Repeated Agreement is Cheap!

On Weak Accountability and Multishot Byzantine Agreement

PIERRE CIVIT, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
MUHAMMAD AYAZ DZULFIKAR, NUS Singapore, Singapore
SETH GILBERT, NUS Singapore, Singapore
RACHID GUERRAOUI, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
JOVAN KOMATOVIC, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland
MANUEL VIDIGUEIRA, École Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Byzantine Agreement (BA) allows n processes to propose input values to reach consensus on a common, valid L_o-bit value, even in the presence of up to $t < n$ faulty processes that can deviate arbitrarily from the protocol. Although strategies like randomization, adaptiveness, and batching have been extensively explored to mitigate the inherent limitations of one-shot agreement tasks, there has been limited progress on achieving good amortized performance for multi-shot agreement, despite its obvious relevance to long-lived functionalities such as state machine replication.

Observing that a weak form of accountability suffices to identify and exclude malicious processes, we propose new efficient and deterministic multi-shot agreement protocols for multi-value validated Byzantine agreement (MVBA) with a strong unanimity validity property (SMVBA) and interactive consistency (IC). Specifically, let κ represent the size of the cryptographic objects needed to solve Byzantine agreement when $n < 3t$. We achieve both IC and SMVBA with $O(1)$ amortized latency, with a bounded number of slower instances. The SMVBA protocol has $O(nL_o + n\kappa)$ amortized communication and the IC has $O(nL_o + n^2\kappa)$ amortized communication. For input values larger than κ, our protocols are asymptotically optimal. These results mark a substantial improvement—up to a linear factor, depending on L_o—over prior results. To the best of our knowledge, the present paper is the first to achieve the long-term goal of implementing a state machine replication abstraction of a distributed service that is just as fast and efficient as its centralized version, but with greater robustness and availability.

1 Introduction

There is a long history in distributed computing of working to solve one central problem: *how do we implement a service over a collection of servers so that it acts identically to running on a single "immortal" central server?* In the ideal centralized setting, a trusted server could collect inputs from users, compute the required outputs, and return them. The goal is to provide the illusion of this setting in a distributed system that would be just as fast and efficient as the centralized version, but with greater robustness and availability, specifically, tolerating faulty and malicious servers.

This problem, often referred to as implementing a "replicated state machine," lies at the heart of many distributed services today, ranging from distributed databases and distributed lock servers to blockchains. The underlying problem of Byzantine Agreement is typically the key component of a replicated state machine implementation, and can be viewed as the "one-shot" version of the "replicated state machine" problem.

More formally, a (one-shot) Byzantine Agreement (BA) protocol allows n processes to propose an L_{in}-bit value and agree on an L_o-bit value, while tolerating up to t arbitrary failures. If a process

Authors' Contact Information: Pierre Civit, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, pierre.civit@epfl.ch; Muhammad Ayaz Dzulfikar, NUS Singapore, Singapore, Singapore, ayaz.dzulfikar@u.nus.edu; Seth Gilbert, NUS Singapore, Singapore, Singapore, seth.gilbert@comp.nus.edu.sg; Rachid Guerraoui, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, rachid.guerraoui@epfl.ch; Jovan Komatovic, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, jovan.komatovic@epfl.ch; Manuel Vidigueira, École Polytechnique Fédérale de Lausanne (EPFL), Lausanne, Switzerland, manuel.ribeirovidigueira@epfl.ch.

exhibits an arbitrary failure, the process is said to be faulty; otherwise, the process is said to be correct or honest. The following guarantees must be satisfied:

- *Termination:* All correct processes eventually decide.
- *Agreement:* Correct processes decide on the same value.

To avoid trivial agreement on a pre-established value, BA necessitates an additional *Validity* property, delineating permissible decisions.

- *Interactive Consistency* (IC) requires that the output corresponds to a vector with the proposals of all correct processes.
- In *Strong Unanimity* (SBA), if all correct processes propose the same value, that value must be decided.
- Validated BA (VBA) ensures *External Validity*, stipulating that a decided value satisfies a globally verifiable predicate. When the implementation is optimized for long values, the problem is often referred to as multi-valued VBA (MVBA). An MVBA which is also an SBA is often called SMVBA.

Unfortunately, Byzantine Agreement—and the consensus problem in general—is inherently expensive. In a system of n servers, of which t may fail, every deterministic Byzantine Agreement protocol requires at least $t + 1$ rounds [47] and at least $\Omega(n^2)$ messages [46]. Any replicated state machine implementation would seem doomed to be much slower and more bandwidth-intensive than a centralized service, and so the goal of efficiently mimicking a single machine has seemed impossible to achieve.

