 6.2, 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 which we present in Section 6.3 is the fairly simple observation that using our subquadratic AAB 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 against static corruptions. It has previously been shown that AMPC can be linear in the circuit size[CP15]. [CHLZ21] 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 κ^2 members of the ground population give input of combined size β, and the output has size γ then the communication complexity is $\mathcal{O}(\kappa\beta + n\kappa\gamma)$.

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 4.1 when discussing concrete parameters.

We consider protocols for a fixed set of parties $\mathcal{P} = \{\mathsf{P}_1, \dots, \mathsf{P}_n\}$. For the protocols presented in Section 3 we instead consider a set of parties $\mathcal{C} = \{\mathsf{C}_1, \dots, \mathsf{C}_{n_\mathcal{C}}\}$, which we call the committee. We assume that at most $t_\mathcal{C} < n_\mathcal{C}/2$ of these are corrupted. The reason for introducing this distinction is that in those of our constructions which have subquadratic complexity we will be describing protocols for $\mathcal{P}$, assume that at most $t < n/3$ of these are corrupted, and then sample a committee, $\mathcal{C} \subseteq \mathcal{P}$, to run Π_{AAB} from Section 3. In Section 4.1 we discuss how to set $n_\mathcal{C}$ to ensure $t_\mathcal{C} < n_\mathcal{C}/2$

except with probability $2^{-\sigma}$ with statistical security parameter σ. For simplicity we assume that $n_{\mathcal{C}} = \Theta(\kappa)$ as it allows a more concise analysis in places.

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

We assume a collision resistant hash function Hash. We will be using accumulators to efficiently proof membership of sets. We assume as setup and accumulation key, ak, has been generated. Using this an accumulation value of a set S can be deterministically computed using $z = \mathsf{Acc}(ak, S)$. Then for each $s \in S$ a witness can be computed using $w = \mathsf{Wit}(ak, z, s)$, and inclusion of s in S can then be checked using $\mathsf{Mem}(ak, z, w, s)$. These proofs of membership can be of size $\mathcal{O}(\kappa)$ using RSA accumulators [Bd94].

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 $\mathsf{P}_i \in \mathcal{P}$ 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 P_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 P_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 $\mathcal{P}$. 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 P_i. The secret values can be correlated, for instance a sharing of a secret key.

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 $\mathcal{P}_i$, $\mathsf{Sig}_{\mathsf{sk}_i}$ partially signs m, Ver can verify a partial signature, and $\mathsf{Combine}$ computes $\sigma = \mathsf{Sig}_{\mathsf{sk}}(m)$ from

$n - t$ verified shares. 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 $\mathcal{P}_i$, $\mathsf{Enc}_{\mathsf{ek}}$ 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 from, 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. 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 $J^{\mathsf{mid}}(m, \mathsf{P}, \tau)$ to denote that we apply J^{mid} to m and the local state of P at time τ. 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* τ *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, \tau) = \top$ *then at all* $\tau' \geq \tau$ *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, \tau') = \top$.

Propagating: *If for honest* P *and some point in time* τ *it holds that* $J^{\mathsf{mid}}(m, \mathsf{P}, \tau) = \top$, *then eventually the execution will reach a time* τ' *such that* $J^{\mathsf{mid}}(m, \mathsf{P}', \tau') = \top$ *for all honest parties* P'.

Most of our protocols will be *justified protocols* which puts validity constraints on the inputs and outputs.

Definition 2 (Justified Protocol). *A justified protocol* Π *can have input and output justifiers.*

Input justifier: *If a protocol has an input justifier* J_{IN} *it means that a message identifier* mid_{IN} *is associated with the input,* $J_{IN} = J^{\mathsf{mid}_{IN}}$, *and it is* $\mathsf{P}^{\mathsf{mid}_{IN}}$ *which gets the input. Furthermore, if* $\mathsf{P}^{\mathsf{mid}_{IN}}$ *is honest then it is guaranteed that* $J_{IN}(m) = \top$ *whenever* m *is input to* $\mathsf{P}^{\mathsf{mid}_{IN}}$.

Output justifier: *If a protocol has an output justifier* J_{OUT} *it means that a message identifier* mid_{OUT} *is associated with the output,* $J_{OUT} = J^{\mathsf{mid}_{OUT}}$, *it is* $\mathsf{P}^{\mathsf{mid}_{OUT}}$ *which gives the output, and when it gives the output it may send the output to all parties with message identifier* mid_{OUT}. *Furthermore, if* $\mathsf{P}^{\mathsf{mid}_{OUT}}$ *is honest and gets output* m *then it is a security property of the protocol that if* P *does send its output to all parties, then* $J_{OUT}(m) = \top$.

Similarly most internal protocol messages will come with a justifier, which intuitively is used to force the adversary to behave honestly. In more precise terms: we will say that a predicate P holds for all *possible justified messages* of a protocol, by which we mean that it holds for honest messages which are sent to all parties and in addition that the adversary cannot even cook up messages which look justified to some honest party but do not have the property P.

Definition 3 (Possible Justified Messages). *Let Π be a protocol. When we say that an ℓ-ary predicate P holds* for all possible justified messages *we mean: Run the protocol Π under attack by the adversary. At some point the adversary may output a sequence of triples $(\mathsf{P}^1, \mathsf{mid}^1, m^1), \ldots, (\mathsf{P}^\ell, \mathsf{mid}^\ell, m^\ell)$. We say that the adversary wins if the messages identifiers $\mathsf{mid}^1, \ldots, \mathsf{mid}^\ell$ identify messages of Π, $\mathsf{P}^1, \ldots, \mathsf{P}^\ell$ are honest (but not necessarily distinct) parties, 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.*

We will mostly use this to talk about predicates that are satisfied by a subset of all possible justified messages of a protocol, e.g. belonging to a specific round of a protocol. This is covered by Definition 3 by making the class of messages part of the predicate P. Note that it is important for the definition to be meaningful that honest parties send their messages through a channel that leak them to the adversary, as otherwise an honest message that does not satisfy the P does not imply the adversary winning the game. For this reason it will be convenient when defining security properties to use all *possible justified outputs* of a justified protocol as shorthand for the outputs of honest parties as well as anything the adversary could convincingly claim to have gotten as output. This motivates Definition 4 which considers a predicate P only on outputs of the protocol Π, but then requires the outputs to be sent to all parties. As honest parties send their outputs, the adversary could easily win the game if an honest party receives output that does not satisfy P.

Definition 4 (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: Let Π' be the protocol Π with only change being that each party on getting output, sends their output to all parties if this was not already done. Run the protocol Π' under attack by the adversary. At some point the adversary may output a sequence of triples $(\mathsf{P}^1, \mathsf{mid}^1, m^1), \ldots, (\mathsf{P}^\ell, \mathsf{mid}^\ell, m^\ell)$. We say that the adversary wins if the $\mathsf{mid}^1, \ldots, \mathsf{mid}^\ell$ are identified with outputs of Π, $\mathsf{P}^1, \ldots, \mathsf{P}^\ell$ are honest (but not necessarily distinct) parties, 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.*

2.2 Justified Reliable Broadcast

We use the notion of reliable broadcast (RB) (cf. [Bra87]). For a message identifier mid we can have a possible corrupt sender S send a message m associated to mid such that all parties which receive m will receive the same m. Furthermore, if any honest party receives m then they all receive m. We stress that S does not need to be part of $\mathcal{C}$ and that while the security properties in the definition only apply to $\mathcal{C}$ and S, we will later implement a protocol which satisfies the definition but additionally requires participation from a different set of parties $\mathcal{P}$.

Definition 5 (Justified Reliable Broadcast). *A protocol Π for $n_\mathcal{C}$ parties $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_{n_\mathcal{C}}\}$, where all parties have input mid. The message identifier mid contains the identity of a sender 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 S at the time of input.*

Validity: *If honest S has input (mid, m) and an honest C_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 S is honest and has input $(\mathsf{mid}, \cdot)$, and all honest P_j start running the protocol, then eventually all honest C_i have output $(\mathsf{mid}, \cdot)$.*

Eventual Output 2: *If an honest* C_j *has output* $(\mathsf{mid}, \cdot)$*, and all honest parties start running the protocol then eventually all honest* C_i *have output* $(\mathsf{mid}, \cdot)$*.*

We will later give many different implementations of reliable broadcast. We will also show how to get atomic broadcast among $\mathcal{C}$ from reliable broadcast secure against $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ parties.

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* $n_{\mathcal{C}}$ *parties* $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$ *where the input of the parties is a fixed symbol* ELECT*. The output is* $j \in [n_{\mathcal{C}}]$*. There is an output justifier* J_{OUT} *specified by the protocol.*

Liveness: *If all honest parties start running the protocol with input* ELECT*, then eventually all honest parties* C_i *have an output.*
Validity: *For all possible justified outputs* j *it holds that* $j \in \{1, \ldots, n_{\mathcal{C}}\}$*.*
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* $1/n_{\mathcal{C}} + \mathrm{negl}$.

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 parties. To implement the Leader Election protocol which is defined for $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}\}$ as above, we will rely on $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ and honest majority in both $\mathcal{P}$ and $\mathcal{C}$. In some settings $\mathcal{P} = \mathcal{C}$ and this is a distinction without a difference, but in other settings we need to sample $\mathcal{C}$ at random from $\mathcal{P}$ and do not want to require a special setup between the sampled parties. We will refer to the following simple protocol as $\Pi_{\mathrm{CONSTANTINE}}$. On input $(\mathrm{ELECT}, \mathsf{sid})$ a party C_i sends an authenticated $(\mathrm{ELECT}, \mathsf{sid})$ to $\mathcal{P}$. On seeing $t_{\mathcal{C}}+1$ such messages party P_i sends its signature share on sid. Given $n - t$ signature shares a party 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. The output is justified by verifying σ_{sid} This costs communication $\mathcal{O}(n \cdot n_{\mathcal{C}} \cdot \kappa)$.