In response to this depressing state of affairs, there has been a significant amount of work attempting to find workarounds: some systems focus on ensuring good performance *in the good case* (when there is not too much bad behavior) [5, 6, 45, 73], some by limiting resilience, some by weakening the attackers' powers. Others relax the basic requirements, focusing on ensuring eventual agreement, or on probabilistic correctness guarantees. None of these approaches, however, achieve the goal of "state machine replication": implementing in a distributed fashion the abstraction of a service that would be just as fast and efficient as its centralized version, but with greater robustness and availability.

Multi-shot agreement. In this paper, we show that this long-term goal **can be achieved**: in the case of repeated, or multi-shot agreement, we can obtain "best-case performance" in the worst case.

A multi-shot Byzantine Agreement corresponds to a sequence of $poly(\kappa)$ instances of Byzantine Agreement, where each instance is labeled by a sequential number. Importantly, we allow causal relationship between instances, meaning that a process proposes its input for the $(x+1)$-th instance after having decided the outputs of the x first instances.[1]

Specifically, we give a multi-shot SMVBA (resp., IC) protocol where all but $O(n^3)$ (resp., $O(n^2)$) instances terminate in $O(1)$ time, and, after $O(n^3)$ (resp., $O(n^2)$) instances, the average communication cost per instance is $O\big(n(L_o + \kappa)\big)$, where L_o is the size of the decision and κ is a security parameter. Since it would certainly take at least $\Omega(1)$ time and $\Omega(nL_o)$ communication to even broadcast the value to all the participants, essentially the agreement component of the protocol becomes (asymptotically) free!

We are not, of course, the first to explore the idea of amortizing (or averaging) costs over repeated instances of agreement. In a farsighted paper, in 1991, Bar-Noy, Deng, Garay, and Kameda [12] showed how to achieve an average of $O(1)$ time—but still suffered $O(nt)$ average communication per instance. More recently, an exciting paper by Momose, Ren, Shi, Wan, and Xiang [98] went

[1]We assume that as soon as a process has obtained an output, it is ready to propose a value for the next instance.

in the opposite direction, giving an implementation of Byzantine broadcast that has average communication cost $O(n(L_o + \kappa))$—but takes $O(t)$ time to complete.

Others have looked at a different perspective on "average cost," batching many decisions together: by (effectively) executing many agreement instances in parallel, the average costs can be reduced [9, 41, 57, 93]. The disadvantage, here, lies in the time necessary to wait for a sufficiently big batch to arrive, and the problem that inputs (representing operations for the replicated service) may have causal dependencies—and hence need to be executed sequentially. It is also important to note that state machine replication implementations based on practical agreement systems, e.g., [27, 75, 100] could average time and messages over $\Theta(n)$ rounds/views (of a linear-view-change protocol). The resulting protocol would amortize the cost, but with a weaker validity property and would not guarantee that all but a bounded number of instances are fast.

To the best of our knowledge, the present paper is the first to achieve the long-term goal of ensuring average $O(1)$ cost, a bounded number of slow instances, and average $O(nL_o)$ communication, while ensuring strong consistency (and validity) properties.

Key idea: Player-elimination. The intuition underlying this paper is deceptively simple: if we can detect and kick out malicious processes, then the protocol will get faster and more efficient as it runs. More specifically, if an agreement instance is either slow or has high communication, then ideally we would identify at least one provably malicious process and exclude it from future instances.

This idea of taking advantage of faulty process detection goes back at least as far as the paper by Bar-Noy, Dolev, Dwork and Strong [13]. In fact, the idea of a "trust graph", used in [98], is based on a similar notion of detection. Other recent papers on Byzantine Agreement have also tried to leverage the idea of detecting and kicking out malicious users [31, 49, 66–68, 76].

A first natural approach would require a strong notion of accountability. Typically, we wanted to guarantee two properties: whenever the cost is high, some malicious process is detected, and no honest process is ever suspected of being malicious. The latter seems important to ensure that we preserve the honest majority. Unfortunately, such a strong notion of accountability seems inherently expensive. You may have A accuse B and B accuse A, and it is expensive to determine which is honest. (And at the very least, it would require that everyone agree on who is malicious.)