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 (CC). Each message m to be causal cast 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 $\mathsf{C}^{\mathsf{mid}}$. 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 parties 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 $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$ is called a causal cast (CC) if it has the following properties.*

Free-Choice Send: *A party C_i can have input $(\textsc{caucast-send}, \mathsf{mid}, m)$ where mid is a free choice identifier $\mathsf{C}_i = \mathsf{C}^{\mathsf{mid}}$ and ${J_{IN}}^{\mathsf{mid}}(m) = \top$ at $\mathsf{C}^{\mathsf{mid}}$ at the time of input.*

Computed-Message Send: *A party C_i can have input $(\textsc{caucast-send}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell)$, where mid is a computed-message identifier, $\mathsf{C}_i = \mathsf{C}^{\mathsf{mid}}$, C_i earlier gave outputs $(\textsc{caucast-del}, \mathsf{mid}_j, m_j)$ 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 party C_i can have input $(\textsc{caucast-send}, \mathsf{mid}, m)$ where mid is a constant identifier. In that case it is guaranteed that all parties eventually have the same input $(\textsc{caucast-send}, \mathsf{mid}, m)$.*

Free-Choice Validity: *A party C_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 C_i at the time of output. Furthermore, if $\mathsf{C}_j = \mathsf{C}^{\mathsf{mid}}$ is honest, then C_j had input $(\textsc{caucast-send}, \mathsf{mid}, m)$.*

Leader Election Validity: *A party C_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 $\Pi_{ELECT}.J^{\mathsf{sid}}_{OUT}(m)$ at C_i at the time of output.*

Computed-Message Validity: *A party C_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 C_i earlier gave outputs $(\textsc{caucast-del}, \mathsf{mid}_j, m_j, \ldots)$ 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 Validity: *A party C_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 party C_i had input $(\textsc{caucast-send}, \mathsf{mid}, \ldots)$ or some honest party had output $(\textsc{caucast-del}, \mathsf{mid}, \ldots)$ and all honest parties are running the system, then eventually all honest parties have output $(\textsc{caucast-del}, \mathsf{mid}, \ldots)$.*

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

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

Free-Choice Send: On input ($\text{CAUCAST-SEND}, \mathsf{mid}, m$) at C_i where mid is a free choice C_i will RB m with message identifier mid.

Computed-Message Send: On input ($\text{CAUCAST-SEND}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell$) at C_i, where mid is a computed-message identifier, C_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 $\mathsf{C}^{\mathsf{mid}}$ 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 ($\text{CAUCAST-DEL}, \mathsf{mid}, m$).

Computed-Message Deliver: On RB ($\mathsf{mid}_1, \ldots, \mathsf{mid}_\ell$) with message identifier mid from $\mathsf{C}^{\mathsf{mid}}$ wait until outputs ($\text{CAUCAST-DEL}, \mathsf{mid}_j, m_j$) 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 ($\text{CAUCAST-DEL}, \mathsf{mid}, m, \mathsf{mid}_1, \ldots, \mathsf{mid}_\ell$) if $m \neq \perp$.

Leader-Election Deliver: On output ($\text{ELECT}, \mathsf{sid}, K$) from Π_{ELECT} output ($\text{CAUCAST-DEL}, \mathsf{sid}, K$).

Constant Deliver: On input ($\text{CAUCAST-SEND}, \mathsf{mid}, m$) where mid is a constant identifier, output ($\text{CAUCAST-DEL}, \mathsf{mid}, m$).

The protocols presented in Section 3 will, with the exception of Π_{AAB}, all follow a similar pattern which we define as Justified Causal Cast. Often these protocols are chained together with the input of each being computed from the output of a previous protocol and only the first in the chain being given free-choice inputs satisfying some J_{IN} predicate. We will in sometimes write that a value is Causal Cast without specifying if it sent as a free-choice or a computed message. In those cases we let the type of message be decided by the justifier: i.e. if the value is justified by being computed from a set of previous messages, then it will be sent as a computed message; otherwise, it will be sent as free-choice message.

Definition 8 (Justified Causal Cast Protocols). *A Justified Causal Cast protocol is a special case of a Justified protocol as defined Definition 1, where all message are sent through Causal Cast. We let the justifier* J^{mid} *of a message* m *at* C_i *be that* ($\textit{CAUCAST-DEL}, \mathsf{mid}, m, \ldots$) *was output by* C_i. *The input* m *of each party, identified by* mid_{IN}, *is a message with* $J_{IN}(m) = J^{\mathsf{mid}_{IN}}(m) = \top$. *The output* m *of each party, identified by* mid_{OUT}, *can be sent as a computed message in which case* $J_{OUT}(m) = J^{\mathsf{mid}_{OUT}}(m) = \top$. *In particular this means that even when a party does not send its output, it is computed as a function,* $\mathsf{NextMessage}^{\mathsf{mid}_{OUT}}$, *of previously Causal Cast messages.*

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 parties P_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 $n_{\mathcal{C}}, n \in \text{poly}(\kappa)$ 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 $n_\mathcal{C} - t_\mathcal{C}$ outputs of $n_\mathcal{C}$ possible messages with session identifiers $\mathsf{sid}_1, \ldots, \mathsf{sid}_{n_\mathcal{C}}$ known by all parties. In these cases we can send an $n_\mathcal{C}$-bit vector indicating the $n_\mathcal{C} - t_\mathcal{C}$ session identifiers to use. Then the communication is only $\mathcal{O}(\kappa + n_\mathcal{C}) = \mathcal{O}(\kappa)$ 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 CC has (expected communication) complexity*

$$\mathcal{O}(c_1 \text{ IN} + c_2 \text{ RB} + c_3 \text{ RB}_{\#} + c_4 \text{ELECT} + c_5)$$

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 RB [4] *as inputs, c_2 is the total number of bits that the protocol needs to RB as intermediary values and outputs, c_3 is the* number *of RB 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 parties.*

The reason why we single out the complexity of inputs is that when inputs are given as computed messages, c_1 IN only needs to accommodate the description of the set of messages used to compute the inputs.

3 Honest Majority Asynchronous Atomic Broadcast from Reliable Broadcast

We now present a protocol for atomic broadcast. As noted in Section 2, we will describe this protocol for a set of parties $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_{n_\mathcal{C}}\}$ of which we assume at most $t_\mathcal{C} < n_\mathcal{C}/2$ are corrupted.

Definition 10 (Atomic Broadcast). *A protocol for $n_\mathcal{C}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_\mathcal{C}}$. There is an input justifier J_{IN}. Each C_i holds a list Ledger_i.*

Input: P_i *get inputs m where $J_{\text{IN}}(m) = \top$. We use* Scheduled *to denote the set of (i, m) such that m was input at an honest P_i. We use $\mathsf{Scheduled}^\tau$ to denote the value of this set at time τ.*
Liveness: *For all honest C_i and all times τ it holds eventually that $\mathsf{Scheduled}^\tau \subseteq \mathsf{Ledger}_i$.*
Monotone: *For all C_i and $\tau' \geq \tau$ it holds that $\mathsf{Ledger}_i^\tau \sqsubset \mathsf{Ledger}_i^{\tau'}$.*
Agreement: *For all C_i and C_j it holds that $\mathsf{Ledger}_i^\tau \sqsubset \mathsf{Ledger}_j^\tau$ or $\mathsf{Ledger}_j^\tau \sqsubset \mathsf{Ledger}_i^\tau$.*

In this section we build an Atomic Broadcast which has communication complexity $\mathcal{O}(\beta \cdot \text{RB})$ when there is enough traffic, as discussed in detail later.

3.1 Justified Gather

We describe and analyse our Justified Gather protocol. We consider protocols where each party 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 $U = \{(\mathsf{C}_j, B_j)\}_{\mathsf{C}_j \in P}$, where $P \subseteq \mathcal{C}$ and $|P| \geq n_\mathcal{C} - t_\mathcal{C}$.*

[4] I.e., the sum of the length of messages input to a RB.

We think of $(\mathsf{C}_j, B_j) \in U$ as C_j having input B_j. The Gather primitive says that each party C_i has a block set U_i as output and there is a large common core, i.e., a set U of size $n_{\mathcal{C}} - t_{\mathcal{C}}$ which is a subset of each U_i. So all parties to some extend agree on a large set U, but some C_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 $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest C_i have an input B_i for which $J_{\text{IN}}(B_i) = \top$ at C_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 parties have a J_{OUT}-justified output.*
Justified Blocks: *For all possible justified outputs U and all (potentially corrupt) C_i and all $(\mathsf{C}_i, B_i) \in U$ it holds that $J_{\text{IN}}(B_i) = \top$.*
Validity: *For all possible justified outputs U and all honest C_i and all $(\mathsf{C}_i, B_i) \in U$ it holds that C_i had input B_i.*
Agreement: *For all possible justified outputs U and U' and all $(\mathsf{C}_i, B_i) \in U$ and $(\mathsf{C}_i, B'_i) \in U'$ it holds that $B_i = B'_i$.*
Large Core: *For all possible justified outputs $(U^1, \ldots, U^m)$ it holds that $|\bigcap_{k=1}^{m} U^k| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.*

We present a Justified Gather protocol, Π_{GATHER}, in Fig. 1. The protocol follows the structure of the get-core protocol presented by Attiya and Welch in [AW04] and attributed to Gafni. The get-core protocol tolerates crash failures of up to half of the parties and works as follows: All parties gossip an input (U^0) and then in two rounds gather sets of inputs from $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties take the union (U^1 and U^2) and gossips it. When taking the union of the U^2 sets it can be shown that all resulting sets (U^3) have a common core of size $n_{\mathcal{C}} - t_{\mathcal{C}}$. The proof goes by arguing that the U_i^1 set of some party, C_i must be included in the majority of U^2 sets and thus in all U^3 sets. The get-core protocol was later adapted to the Byzantine setting by Dolev and Gafni in [DG16] using an abstraction similar to Causal Cast. However, the proof relies on an honest supermajority sending U sets. We stress that the proof of Theorem 1 implies that the protocol presented in [DG16] would also be secure against a minority of Byzantine parties when given a RB functionality.

1. The input of C_i is B_i with $J_{\text{IN}}(B_i) = \top$. Party C_i lets $U_i^0 = \{(\mathsf{C}_i, B_i)\}$. The singleton set is justified by B_i satisfying J_{IN}.
2. For $r \in [1;3]$ Party C_i causal casts U_i^{r-1}[a] and collects incoming U_j^{r-1} from parties C_j, lets P_i^r be the set of C_j it heard from, waits until $|P_i^r| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$, and lets
$$U_i^r = \bigcup_{\mathsf{C}_j \in P_i^r} U_j^{r-1}.$$
The set is justified by being computed from the set P_i^r where $|P_i^r| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.
3. Party C_i outputs U_i^3.

[a] Whether U_i^{r-1} is causal cast as a free-choice or computed message is determined by its justifier.

Fig. 1. Protocol Π_{GATHER}.

We show that Π_{GATHER} is a Justified Gather protocol. The proof largely follows the idea from [AW04] of describing a table and counting entries with ones. While they only consider crash failures, we allow Byzantine corruptions. However, since the all the messages are justified and sent through

Causal Cast, the adversary must either follow the protocol or stay silent. This allows the proof from [AW04] to go through with the original combinatorial argument, but a slightly different interpretation of the table. Dolev and Gafni, who previously adapted the protocol to tolerate Byzantine corruptions in [DG16], altered the table to only have a row and column for each honest party, which gives a simpler proof but also means that it needs an honest supermajority to go through.

Theorem 1. *If $t_\mathcal{C} < n_\mathcal{C}/2$ then Π_{GATHER} is a Justified Gather. If $\beta = \sum_{i=1}^{n_\mathcal{C}} |B_i|$ then it has complexity $\mathcal{O}(\beta\,\text{IN} + n_\mathcal{C}\,\text{RB}_\# + {n_\mathcal{C}}^2\,\text{RB})$.*

Proof. We first address the complexity. In the first round $n_\mathcal{C}$ parties each causal cast their input, possibly as a computed message. This contributes $\beta\,\text{IN} + n_\mathcal{C}\,\text{RB}_\#$. Then in a constant number of rounds each party RBs a set described by $n_\mathcal{C}$ bits, adding $\mathcal{O}(n_\mathcal{C}\,\text{RB}_\# + {n_\mathcal{C}}^2\,\text{RB})$ and resulting in total of $\mathcal{O}(\beta\,\text{IN} + n_\mathcal{C}\,\text{RB}_\# + {n_\mathcal{C}}^2\,\text{RB})$.

Liveness, Justified Blocks, Validity, and Agreement are all trivial so we only show Large Core. Initially singleton sets of inputs, U^0, are sent through causal cast with the only restriction being that the block satisfies J_{IN}. If the message of a corrupted party reaches any honest party then it reaches all honest parties and satisfies J_{IN}. As this is all we require of a message from an honest party, the only way to deviate from the protocol is to not send a valid message. Similarly, in the following rounds $r \in [1;2]$ the accumulated set U^r is sent through causal cast as a computed message based on the messages from previous round. So, the adversary must choose to either stay silent or send a message identical to what would have been sent by an honest party if they had received messages from the claimed set of parties P^r. The result now follows from the counting argument in [AW04]. We show that there is at least one party C_i whose U_i^1 set is included in all *possible justified outputs*. This is sufficient for the Large Core property as all justified U^1 sets include $n_\mathcal{C} - t_\mathcal{C}$ input values.

Let T be an $n_\mathcal{C}$ by $n_\mathcal{C}$ table. For each row i: if C_i sends a message U_i^2 which at some point is received by an honest party, then each entry $T[i,j]$ is one if $\mathsf{C}_j \in P_i^2$ and zero otherwise. Alternatively: U_i^2 is never received by an honest party and we let $T[i,j]$ be one if and only if U_j^1 is eventually received by an honest party. Since all rows contain at least $n_\mathcal{C} - t_\mathcal{C}$ ones, there are at least $n_\mathcal{C}(n_\mathcal{C} - t_\mathcal{C})$ ones in the table. So, at least one of the $n_\mathcal{C}$ columns, k, must contain $n_\mathcal{C} - t_\mathcal{C}$ ones. This means that set of parties whose U^2 set will eventually be received by an honest party and which does not include U_k^1 has size at most $t_\mathcal{C}$. Call this set of parties S. By Computed-Message Validity, if a message which is never received by an honest party is referenced in a computed message, then that computed message is justified in the view of an honest party. In particular if U_i^2 is never received by an honest party, and C_j sends U_j^3 justified by P_j^3 which points to U_i^2, then U_j^3 will never be justified in the view of an honest party.[5] So, when in the next round any *possible justified output* U^3 is justified by taking the union of $n_\mathcal{C} - t_\mathcal{C}$ U^2 sets, at least $n_\mathcal{C} - t_\mathcal{C} - |S| \geq n_\mathcal{C} - 2t_\mathcal{C} > 1$ of them will not be in S, i.e. it will contain U_k^1. Thus U_k^1 is a subset of all *possible justified outputs* U^3 and these all satisfy Large Core.

3.2 Justified Graded Gather

We describe and analyse our Justified Graded Gather protocol. This is just a Justified Gather, where each party also has knowledge about the common core. Each C_i will output a set T_i of size at least $n_\mathcal{C} - t_\mathcal{C}$ which is guaranteed to be a subset of the common core U.

[5] Note that Definition 4 only concerns outputs that can be sent and satisfy J_{OUT}.

Definition 13 (Justified Graded Gather). *A protocol for $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest C_i have an input B_i for which $J_{\text{IN}}(B_i) = \top$ at C_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 parties have a J_{OUT}-justified output.*

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

Sub Core: *For all possible justified outputs $((U^1, T^1), \ldots, (U^m, T^m))$ it holds that $T^i \subseteq \bigcap_{k=1}^m U^k$ for all $i \in [m]$.*

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

Agreement: *For all possible justified outputs (U, T) and (U', T') and all $(\mathsf{C}_i, B_i) \in S$ and $(\mathsf{C}_i, B_i') \in U'$ 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 $|\bigcap_{k=1}^m T^k| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.*

1. The input of C_i is B_i with $J_{\text{IN}}(B_i) = \top$. All parties run Π_{Gather} with C_i inputting B_i justified by J_{IN}. Let the output of C_i be U_i'.
2. Party C_i causal casts U_i' as a computed-message justified by $\Pi_{\text{Gather}}.J_{\text{OUT}}$ and collects justified U_j' from parties C_j, lets P_i be the set of C_j it heard from and waits until $|P_i| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.
3. Party C_i outputs
$$(U_i,\ T_i) = \Big(\bigcup_{\mathsf{C}_j \in P_i} U_j' \ , \quad \bigcap_{\mathsf{C}_j \in P_i} U_j' \Big) \ .$$
The outputs are justified by being computed as above from justified sets.

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

Theorem 2. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then $\Pi_{\text{GradedGather}}$ is a Justified Graded Gather. If $\beta = \sum_{i=1}^{n_{\mathcal{C}}} |B_i|$ and $\Pi_{\text{GradedGather}}$ uses Π_{Gather} from Fig. 1 as sub-protocol, then it has complexity $\mathcal{O}(\beta\ \mathit{IN} + {n_{\mathcal{C}}}^2\ \mathit{RB} + n_{\mathcal{C}}\ \mathit{RB}_{\#})$.*

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

Lemma 1 (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 $(\mathsf{C}_k, B_k) \in T_i$ then $(\mathsf{C}_k, B_k) \in U_\iota$. If $(\mathsf{C}_k, B_k) \in T_i$ then $\mathsf{C}_k \in U_j'$ for all $\mathsf{C}_j \in P_i$. Since $|P_i| \geq n_{\mathcal{C}} - t_{\mathcal{C}} \geq t_{\mathcal{C}} + 1$ it follows that any party receiving $n_{\mathcal{C}} - t_{\mathcal{C}}$ justified sets U_j' will also receive a set U_j' with $\mathsf{C}_k \in U_j'$. Namely, the sets are reliably broadcast so if two parties receive justified U_j' and $\hat{U}_j'$ then $U_j' = \hat{U}_j'$. Since all parties collect $n_{\mathcal{C}} - t_{\mathcal{C}}$ justified sets $U_\cdot^\ell$ to justify $U_\iota = \bigcup_{\mathsf{C}_j \in P_\iota^\ell} U_j^\ell$ it follows that $(\mathsf{C}_j, B_k) \in U_\iota$. □

3.3 Justified Graded Block Selection

We now present our graded block selection protocol. Here each party 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 parties can pick their own input and we allow that $C_i = B_i$ for a corrupt C_i we simply define validity by saying that the output should be some justified input. Note that this implies that if there is only 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 $n_{\mathcal{C}}$ parties $\mathsf{C}_1,\ldots,\mathsf{C}_{n_{\mathcal{C}}}$. There is an input justifier J_{IN} and an output justifier J_{OUT} specified by the protocol. All honest C_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 parties start running the protocol with a J_{IN}-justified input then eventually all honest parties have a J_{OUT}-justified output.*

Justified Output: *$J_{\text{IN}}(C_i) = \top$ holds for all possible J_{OUT}-justified outputs $(C_i,\cdot)$.*

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 $\alpha - \text{negl}$ all possible justified outputs of at least $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties will have grade $g_i = 2$.*

Stability: *If there are possible justified outputs $(C_i,\cdot)$ and $(C_j,\cdot)$ with $C_i \neq C_j$ then there exist two justified inputs B_i and B_j with $B_i \neq B_j$.*

1. The input of C_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. The parties run $\Pi_{\text{GradedGather}}$ with input B_i and input justifier J_{IN}. Let the output of C_i be (U_i, T_i) and causal cast this as a computed message.[a]
3. On receiving (U_j, T_j) from $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties $\mathsf{C}_j \in P_i$, the parties run a justified leader election to elect a justified king C_k.
4. Party C_i outputs

$$(C_i, g_i) = \begin{cases} (B_k, 2) & \text{if } \exists(\mathsf{C}_k, B_k) \in T_i \\ (B_k, 1) & \text{if } \exists(\mathsf{C}_k, B_k) \in U_i \setminus T_i \\ (B_i, 0) & \text{if } \nexists(\mathsf{C}_k, \cdot) \in U_i \ . \end{cases}$$

The output is justified by being computed as above from justified values.