A key insight has been formulated in the *player-elimination framework [64]*, where malicious processes aren't the only ones that can be excluded! As long as we ensure that at least half the excluded participants are malicious, we can ensure that an honest majority remains. To this end, when we have conflicting accusations (i.e., A accuses B and B accuses A), we can implement **the unfair parent rule**:

*When a parent finds two kids squabbling, they send them **both** to their room!*

In some sense, it does not matter "who started it" or which accusation is correct. Both will be excluded from the system. As long as an honest process always makes good accusations, we can be sure that at least one of the excluded processes is malicious and an honest majority is maintained.[2] Although this approach has already been used in secure multi-party computation (MPC) [11, 17, 40, 59, 63, 65, 80, 86], or (one-shot) Byzantine Broadcast over long value [31, 66], this is, to the best of our knowledge, the first time it has appeared in a multi-shot agreement task. Beyond this fundamental idea, there are several technical complications in its implementation, e.g.,

[2]Our protocols still consider the inputs of correct processes excluded from the "deciding committee" in the definition of validity. While this is "for free" for SMVBA, it requires extra consideration for our IC solution. Let us note that this extra work crucially relies on signatures, leaving open the question of signature-free IC with quadratic amortized communication and amortized constant latency.

how to deal with accusations from processes that have already been excluded, how to prioritize accusations, etc.

The second technical challenge lies in parallelizing the Byzantine Agreement protocols so that different pieces can be executed at the same time, and so that we can make optimistic progress when there are no malicious processes while rolling back properly when malice is detected. To this end, we deconstruct our Byzantine agreement protocol, carefully multiplexing views over epochs while simultaneously executing the weak accountability protocol in parallel.

Our results. Overall, then, the main goal of the paper is to give a new algorithm for repeated Byzantine Agreement with optimal amortized performance. We consider both traditional Byzantine Agreement with strong unanimity and external validity (SMVBA), as well as the more general problem of Interactive Consistency (IC):

- **Strong & External Validity**: We give an SMVBA protocol that has amortized $O(1)$ latency, bounded number of slow instances, and amortized $O(n(L_o + \kappa))$ communication. To our knowledge, no previous agreement protocol has achieved all three together.
- **Interactive Consistency**: We give an IC protocol that has amortized $O(1)$ latency, bounded number of slow instances, and amortized $O(nL_o + n^2\kappa)$ communication. As far as we know, no prior IC protocol has achieved all three together.

A summary of our work, compared to existing results, is presented at Figure 1.

Protocol	Problem	Resiliency	Amortized Communication	Amortized Latency	Crypto
[12] + [28]	S	$n/3$	$O(nL + n^2 \log n)$	$O(1)$	None
[34]	S+E	$n/2$	$O(nL + n^2\kappa)$	$O(t)$	TSS
[98]	BB	$n/(2+\epsilon)$	$O(n(L+\kappa))$	$O(t)$	TSS
This - 1	S+E	$n/2$	$O(n(L+\kappa))$	$O(1)$	TSS
[12] + [28]	IC	$n/3$	$O(n^2L + n^3 \log n)$	$O(1)$	None
[34]	IC	$n/2$	$O(n^2(L+\kappa))$	$O(t)$	TSS
This - 2	IC	$n/2$	$O(n^2(L+\kappa))$	$O(1)$	TSS
LB	S	-	$\Omega(nL)$	$\Omega(1)$	Any
LB	IC	-	$\Omega(n^2L)$	$\Omega(1)$	Any

Fig. 1. Performance of various protocols with L-bit proposals and κ-bit size of cryptographic objects. $\epsilon \in \Theta(1)$ denotes a small constant. S, E, BB, and IC refer to strong unanimity, external validity, byzantine broadcast, and interactive consistency, respectively. Byzantine broadcast ensures if the predetermined sender is correct, only the sender's value can be decided.

2 Related Work

The cost of one-shot agreement. In this paper, we address the inherent cost of Byzantine agreement by amortizing over many instances. We now describe other approaches: randomization and batching.

One (alternate) approach for circumventing this lower bound has been through randomization, including subquadratic protocols for static adversaries and resilience below $n/3$ [22–25, 51, 56, 70], as well as adaptive adversaries [71, 72]. For $n/(2+\Omega(1))$ resilience and an adaptive adversary, see: [2, 3, 19, 29, 87]. For $n/2$ resilience, there are still no subquadratic solutions. And for Byzantine Broadcast, quadratic communication complexity remains unavoidable, even for randomized protocols, under a strongly adaptive adversary [2, 3].