[a] The point of this message is not to distribute the sets, but to commit $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties to their output of $\Pi_{\text{GradedGather}}$ before an honest party initiates leader election.

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

Theorem 3. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then $\Pi_{\text{GradedSelectBlock}}$ is a Justified Graded Block Selection. When $\beta = \sum_{i=1}^{n_{\mathcal{C}}} |B_i|$ and when using $\Pi_{\text{GradedGather}}$ from Fig. 2 as sub-protocol the complexity is $\mathcal{O}(\beta\,\text{IN} + {n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$.*

Proof. We start with the complexity. $\Pi_{\text{GradedGather}}$ has complexity $\mathcal{O}(\beta\,\text{IN} + {n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#})$. In addition to this $\Pi_{\text{GradedSelectBlock}}$ only does one leader election. It has to send no more causal cast information as the justifier for (C_i, g_i) is the justified B_i, the justified (U_i, T_i), and the justified C_k, which have all been causal cast 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 $(\mathsf{C}_k, C_k) \in U_i$ 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 $(\mathsf{C}_k, C_i) \in T_k$ for some justified T_k. Therefore, by Sub Core, $(\mathsf{C}_k, C_i) \in U_j$, and therefore $g_j \geq 1$. Assume then that $g_i, g_j > 0$. In that case $(\mathsf{C}_k, C_i) \in U_i$ and $(\mathsf{C}_k, C_j) \in U_j$, so by Agreement of the Graded Gather it follows that $C_i = C_j$. We then argue Positive Agreement for $\alpha = 1/2$. Consider the first honest C_i to start running the leader election. When this happens C_i already received (U_j, T_j) from $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties C_j. By Large Sub Core of $\Pi_{\text{GradedGather}}$ these T_j sets have an intersection of size at least $n_{\mathcal{C}} - t_{\mathcal{C}}$. Since $|\bigcap_{\mathsf{C}_j \in P_i} T_i| \geq n_{\mathcal{C}} - t_{\mathcal{C}} > n_{\mathcal{C}}/2$ it follows that $\mathsf{C}_k \in T_j$ for all $\mathsf{C}_j \in P_i$ with probability at least $1/2 - \text{negl}$. Whenever this happens, no party in P_i can justify an output with grade less than 2. To argue Stability just note that it holds for both C_i and C_j that they are justified inputs of $\Pi_{\text{GradedSelectBlock}}$. □

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 $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$. There is an input justifier J_{IN} and an output justifier J_{OUT}. All honest C_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 parties start running the protocol with a J_{IN}-justified input then eventually all honest parties 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 C_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 $g_i^0 = 0$, which is justified if $J_{\text{IN}}(B_i^0) = \top$ and $g_i^0 = 0$.
3. For rounds $r = 1, \ldots$ each party C_i with $\mathsf{GaveOutput}_i = \bot$ runs $\Pi_{\text{GradedSelectBlock}}$ where:
   (a) C_i has input B_i^{r-1}.
   (b) The input of C_i is justified by a justified (B_i^{r-1}, g_i^{r-1}) where $g_i^{r-1} < 2$.
   (c) C_i eventually gets justified output (B_i^r, g_i^r).[a]
4. In addition to the above loop each C_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 $C_i = B_i^r$. Causal cast the output as a computed message using 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 $\mathsf{C}_j \neq \mathsf{C}_i$ 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).

[a] Recall that by Definition 8 this output can be sent as a computed message.

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

Theorem 4. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then $\Pi_{\text{SelectBlock}}$ is a Justified Select Block Protocol. When $\beta = \sum_{i=1}^{n_{\mathcal{C}}} |B_i|$ and using the protocol $\Pi_{\text{GradedSelectBlock}}$ from Fig. 3 as sub-protocol the complexity is $\mathcal{O}(\beta\,\text{IN} + {n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$.*

Proof. We start with the complexity. The first run of $\Pi_{\text{GradedSelectBlock}}$ has complexity $\mathcal{O}(\beta\,\text{IN} + {n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$, and each following run has complexity $\mathcal{O}({n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$, where we ignore the IN component as the size of the message identifiers needed to send the outputs as computed messages ($n_{\mathcal{C}}$ bits for each) is dominated by other costs. Besides this the protocol only causal casts computed values for which the receiver knows the message identifier, so there is no more information to causal cast. 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 all possible justified outputs from at least $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties of the r^{th} iteration of $\Pi_{\text{GradedSelectBlock}}$ will have $g_i^r = 2$. These parties cannot give justified input to $\Pi_{\text{GradedSelectBlock}}$ in round $r + 1$, which means that it will deadlock. Additionally one of these parties is honest and 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 C_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 party. 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 party C_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 party propose a set and then picks $n_{\mathcal{C}} - t_{\mathcal{C}}$ of them.

Definition 16 (Justified Agreement on a Core Set). *A protocol for $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$ with input and output justifiers J_{IN} and J_{OUT}. All honest C_i have an input B_i for which $J_{\text{IN}}(B_i) = \top$ at the time of input.*

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

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

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

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 $|S| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.*

Theorem 5. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then Π_{ACS} is a JACS protocol. When $\beta = \sum_{i=1}^{n_{\mathcal{C}}} |B_i|$ and when using $\Pi_{\text{SelectBlock}}$ from Fig. 4 as sub-protocol the complexity is $\mathcal{O}(\beta\,\text{IN} + {n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$.*

1. The input of C_i is B_i with $J_{\text{IN}}(B_i) = \top$.
2. Party C_i causal casts B_i as a free-choice message. This message is justified by $J_{\text{IN}}(B_i) = \top$.
3. Party C_i collects at least $n_{\mathcal{C}} - t_{\mathcal{C}}$ justified B_j from parties $\mathsf{C}_j \in P_i$ and lets $U_i = \{(\mathsf{C}_j, B_j)\}_{\mathsf{C}_j \in P_i}$. This value is justified by each B_j being justified and $|P_i| \geq n_{\mathcal{C}} - t_{\mathcal{C}}$.
4. Run $\Pi_{\text{SelectBlock}}$ where C_i inputs U_i. The input justifier of $\Pi_{\text{SelectBlock}}$ is that U_i is computed from P_i as in the above step.
5. Party C_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}

Proof. Safety and liveness properties follows directly from those of $\Pi_{\text{SelectBlock}}$. We address the complexity. The causal cast of all blocks B_i costs $\beta\,\text{IN}$. Causal casting the inputs to $\Pi_{\text{SelectBlock}}$, U_i, costs ${n_{\mathcal{C}}}^2\,\text{RB}$ to specify the sets P_i. Running $\Pi_{\text{SelectBlock}}$ then costs an additional expected $\mathcal{O}({n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$, where we ignore the IN component as it is run on computed messages. There are no further costs. ☐

3.6 Atomic Broadcast

We now give a protocol for atomic broadcast assuming reliable broadcast. It allows an arbitrary set of parties of polynomial size in κ to add transactions to a an ordered eventually consistent list Ledger. [6] 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 FIFO delivery on reliable broadcast of blocks with non-duplicate transaction identifiers.*

- *In the Π_{AAB} protocol the parties will send around so-called* transaction identifiers (j, c_j) *identifying transaction number c_j by S_j. We say that (j, c_j) is justified at C_i if it saw that some (j, c_j, m_j) was reliable broadcast by S_j and $c_j = 1$ or $(j, c_j - 1)$ is similarly justified at C_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 $\mathsf{Msg}(j, c_j) = m_j$.*
- *Parties C_i will send out* blocks B_i^e, *where e is an* epoch number *and B_i^e is a set of transaction identifiers (j, c_j). The set B_i^e is justified at C_k if all $(j, c_j) \in B_i^e$ are justified and not included in $B_i^{e'}$ for any $e' < e$ and C_k already received previous block B_i^{e-1} which is similarly justified.*[7] *Justified blocks define message sets as follows. Let $\mathsf{Msg}(B_i^0) = \emptyset$ and for $e > 0$ and justified B_i^e let $\mathsf{Msg}(B_i^e) = \{\mathsf{Msg}(j, c_j) | (j, c_j) \in B_i^e\} \cup \mathsf{Msg}(B_i^{e-1})$.*

Theorem 6. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then protocol Π_{AAB} is an atomic broadcast. 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 $n_{\mathcal{C}} \log(\kappa)$, the communication complexity is*

$$\mathcal{O}\left(\beta\,RB + \iota \cdot RB_{\#} + \rho \cdot \left({n_{\mathcal{C}}}^2\,RB + n_{\mathcal{C}}\,RB_{\#} + ELECT\right)\right) \;.$$

[6] If these parties are a subset of $\mathcal{C}$, then the AB can be constructed simply by running Π_{ACS} with transactions as input, outputting the result as an extension to Ledger and then repeating.

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

Init: All C_i lets $\mathsf{Ledger}_i = ()$, $\mathsf{Scheduled}_i = \emptyset$, $\mathsf{OnGoing}_i = \bot$, and $e_i = 0$. Define $B_i^0 = \emptyset$ and that B_i^0 has been CC received from C_i by all C_j already. For all $\mathsf{P}_j \in \mathcal{P}$ let $c_j = 0$.

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

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

Propose Next Block: Party C_i: If $\mathsf{OnGoing}_i = \bot$ and $\mathsf{Scheduled}_i \setminus (\mathsf{Msg}(B_i^{e_i}) \cup \mathsf{Ledger}_i) \neq \emptyset$, let $\mathsf{OnGoing}_i = \top$ and atomically do the following:[a]

1. Let $e_i \leftarrow e_i + 1$.
2. Let $B_i^{e_i} = \mathsf{Scheduled}_i \setminus (\mathsf{Msg}(B_i^{e_i}) \cup \mathsf{Ledger}_i)$.
3. Start ${\Pi_{\text{ACS}}}^{e_i}$ with input $B_i^{e_i}$ and the input justifier being that it is a valid block composed of message identifiers as explained in Definition 17.[b]

Extend Ledger: If $\mathsf{OnGoing}_i = \top$ and ${\Pi_{\text{ACS}}}^{e_i}$ produces output $U^{e_i} = \{(\mathsf{C}_j, B_j^{e_i})\}_{\mathsf{C}_j \in P}$, then let $M^{e_i} = \cup_{\mathsf{C}_j \in P} \mathsf{Msg}(B_j^{e_i})$, 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 $\mathsf{Scheduled}_i \setminus (\mathsf{Msg}(B_i^{e_i}) \cup \mathsf{Ledger}_i)$ grow to some bigger size.

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

Fig. 6. Atomic Broadcast Protocol Π_{AAB}.

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 by a simple induction. We then look at liveness. Liveness of RB means that (j, c_j) eventually ends up in $\mathsf{Scheduled}_i$ at all honest C_i. In the next run of Π_{ACS} message m_j will then be in $\mathsf{Msg}(B_j^{e_j})$ for all honest $\mathsf{C}_j \in P$. Since $|P| = n_{\mathcal{C}} - t_{\mathcal{C}} \geq t_{\mathcal{C}} + 1$ there is an honest C_j in P, so m will be in M^{e_i} and then Ledger_i. We count complexity. Each input is reliably broadcast. This is $\mathcal{O}(\beta\,\text{RB} + \iota\,\text{RB}_{\#})$. To causal cast the blocks in Π_{ACS} we only need to causal cast 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 $n_{\mathcal{C}}$ 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 $\sum_{i,e} |\text{enc}(B_i^e)| = \mathcal{O}(\iota n_{\mathcal{C}} \log(\kappa))$. This overall adds $\mathcal{O}(\iota n_{\mathcal{C}} \log(\kappa)\,\text{RB})$. We assumed that inputs have average length at least $n_{\mathcal{C}} \log(\kappa)$. Therefore $\iota n_{\mathcal{C}} \log(\kappa) = \mathcal{O}(\beta)$, so the contribution is $\mathcal{O}(\beta\,\text{RB})$. In each of ρ epochs, C_i inputs B_i to Π_{ACS} which contributes an extra $\mathcal{O}({n_{\mathcal{C}}}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT})$. ☐

4 Optimally Resilient Subquadratic AAB

We now present an Asynchronous Atomic Broadcast protocol for n parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ with subquadratic communication and optimal resilience against $t < n/3$ Byzantine corruptions. It uses a committee $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}\}$ in which assume $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ parties are corrupted. This can be realised by sampling $\mathcal{C}$ from $\mathcal{P}$ as we discuss in Section 4.1. We will refer to $\mathcal{P}$ as the ground population, parties $\mathsf{P} \in \mathcal{P}$ as ground members, $\mathcal{C}$ as the committee, and parties $\mathsf{C} \in \mathcal{C}$ as committee members.

The blueprint for the construction is to have the committee run Π_{AAB} and then disseminate the results to the ground population. However, Π_{AAB} relies on RB and because we want optimal resilience there is only a constant probability that $\mathcal{C}$ has the honest supermajority needed for asynchronous RB. For this reason the parties in $\mathcal{P}$ will be assisting $\mathcal{C}$ in the implementation of RB for $\mathcal{C}$ in Section 4.2. Finally, in Section 4.3 we give a protocol that allows $\mathcal{C}$ to distribute extensions of the ledger to $\mathcal{P}$, which provides the last piece of the puzzle for subquadratic AAB for $\mathcal{P}$ which we analyse in Section 4.4.

4.1 Sampling a Committee with Honest Majority

As demonstrated in [ACKN23], when a protocol requires honest supermajority and a subprotocol only needs honest majority, the subprotocol can be delegated securely to a randomly sampled committee with reasonable constants, assuming corruptions are static. The basic idea is that from the n parties $\mathcal{P}$ one samples a uniformly random subset $\mathcal{C}$ of size $n_{\mathcal{C}} = \Theta(\kappa)$. Since we are sampling from a set with a supermajority of honest parties and only need that $\mathcal{C}$ has a majority of honest parties the size of $\mathcal{C}$ can be practical. In our setting the set $\mathcal{C}$ could be constructed simply by running $n_{\mathcal{C}} = |\mathcal{C}|$ justified leader elections in parallel and wait for all of them and let $\mathcal{C}$ be the $n_{\mathcal{C}}$ winners. A party elected multiple times can be handled by proceeding with a smaller $\mathcal{C}$ or letting the party run multiple parties. 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 $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ corruptions in $\mathcal{C}$. For now we just remark that if committees are sampled with replacement from $\mathcal{P}$ 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 and optimal resilience of $t < n/3$.

To allow comparison against other works that get subquadratic communication via random committees without optimal resilience, we compute (cf. Appendix A) the minimal secure committee sizes for various choices of statistical security parameter (σ) and maximal fraction of corruption tolerated among $\mathcal{P}$ (c). The method used is simply checking that the cumulative distribution function of the honest parties getting elected for a minority of the roles on the committee is bounded by $2^{-\sigma}$. The results are given 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 well 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
$n_{\mathcal{C}}$	81	127	307	111	173	423	173	269	653	235	363	887

Table 1. Minimal committee sizes ($n_{\mathcal{C}}$) 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.

4.2 Subquadratic Reliable Broadcast

We now present a reliable broadcast protocol for $n_{\mathcal{C}}$ parties $\mathcal{C}$ implemented using $\mathcal{P}$. We assume that at most $t < n/3$ parties in $\mathcal{P}$ are corrupt and at most $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ parties in $\mathcal{C}$ are corrupt.

Setup: We assume a setup for a threshold signature scheme with verification key vk being public and each P_i holding key share sk_i. The reconstruction threshold is $n - t$. The protocol also use a Merkle-tree scheme $(\mathsf{Acc}, \mathsf{Wit}, \mathsf{Mem})$ and uses an erasure code $\mathsf{EC} = (\mathsf{Enc}, \mathsf{Dec})$ with $n_{\mathcal{C}}$ code words and reconstruction threshold $n_{\mathcal{C}} - t_{\mathcal{C}}$.

Input: Designated sender S: On input (mid, m) with $J^{\mathsf{mid}}(m) = \top$ let $M = \{i, m_i\}_{[n_{\mathcal{C}}]} = \mathsf{EC.Enc}(m)$, let $z = \mathsf{Acc}(ak, M)$, and send (mid, z) to all P_k.

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

Send Shards: S: On having received $n - t$ valid signature shares compute $\sigma = \mathsf{Sig}_{\mathsf{sk}}((\mathsf{mid}, z))$ using $\mathsf{Combine}$ and send $(\mathsf{mid}, z, \sigma, m_i, w_i = \mathsf{Wit}(ak, z, m_i))$ to C_i.

Echo and Record own Shard: C_i: On having received $(\mathsf{mid}, z, \sigma, m_i, w_i)$ with $\mathsf{Ver}_{\mathsf{vk}}((\mathsf{mid}, z), \sigma) = \top$ and $\mathsf{Mem}(ak, z, w_i, m_i) = \top$ from S or some party $\mathsf{C}_j \in \mathcal{C}$ send $(\mathsf{mid}, z, \sigma, m_i, w_i)$ to all other parties and record $(\mathsf{mid}, z, i, m_i)$.

Record other Shards: $\mathsf{C}_j \in \mathcal{C}$: On having received $(\mathsf{mid}, z, \sigma, m_i, w_i)$ with $\mathsf{Ver}_{\mathsf{vk}}((\mathsf{mid}, z), \sigma) = \top$ and $\mathsf{Mem}(ak, z, w_i, m_i) = \top$ from C_i record $(\mathsf{mid}, z, i, m_i)$. Note that we deliberately do not echo here.

Combine Shards: $\mathsf{C}_j \in \mathcal{C}$: On having recorded $(\mathsf{mid}, z, i, m_i)$ for $n_{\mathcal{C}} - t_{\mathcal{C}}$ parties C_i for the same mid, call the set of these parties S, compute $m = \mathsf{EC.Dec}(\{(i, m_i)\}_{\mathsf{C}_i \in S})$ and $M = \mathsf{EC.Enc}(m)$ and $z' = \mathsf{Acc}(ak, M)$. If $z' = z$ then output (mid, m) and for each $\mathsf{C}_i \in \mathcal{C}$ compute $w_i = \mathsf{Wit}(ak, z, m_i)$ and send $(\mathsf{mid}, z, \sigma, m_i, w_i)$ to C_i.

Fig. 7. Π_{SQRB}: A protocol for reliable broadcast with designated sender S for $n_{\mathcal{C}}$ parties of which at most $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ are corrupt. It is implemented using n ground members of which at most $t < n/3$ are corrupt.