Looking at randomized protocols from an amortized perspective, several yield expected $O(1)$ round complexity, with: [7] has $O(nL + n^2\kappa)$ average communication, [29, 88] has $O(n\kappa(L + \kappa))$

average communication, and [20] + [2] has $O(nL + n\kappa^3)$ average communication. This paper, by contrast, focuses on deterministic algorithms (an approach that has proven popular in practice).

Another (deterministic) approach is to focus on efficient performance for large output sizes L_o via "batching", both in a setup-free setting [10, 28, 35], or within the honest-majority regime [7, 34], or in the dishonest majority setting [31, 54, 66, 83]. Typically, these protocols exhibit a communication complexity of the form $O(nL_o + X)$, which becomes asymptotically optimal when $L_o > X/n$. To achieve optimality, these methods must wait until a large enough batch input is available (e.g., $L = \Omega(\kappa n)$), which may introduce significant latency. Additionally, this approach does not support sequential, causally ordered decisions, which are critical for certain applications (e.g., generic compilers that simulate honest-but-curious behavior in a model with Byzantine failures [16, 58]).

On amortization techniques. While numerous works leverage coding techniques [8, 15, 21, 62, 89] to optimize communication for long values, whether in synchronous [7, 10, 28, 34, 35] or non-synchronous networks [1, 26, 30, 33, 36, 38, 42–44, 48, 50, 53, 69, 79, 83, 91, 92, 97, 101] (see [36]'s related work section for more details), there is a notable scarcity in the literature concerning amortization analysis for multi-shot tasks.

To the best of our knowledge, only two works address this: (1) the error-free (information-theoretically secure) protocol of [12] from the early 1990s (which can be combined with recent techniques from [28] to support large values), and (2) the recent cryptographic solution by [98], focusing on Byzantine Broadcast.

While [12] achieves good amortized latency, it still has quadratic communication. In some sense, this is due to being signature-free (since the quadratic Dolev-Reischuk bound applies even to failure-free executions in deterministic, signature-free protocols). Also, this lack of signatures inherently limits its resiliency to $n/3$, due to the well-known FLM impossibility result [52, 85]. In addition, their approach is restricted to achieving strong unanimity validity; extending their approach to more robust validity properties, such as SMVBA or IC presents significant challenges.

The second solution [98] is undoubtedly the most relevant related work. It offers two algorithms for amortized Byzantine broadcast: one for the challenging dishonest majority setting (which we do not address here) with a complexity that reflects that difficulty, and another for the honest majority setting with $n/(2+\Omega(1))$ resiliency. For this latter setting, we employ distinct and novel techniques to achieve significantly improved results. First, our solutions ensure $n/2$ resiliency. Second, we achieve better amortized $O(1)$ latency, both for SMVBA and for IC.

Excluding faulty processes. Our approach leverages the strategy of excluding faulty processes that contribute to complexity overheads. Existing methods of detecting failures, however, fall short in optimizing (amortized) complexity for agreement tasks. First, note that Dolev-Reischuk-style bounds apply even to omission failures [2, 3, 37, 82, 95], which are harder to detect than commission failures [61] and cannot easily be accompanied by an externally-verifiable proof of culpability [74].[3] Nevertheless, there has been a line of research that uses some form of detecting and masking failures on top of Exponential Information Gathering [4, 18, 55, 81] to achieve impressive latency optimization in one-shot consensus that is information-theoretic secure, although at a significant communication cost.

A relevant work in this regard is the trust-graph-based [99] solution proposed in [98], which maintains a trust graph to track misbehaviour across different instances. Nodes update this graph based on observed accusations between nodes, removing a dishonest node when it has lost the trust

[3] Our results use threshold signatures to bypass the impossibility result of Dolev-Reischuk, which applies even to failure-free executions in signature-free protocols, or in protocols with "idealized" digital signatures where a cryptographic certificate of $O(n)$ signatures still must contain $O(\kappa n)$ bits.