Theorem 7. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ and $t < n/3$, then Π_{SQRB} is a reliable broadcast for $\mathcal{C}$. If messages have length $\beta \geq \kappa$, then the complexity is $\mathcal{O}(\beta n_{\mathcal{C}} + n\kappa + {n_{\mathcal{C}}}^2\kappa)$, which is of the form $\mathcal{O}(\beta\,\textsc{rb} + \textsc{rb}_{\#})$ for $\textsc{rb} = n_{\mathcal{C}}$ and $\textsc{rb}_{\#} = n\kappa + {n_{\mathcal{C}}}^2\kappa$. For large populations $n \geq {n_{\mathcal{C}}}^2$ or large messages $\beta \geq n_{\mathcal{C}}\kappa$ the complexity is $\mathcal{O}(\beta n_{\mathcal{C}} + n\kappa)$ such that $\textsc{rb}_{\#} = n\kappa$.*

Proof. We argue agreement. If a party accepts $(\mathsf{mid}, z, \sigma, m_i, w_i)$ then it was signed by $n - t$ ground members and therefore at least $t + 1$ honest ground members. Therefore z is unique for mid. And if C_j outputs (mid, m) then $z = \mathsf{Acc}(ak, \mathsf{EC.Enc}(m))$ 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 C_j outputs (mid, m). Then it sends $(\mathsf{mid}, z, \sigma, m_i, w_i)$ to each C_i and therefore each honest C_i sends $(\mathsf{mid}, z, \sigma, m_i, w_i)$ to all parties in **Echo and Record own Shard**. There are $n_{\mathcal{C}} - t_{\mathcal{C}}$ honest parties doing this, so all honest will end up recording $n_{\mathcal{C}} - t_{\mathcal{C}}$ shards and output (mid, m) in **Combine Shards**. We count complexity. Getting the signature shares from n ground members P_j costs $n\kappa$. Each m_i and w_i is sent to and from C_i at most $\mathcal{O}(n_{\mathcal{C}})$ times which contributes $\mathcal{O}(n_{\mathcal{C}}|m_i| + n_{\mathcal{C}}\kappa)$. Summing over C_i this gives $\mathcal{O}(n_{\mathcal{C}}|m| + {n_{\mathcal{C}}}^2\kappa)$. For large ground populations $n \geq {n_{\mathcal{C}}}^2$ we have that $\mathcal{O}(\beta n_{\mathcal{C}} + n\kappa + {n_{\mathcal{C}}}^2\kappa) = \mathcal{O}(\beta n_{\mathcal{C}} + n\kappa + n\kappa) = \mathcal{O}(\beta n_{\mathcal{C}} + n\kappa)$. For large messages $\beta \geq n_{\mathcal{C}}\kappa$ we have that $\mathcal{O}(\beta n_{\mathcal{C}} + n\kappa + {n_{\mathcal{C}}}^2\kappa) = \mathcal{O}(\beta n_{\mathcal{C}} + n\kappa + n_{\mathcal{C}}\beta) = \mathcal{O}(\beta n_{\mathcal{C}} + n\kappa)$. □

Corollary 1. *Protocol Π_{AAB} when using Π_{ACS} for ACS, $\Pi_{\textsc{Constantine}}$ for leader election, and Π_{SQRB} for RB is an atomic broadcast for $\mathcal{C}$. Let β be the total length of inputs and let ι be the number of inputs. Then assuming that on average inputs have length at least $n_{\mathcal{C}} \log(\kappa)$ the communication complexity is*

$$\beta n_{\mathcal{C}} + \iota \cdot (n\kappa + {n_{\mathcal{C}}}^2\kappa) + \rho \cdot \left(n n_{\mathcal{C}}\kappa + {n_{\mathcal{C}}}^3\kappa\right) \ . \tag{1}$$

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

$$\beta n_{\mathcal{C}} + \iota n\kappa \ . \tag{2}$$

Proof. The complexity of Π_{AAB} is $\beta\,\textsc{rb} + \iota \cdot \textsc{rb}_{\#} + \rho \cdot (n_{\mathcal{C}}{}^2\,\textsc{rb} + \textsc{elect} + n_{\mathcal{C}}\,\textsc{rb}_{\#})$. Plug in $\textsc{rb} = n_{\mathcal{C}}$, $\textsc{rb}_{\#} = n\kappa + n_{\mathcal{C}}{}^2\kappa$, and $\textsc{elect} = nn_{\mathcal{C}}\kappa$ to get

$$\beta n_{\mathcal{C}} + \iota \cdot (n\kappa + n_{\mathcal{C}}{}^2\kappa) + \rho \cdot \left(nn_{\mathcal{C}}\kappa + n_{\mathcal{C}}{}^3\kappa\right) \ .$$

If on average we consume $n_{\mathcal{C}}$ messages per epoch then $\rho n_{\mathcal{C}} \leq \iota$, so we get

$$\rho \cdot \left(nn_{\mathcal{C}}\kappa + n_{\mathcal{C}}{}^3\kappa\right) \leq \iota \cdot (n\kappa + n_{\mathcal{C}}{}^2\kappa) \ .$$

So we can drop the LHS asymptotically. If $n \geq n_{\mathcal{C}}{}^2$ then $n\kappa \geq n_{\mathcal{C}}{}^2\kappa$. Equivalently, if $\beta/\iota \geq n_{\mathcal{C}}\kappa$ then $\beta n_{\mathcal{C}} \geq \iota \cdot n_{\mathcal{C}}{}^2\kappa$. In both cases we can drop $\iota \cdot n_{\mathcal{C}}{}^2\kappa$ asymptotically. □

4.3 Justified Outcast

To transform Π_{AAB} into an AB for the ground population, we need $\mathcal{P}$ to receive the updated state of the ledger from $\mathcal{C}$. We call this primitive outcast as it can be thought of as reliable broadcast of a message held by committee "out" to the ground population. We assume that all parties agree on the message m to be sent.

Definition 18 (Reliable Outcast). *A protocol for $n_{\mathcal{C}}$ parties $\mathsf{C}_1, \ldots, \mathsf{C}_{n_{\mathcal{C}}}$ and n ground members $\mathsf{P}_1, \ldots, \mathsf{P}_n$, where all parties have input* mid*. The parties additionally have an already agreed upon input $m \in \{0,1\}^*$.*

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

Outcasting is trivial when at most $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ parties are corrupt. First each party takes m and uses an erasure code to encode it as shards $(m_1, \ldots, m_{n_{\mathcal{C}}})$ such that it can be decoded using $n_{\mathcal{C}} - t_{\mathcal{C}}$ of the values m_i. Using standard techniques and assuming $|m| \geq n_{\mathcal{C}} \log n_{\mathcal{C}}$, this can be done with $\sum_{i=1}^{n_{\mathcal{C}}} |m_i| = \mathcal{O}(|m|)$. Let M be the set of these shards paired with their index, i.e. $M = \{m_i, i\}_{i \in [1;n_{\mathcal{C}}]}$. Then each party $\mathsf{C}_i \in \mathcal{C}$ computes $z = \mathsf{Acc}(ak, M)$ and $w_i = \mathsf{Wit}(ak, z, m_i)$. Party C_i sends to each P_j the digest z and m_i and a witness w_i showing that m_i is in M. Each ground member P_j waits for $n_{\mathcal{C}} - t_{\mathcal{C}}$ identical reports of z and adopts this value. It was sent by at least one honest party, so it must be correct. Then it waits for $n_{\mathcal{C}} - t_{\mathcal{C}}$ messages m_i along with proofs that they are in M. From these $n_{\mathcal{C}} - t_{\mathcal{C}}$ values it computes m. We call this protocol $\Pi_{\textsc{Outcast}}$. This costs communication $\mathcal{O}(n_{\mathcal{C}} n\kappa + n|m|)$. For $|m| \geq n_{\mathcal{C}}\kappa$ this is $\mathcal{O}(n|m|)$, which is asymptotically optimal as each ground member has to receive m.

4.4 Atomic Broadcast

We finally construct and analyse an AB protocol for the ground population. We will use the protocol Π_{AAB} 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. 8.

Theorem 8. *If $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ then protocol Π_{SQAAB} is an atomic broadcast. When β is the total bit-length of all inputs, ι is the number of inputs, and ρ is the number of epochs in Π_{AAB}, and when using Π_{AAB} from Fig. 6 and $\Pi_{\textsc{Outcast}}$ from Section 4.3 as sub-protocol, and assuming that on average inputs have length at least $\kappa + n_{\mathcal{C}} \log(\kappa)$ communication complexity is*

$$\mathcal{O}\left(\beta(\textit{RB} + n) + \iota \cdot \textit{RB}_{\#} + \rho \cdot \left(n_{\mathcal{C}}{}^2(\textit{RB} + n) + n_{\mathcal{C}}\,\textit{RB}_{\#} + \textit{ELECT}\right)\right) \ .$$

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

Fig. 8. Atomic Broadcast Π_{SQAAB}

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

$$\mathcal{O}\left(\beta\,\text{RB} + \iota \cdot \text{RB}_{\#} + \rho \cdot \left(n_{\mathcal{C}}{}^2\,\text{RB} + n_{\mathcal{C}}\,\text{RB}_{\#} + \text{ELECT}\right)\right)$$

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

Corollary 2. *Protocol Π_{SQAAB} when using Π_{AAB} from Corollary 1 and Π_{Outcast} for outcasting is an AB for $\mathcal{P}$. Let β be the total length of inputs and let ι be the number of inputs. Then assuming that on average inputs have length at least $n_{\mathcal{C}} \log(\kappa)$ the communication complexity is*

$$\beta n + \iota \cdot (n\kappa + n_{\mathcal{C}}{}^2\kappa) + \rho \cdot \left(n n_{\mathcal{C}}\kappa + n_{\mathcal{C}}{}^3\kappa\right)\ . \tag{3}$$

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

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

Proof. The contribution of outcasting is $\mathcal{O}(\beta n + \rho n n_{\mathcal{C}}\kappa)$. The term $\rho n n_{\mathcal{C}}\kappa$ 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 and AB with Dual Threshold and Asymmetric Synchrony Assumptions

We now present a RB for $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$. We will not focus on communication complexity but resilience. The protocol can be turned into a communication efficient reliable broadcast using threshold signatures and the sharding technique from Π_{SQRB}, 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 $t_{\text{A}} \leq t_{\text{S}} < n/2$ and where there are at most t_{S} corruptions.