of all others. This approach, however, suffers from two drawbacks. First, it is inefficient. Even with t accusations against a process p_i, we cannot conclude that p_i is faulty, as these accusations may come from faulty nodes. If $t + 1$ accusations are required, a faulty node could misbehave in front of t correct nodes, inflating communication without definitive proof of fault. Our approach avoids this by not strictly focusing on excluding only faulty nodes. Second, when the considered validity property is highly dependent on the inputs of all the processes, it is hard to see how this approach can be implemented at a linear cost. Observe that our goals of linear communication complexity and constant latency are naturally met by a leader-based approach, where the leader(s) drive(s) a (causally-related) sequence of gradecast-like views, ensuring agreement and validity at all times while guaranteeing termination only optimistically. However, if a correct, non-leader process fails to decide at some instance, it is not clear who is to blame. If the leader does not receive enough inputs to produce a provably valid candidate output, processes cannot, at linear cost, resolve what we call the *omission ambiguity problem*, i.e., distinguish between (i) a receive-omission failure from the leader and (ii) send-omission failure(s) from the corresponding replica(s). Since they are indistinguishable in this setting, such situations cannot be easily resolved through undeniable misbehaviour detection. We address these issues by adopting the *player-elimination* framework [64], and the *king-fail-detect* paradigm [65], as described below.

Player-Elimination & Dispute-Control. Player-elimination is a general technique originally used to efficiently evaluate multiplication gates in secure multi-party computation (MPC) protocols based on secret sharing [11, 17, 40, 59, 63, 65, 80, 86]. In this framework, computation is divided into *segments*, at the end of which a consistency check is performed. If the check fails, processes should be able to agree on a set of *disputes*, pairs of processes such that at least one is faulty. This is typically achieved through mutual accusations while maintaining the property of *accuracy*, ensuring that no correct process accuses another correct process. Any such pair of processes can be *eliminated*, preserving a sufficient fraction of correct processes within a committee of non-eliminated participants. The segment is then "retried" to ensure correctness. At most t such eliminations are needed before reaching a fully correct committee. In the honest-majority setting, player-elimination offers a linear improvement over the [98] trust-graph approach. Player-elimination differs from the *dispute-control framework* [14], a generalization that has been successfully applied in MPC [17, 32, 60, 94], where disputes do not lead to eliminations but are used to improve the efficiency of subsequent segments (e.g., by making disputing pairs ignore each other). Player-elimination and dispute-control have also been used in the context of one-shot agreement on a long output value of size L_o [31, 66, 84], but with significant, albeit L_o-independent, additive communication overhead. Here, the segments correspond to agreement on the blocks making up the final result. In the context of Byzantine Broadcast, [99] introduces the notion of *trust graphs*, where processes maintain local sets of disputes without reaching agreement on them. This technique allowed a constant expected latency despite the absence of an honest majority. For multi-shot Byzantine Broadcast with good amortized complexity, [98] proposes a variant of dispute-control in which signed accusations are systematically double-echoed, ensuring that processes maintain consistent views of the underlying dispute graph.

King-Fail-Detect (KFD). The player-elimination framework requires only accurate accusations to be retained, where a corresponding dispute necessarily includes at least one faulty process. However, guaranteeing only accurate accusations can be costly due to the omission ambiguity problem. To address this problem, [65] introduced King-Fail-Detect (KFD) sub-protocols, where an arbitrary (non-eliminated) process is designated as the leader for each segment. In a KFD sub-protocol, if a correct process fails to decide as quickly as it should in a failure-free execution, it can temporarily assume that this is the fault of the leader. If the leader is indeed faulty, the corresponding

accusation will be accurate. If not, the (honest) leader would have observed a sending-omission failure from another replica and would broadcast a corresponding accurate accusation, which should take priority over all other (potentially inaccurate) accusations. This approach allows us to avoid resolving the ambiguity at great communication cost.

Paper organization. We state the formal system model and preliminaries in §3. We present the high-level idea of our multi-shot protocol, RARECOMPLAIN, in §4. We then describe the building blocks of RARECOMPLAIN in the following sections. Finally, we formally present and prove the correctness of RARECOMPLAIN in §10, before concluding in §11.

3 Preliminaries

Processes. We consider a static system of $\Pi = \{p_1, p_2, \ldots, p_n\}$ with $n = 2t + 1$ processes. A process that deviates from the prescribed protocol is called *Byzantine* or *faulty*; otherwise, we say the process is *correct* or *honest*. We assume the faulty processes are *computationally bounded*, i.e. they cannot forge signatures from other processes. When committees are involved, we denote n' as the size of the committee and t' as the maximum number of faults in the committee. We assume the existence of a global clock among processes.

Communication Network. The processes are connected by a complete *synchronous* network. In each round, processes send messages, receive messages, and update their local state.