The protocol uses one timeout per party—party C_i waits Δ^i_{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 Δ^i_{Wait}. However, if the actual number of corruptions t is in the interval $t_{\text{A}} < t \leq t_{\text{S}}$ then the running time of the protocol does depend on the Δ^i_{Wait}. The motivation for specifying a separate timeout for each party is that we observe that long as each honest party picks sufficiently long timeouts the protocol is secure. So these timeouts do not need to be global, as would be the usual case in the synchronous setting. However, one can of course as a special case consider the protocol in the usual synchronous model by assuming all parties are given the same global timeout.

The protocol can be proven secure in two settings. One is a setting where the network is always synchronous and where $t_{\text{S}} + 2t_{\text{A}} < n$. In this setting we need that $\Delta^i_{\text{Wait}} \geq 2\Delta_{\text{Net}}$. We call this the

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

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

1. Let $\sigma_i = \mathsf{Sig}_{\mathsf{sk}_i}((\text{ASYNC}, m))$ and send (m, σ_s, σ_i) to all parties.
2. Add $(\mathsf{C}_i, m, \sigma_i)$ to $\mathsf{SignedAsync}$.
3. Set a timeout $\mathsf{Timeout}(\text{COLLECTFROMHONEST}, \Delta^i_{\text{WAIT}})$.

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

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

Collect Synchronous Echos All parties: On receiving (m_j, σ_j) from C_j, where $\mathsf{Ver}_{\mathsf{vk}_j}((\text{SYNC}, m_j), \sigma_j) = \top$ and no such value was received from C_j before, add $(\mathsf{C}_j, m_j, \sigma_j)$ to $\mathsf{SignedSync}$.

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

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

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

- Σ contains $n - t_{\text{S}}$ values $(\mathsf{C}_j, m, \sigma_j)$ for distinct C_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\text{SYNC}, m), \sigma_j) = \top$ (call such a value synchronous-valid), or
- Σ contains $n - t_{\text{A}}$ values $(\mathsf{C}_j, m, \sigma_j)$ for distinct C_j such that $\mathsf{Ver}_{\mathsf{vk}_j}((\text{ASYNC}, m), \sigma_j) = \top$ (call such a value asynchronous-valid),

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

Fig. 9. Π_{RBD}: A protocol for RB with dual thresholds t_{S} and t_{A} with designated sender S. For conciseness we do not explicitly mention the message identifier mid and we let $J_{\text{IN}} = J^{\mathsf{mid}}$.

optimistic model. In the other setting we can tolerate that the network is sometimes asynchronous. Here we only need $\Delta^i_{\text{WAIT}} \geq \Delta_{\text{NET}}$. However, we need that $2t_{\text{S}} + t_{\text{A}} < n$. As a strengthening we can here tolerate that either the network is synchronous and there are at most $t \leq t_{\text{S}}$ corruptions or the network is asynchronous and there are at most $t \leq t_{\text{A}}$ corruptions. We call this the *network agnostic model* below.

These instantiations of Π_{RBD} imply that Π_{AAB} where the committee $\mathcal{C}$ is simply $\mathcal{P}$ and the RB is instantiated using Π_{RBD} with appropriate parameters is a secure AB for $\mathcal{C}$ in the two models respectively.

5.1 Asymmetric Synchrony Assumptions

We construct our ABs 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 parties $\mathsf{C}_1, \dots, \mathsf{C}_n$ 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 C_i can set a sound timeout length Δ^i_{GUESS}, i.e., all messages sent to C_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 AB in the following model where the parties do not agree on a common timeout value.

Definition 19 (Asymmetric Synchrony Assumption (ASA) Model). *The ASA model considers n parties $\mathsf{C}_1, \dots, \mathsf{C}_n$. Each C_i gets its own Δ^i_{GUESS} as private input, i.e., the other parties are not given Δ^i_{GUESS}. The adversary schedules messages, but it is guaranteed that all messages sent at time τ from an honest party to an honest party C_i are delivered no later than at time $\tau + \Delta^i_{\text{GUESS}}$. Note the only messages to C_i are delivered in time Δ^i_{GUESS}. Messages to other honest parties may be slower. Round complexity is measured in units of $\Delta^{\text{MAX}}_{\text{GUESS}} := \max_{\text{honest } \mathsf{C}_i} \Delta^i_{\text{GUESS}}$ and Δ_{NET}, where Δ_{NET} is the longest it took to send a message from an honest party to an honest party. In the weakly asymmetric synchrony assumption (WASA) model we make the stronger assumption that all messages between honest parties are delivered faster than any honest Δ^i_{GUESS}, i.e., $\Delta_{\text{NET}} \leq \Delta^{\text{MIN}}_{\text{GUESS}} := \min_{\text{honest } \mathsf{C}_i} \Delta^i_{\text{GUESS}}$.*

Definition 20. *Let Π^{OPT}_{RBD} be the protocol Π_{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 Π^{NA}_{RBD} be the protocol Π_{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 Network Agnostic RB and AB

We now analyse $\Pi^{\text{NA}}_{\text{RBD}}$ in the ASA model.

Theorem 9 (Network Agnostic RB). *Π^{NA}_{RBD} 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 parties 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 parties 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 parties trigger **Asynchronous Echo** and then within Δ_{NET} all parties 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 2 (Synchronous Echo Agreement). *If two honest C_i and C_j send (m_i, σ_i) and (m_j, σ_j) in **Synchronous Echo** then $m_i = m_j$.*

Proof. Consider C_i and C_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 C_i sent $(m_i, \sigma_s, \sigma_i')$ first. Then C_j started $\mathsf{Timeout}(\text{CollectFromHonest}, \Delta^j_{\text{Wait}})$ for $\Delta^j_{\text{Wait}} = \Delta^j_{\text{Guess}}$ after $(m_i, \sigma_s, \sigma_i')$ was sent. Assume that the network is synchronous. Since we are in the ASA model C_j thus saw $(m_i, \sigma_s, \sigma_i')$ before 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 C_i sent (m_i, σ_i) then it saw $n - t_{\text{S}}$ values $(\mathsf{C}_k, m_i, \cdot) \in \mathsf{SignedAsync}$. If C_j sent (m_j, σ_j) then it saw $n - t_{\text{S}}$ values $(\mathsf{C}_k, m_j, \cdot) \in \mathsf{SignedAsync}$. This means they saw values from $n - 2t_{\text{S}} > t_{\text{A}}$ common parties. So they saw $(\mathsf{C}_k, m_i, \cdot)$ and $(\mathsf{C}_k, m_j, \cdot)$ from at least one joint honest C_k. Therefore $m_i = m_j$. □

Lemma 3 (Network Agnostic Agreement). *If C_i and C_j are honest and output m_i and m_j then $m_i = m_j$.*

Proof. If C_i outputs m_i then it saw a valid (m_i, Σ_i) and if C_j outputs m_j then it saw a valid (m_j, Σ_j). If any of the parties saw a synchronous-valid value, then rename the parties such that C_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 $n - t_{\text{S}}$ parties in Σ_i there is at least one honest party as $n - t_{\text{S}} > t$, where t is the actual number of corruptions. Similarly, among the $n - t_{\text{S}}$ parties in Σ_j there is at least one honest party. Agreement then follows from Lemma 2. Assume then that both (m_i, Σ_i) and (m_j, Σ_j) are asynchronous-valid. Then among the $n - t_{\text{A}}$ parties in Σ_i and the $n - t_{\text{A}}$ parties 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 parties. Therefore there is at least one common honest party. Honest parties 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 $n - t_{\text{S}}$ parties in Σ_i and the $n - t_{\text{A}}$ parties in Σ_j there are at least $n - t_{\text{S}} - t_{\text{A}} > t_{\text{S}}$ common parties. Since $t_{\text{S}} \geq t$, where t is the actual number of corruptions, it follows that there is at least one honest party in common among Σ_i and Σ_j. Clearly, if an honest party signs both (Sync, m) and (Async, m') then $m' = m$. Therefore $m_i = m_j$. □

It follows that Π_{AAB} using $\Pi^{\text{NA}}_{\text{RBD}}$ as RB implements a network agnostic and optimistically responsive AB.

Corollary 3. *Π_{AAB} with $\mathcal{C} = \mathcal{P}$ using Π^{NA}_{RBD} as RB is an Atomic Broadcast (as defined in Definition 10) 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. The duration of each epoch is expected $\mathcal{O}(\Delta_{\text{Net}} + \Delta^{\text{Max}}_{\text{Guess}})$. Furthermore, if $t \leq t_A$ and the sender is honest then the duration of each epoch is expected $\mathcal{O}(\Delta_{\text{Net}})$.*

5.3 RB and AB with Synchronous Security and Optimistic Responsiveness

Theorem 10 (Optimistic). *Π^{OPT}_{RBD} is a justified RB protocol for the WASA model with at most t_S corruptions. All parties 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 parties terminate within time $2\Delta_{\text{Net}}$.*

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 2 still holds. Namely, we now wait $2\Delta^i_{\text{Guess}}$ instead of Δ^i_{Guess}, we are in the WASA model which gives stronger guarantees, and the network is alway synchronous, so the preconditions used for proving Lemma 2 are all stronger.

Lemma 4 (Optimistic Agreement). *If C_i and C_j are honest and output m_i and m_j then $m_i = m_j$.*

Proof. As in the proof of Lemma 3 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 $\Delta^i_{\text{Wait}} \geq 2\Delta^i_{\text{Guess}} \geq 2\Delta_{\text{Net}}$, as we are in the WASA model. Since $n - t_S > t$, clearly, at least one honest C_k signed (Sync, m_i) which in turn implies that it earlier signed (Async, m_i), say at time τ_i. Furthermore, at least one honest C_ℓ signed (Async, m_j), say at time τ_j. Assume for the sake of contradiction that $m_i \neq m_j$. If $\tau_i \leq \tau_j - \Delta_{\text{Net}}$ then (Async, m_i) would by definition of Δ_{Net} have reached C_ℓ before τ_j and then C_ℓ would by construction not have signed (Async, m_j). So we can assume that $\tau_j < \tau_i + \Delta_{\text{Net}}$. This means that the (Async, m_j) signed by C_ℓ reached C_k by $\tau_j + \Delta_{\text{Net}} < \tau_i + 2\Delta_{\text{Net}}$. When C_k signed (Async, m_j) by time τ_i then it did not sign (Sync, m_i) until time $\tau_i + 2\Delta_{\text{Net}}$. But by $\tau_i + 2\Delta_{\text{Net}}$ it received (Async, m_j). And then by construction of **Synchronous Echo** and $m_i \neq m_j$ it will not sign (Sync, m_i), a contradiction. □

It follows that Π_{AAB} using $\Pi^{\text{OPT}}_{\text{RBD}}$ as RB implements a secure and optimistically responsive AB in the WASA (and thus the synchronous) model.

Corollary 4. *Π_{AAB} with $\mathcal{C} = \mathcal{P}$ using Π^{OPT}_{RBD} as RB is an Atomic Broadcast (as defined in Definition 10) for the WASA model with at most t_S corruptions. The duration of each epoch is expected $\mathcal{O}(\Delta_{\text{Net}} + \Delta^{\text{Max}}_{\text{Guess}})$. Furthermore, if $t \leq t_A$ and the sender is honest then the duration of each epoch is expected $\mathcal{O}(\Delta_{\text{Net}})$.*

6 Corollaries

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

6.1 Asynchronous Covert AB with $t < n/2$

We show how to get atomic broadcast for the model with covert security [AL07]. We assume that every P_i and C_i can be Byzantine but does not do anything which will lead to an eventual common detection, where all honest C_j output a proof that C_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 C_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 C_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 $(\mathsf{C}_i \text{ IS CORRUPT: } (\mathsf{mid}, h', \sigma'_i), (\mathsf{mid}, h, \sigma_i))$. The message m can be relayed with linear complexity using an erasure code EC as in Π_{SQRB}.

Theorem 11 (informal). *Assume that $t < n$ and the servers are Byzantine but covert. Then* CRSS *is a secure broadcast protocol. If messages have length $\beta \geq n\log(\kappa)$, then the complexity is $\mathcal{O}(\beta n + n^2(\log(n)+\kappa))$, which is of the form $\mathcal{O}(\beta\,\text{RB} + \text{RB}_{\#})$ for $\text{RB} = n$ and $\text{RB}_{\#} = n^2(\log(n)+\kappa)$.*

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

6.2 Mixed Adversary AAB

We can also get AAB 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 12. *Assume that $2t_{\text{Crash}} + 3t_{\text{Byz}} < n$. 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. ☐

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

Asynchronous Echo C_i: On receiving (m, σ_s) from S, where $J_{\text{IN}}(m) = \top$ and $\mathsf{Ver}_{\mathsf{vk}_s}(m, \sigma_s) = \top$ and where there are no $(\mathsf{C}_j, m_j, \sigma_j) \in \mathsf{SignedAsync}$ 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 $(\mathsf{C}_i, m, \sigma_i)$ to $\mathsf{SignedAsync}$.

Collect Asynchronous Echos All servers: On receiving $(m_j, \sigma_s, \sigma_j)$ from C_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 C_j before, add $(\mathsf{C}_j, m_j, \sigma_j)$ to $\mathsf{SignedAsync}$.

Output All servers: If there exists m such that there are $n-t$ values $(\mathsf{C}_j, m, \sigma_j) \in \mathsf{SignedAsync}$ then let $\Sigma = \{(\mathsf{C}_j, \sigma_j)\}_{(\mathsf{C}_j, m, \sigma_j) \in \mathsf{SignedAsync}}$, 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 $(\mathsf{C}_j, m, \sigma_j)$ for distinct C_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 with designated sender S. For conciseness we do not explicitly mention the messages identifier mid and we let $J_{\text{IN}} = J^{\mathsf{mid}}$.

6.3 Sub-Quadratic Asynchronous MPC with $t < n/3$

We briefly give a subquadratic asynchronous MPC protocol for the setting described in Section 4 where we have ground members $\mathcal{P} = \{G_1, \ldots, \mathsf{P}n\}$ with $t < n/3$ Byzantine corruptions and have sampled a committee $\mathcal{C} = \{\mathsf{C}_1, \ldots, \mathsf{C}_{n_\mathcal{C}}\}$ with $t_\mathcal{C} < n_\mathcal{C}/2$ corruptions to implement a subquadratic AAB. As shown in Lemma 6.1 in [Coh16], given threshold fully homomorphic encryption and atomic broadcast and $t_\mathcal{C} < n_\mathcal{C}/2$ corruptions one can implement asynchronous multiparty computation for $t_\mathcal{C} < n_\mathcal{C}/2$. Since we implement atomic broadcast among the committee for $t_\mathcal{C} < n_\mathcal{C}/2$ from RB we can directly run [Coh16] in our framework and get AMPC from RB and $t_\mathcal{C} < n_\mathcal{C}/2$. This gives MPC for the network agnostic model and the optimistic model. We can also run the protocol in the sub-sampling setting with n ground members and $n_{\mathcal{C}}$ committee members, with corruption threshold $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ and $t < n/3$. To avoid having specialised setup among the committee $\mathcal{C}$ 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 $\mathcal{P}$ with reconstruction threshold $t+1$ and let them provide decryption as a service for the committee. The committee outcasts the encryption c of the output and the ground members send a decryption share to each committee member. If we have a large ground population $n \geq {n_{\mathcal{C}}}^2$ and they all give one input we can enforce that they give inputs in the same rounds, and then our atomic broadcast has communication complexity $\mathcal{O}(\beta n_{\mathcal{C}})$, 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 $n n_{\mathcal{C}}|y|$. This gives sub-quadratic AMPC for $t < n/3$ corruptions.

Theorem 13 (informal). *Assume $t_{\mathcal{C}} < n_{\mathcal{C}}/2$ corruption in $\mathcal{C}$ and $t < n/3$ corruptions in $\mathcal{P}$, assume $n \geq {n_{\mathcal{C}}}^2$, 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 $y = f(x_1, \ldots, x_n)$. Then there is an AMPC protocol for $\mathcal{P}$ with communication complexity $\mathcal{O}(n_{\mathcal{C}}\beta + n_{\mathcal{C}} n \gamma)$.*

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

References

ACKN23. Orestis Alpos, Christian Cachin, Simon Holmgaard Kamp, and Jesper Buus Nielsen. Practical large-scale proof-of-stake asynchronous total-order broadcast. In Joseph Bonneau and S. Matthew Weinberg, editors, *5th Conference on Advances in Financial Technologies, AFT 2023, October 23-25, 2023, Princeton, NJ, USA*, volume 282 of *LIPIcs*, pages 31:1–31:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.

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.

AW04. Hagit Attiya and Jennifer L. Welch. *Distributed computing - fundamentals, simulations, and advanced topics (2. ed.)*. Wiley series on parallel and distributed computing. Wiley, 2004.

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`.

Bd94. Josh Cohen Benaloh and Michael de Mare. One-way accumulators: A decentralized alternative to digital sinatures (extended abstract). In Tor Helleseth, editor, *EUROCRYPT'93*, volume 765 of *LNCS*, pages 274–285, Lofthus, Norway, May 23–27, 1994. Springer, Heidelberg, Germany.

BKL19. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150, Nuremberg, Germany, December 1–5, 2019. Springer, Heidelberg, Germany.

BKL21. Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572, Singapore, December 6–10, 2021. Springer, Heidelberg, Germany.

BKLZL20. Erica Blum, Jonathan Katz, Chen-Da Liu-Zhang, and Julian Loss. Asynchronous byzantine agreement with subquadratic communication. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part I*, volume 12550 of *LNCS*, pages 353–380, Durham, NC, USA, November 16–19, 2020. Springer, Heidelberg, Germany.

BLZLN22. Amey Bhangale, Chen-Da Liu-Zhang, Julian Loss, and Kartik Nayak. Efficient adaptively-secure byzantine agreement for long messages. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 504–525, Taipei, Taiwan, December 5–9, 2022. Springer, Heidelberg, Germany.

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.

CHLZ21. Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. On communication-efficient asynchronous MPC with adaptive security. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part II*, volume 13043 of *LNCS*, pages 35–65, Raleigh, NC, USA, November 8–11, 2021. Springer, Heidelberg, Germany.

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.

DG16. Danny Dolev and Eli Gafni. Some garbage in - some garbage out: Asynchronous t-byzantine as asynchronous benign t-resilient system with fixed t-trojan-horse inputs. *CoRR*, abs/1607.01210, 2016.

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`.

GLL+22. Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo-NG: Fast asynchronous BFT consensus with throughput-oblivious latency. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 1187–1201, Los Angeles, CA, USA, November 7–11, 2022. ACM Press.

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.

PS18. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Heidelberg, Germany.

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
```

```python
    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  <1/5 of GP corrupted (Algorand setting):")
    for secpar in security_parameters:
        smallest_safe_committee(secpar, 1/5)
```