Value. We denote by $\mathsf{Value}_{\mathsf{In}}$ the set of input values, and $\mathsf{Value}_{\mathsf{O}}$ the set of output values. When the input and output value domains are the same, we write both of them as Value. Throughout the paper, we denote $\perp$ as a special value which is not a part of $\mathsf{Value}_{\mathsf{In}}$ nor $\mathsf{Value}_{\mathsf{O}}$. We consider a multi-valued setting where the input has L bits. Abusing the notation, we may denote L as the input size of a protocol.

Public-Key Infrastructure. We assume the use of public-key infrastructure. Each process has a pair of public key and private key. By default, each message is signed by its sender.

Threshold signatures. We use a (k, n)-threshold signature scheme [77, 90]. Throughout our results, we use either of the following parameters: $(t + 1, n)$, $(t' + 1, n')$, and (n', n'). Notably, the last two are used in protocols with committee.

In a (k, n)-threshold signature scheme, each process holds a distinct private key; there exists a single public key. Each process p_i can use its private key to make a partial signature for a message m via $\mathsf{ShareSign}_i^k(m)$. A partial signature *psig* for a message m signed by process p_i can be verified by $\mathsf{ShareVerify}_i^k(m, psig)$. Then, a set of partial signatures S for a message m signed by k distinct processes can be combined into a single (threshold) signature *tsig* for a message m via $\mathsf{Combine}^k(S)$. Furthermore, such threshold signature can be verified by invoking $\mathsf{CombinedVerify}^k(m, tsig)$. Where appropriate, we omit explicit invocations of $\mathsf{ShareVerify}^k(\cdot)$ and $\mathsf{CombinedVerify}^k(\cdot)$. Finally, we denote by P_Signature and T_Signature a partial signature and a (combined) threshold signature, respectively. Importantly, under a security parameter κ, each signature under this scheme consists of $O(\kappa)$ bits.[4]

Error-Correcting Codes. We use error-correcting codes, mainly as erasure codes. Concretely, we use Reed-Solomon codes [89]. We denote RSEnc and RSDec as the encoding and decoding algorithm, respectively. $\mathsf{RSEnc}(M, m, k)$ takes as an input a message M consisting of k symbols, treats it as polynomial of degree $k - 1$, and output its evaluation on m points. Importantly, a symbol

[4]We assume $\kappa > \log n$ to avoid an adversary with exponential computational power and collision between cryptographic signatures.

consists of $O(\frac{|M|}{k} + \log(m))$ bits. Meanwhile, $\mathsf{RSDec}(k, r, T)$ takes as an input a set of symbols T, where r of them can be incorrect, and outputs a polynomial of degree $k-1$. Importantly, RSDec will output the original message if $|T| \geq k + 2r$. As erasure code, we typically use $\mathsf{RSEnc}(M, n, t+1)$ and $\mathsf{RSEnc}(n-t, 0, T)$ to encode and decode, respectively.

Multi-Shot Consensus. We number consensus instances from 1, 2, and so on.[5] We assume that a process immediately obtains the proposal for the i-th instance only after deciding in all instances $i' < i$.[6] All processes start the multi-shot consensus at the same time.[7]

Communication Complexity & Latency. The communication complexity of X consensus instances is the number of bits honest processes send until all honest processes decide in all X consensus instances. The average, or amortized[8], communication cost is the cost of X instances divided by X. Next, we define the latency of a consensus instance as the number of rounds between the first time an honest process obtains the proposal for that instance until the last honest process decides in that instance. Finally, the latency of X consensus instances is the sum of latency from all those X consensus instances. The average, or amortized, latency is the latency divided by X.

4 Technical Overview

In this section, we give an overview of the design process along with the high-level idea of RARECOMPLAIN, our protocol for multi-shot SMVBA and IC.

Observation 1: Optimistic agreement with $O(1)$ latency and $O(nL_o)$ communication complexity. Intuitively, Byzantine agreement is easy when there is no faulty process. In fact, this can be done in constant latency and linear communication complexity. Combined with the idea of detecting faulty process, one can imagine the following. Suppose there is an efficient (with constant latency and linear communication) Byzantine agreement that always guarantee safety (validity and agreement), but only guarantees all honest processes decide when there is no faulty process. Furthermore, it guarantees if some process fails to decide, a misbehaving process can be detected. Then, we can do multi-shot Byzantine agreement by repeatedly doing the following: (1) run this Byzantine agreement, (2) check whether there is a detected fault, and (3) remove any detected faults (and reduce the number of faults we need to tolerate). If a faulty process got removed, to ensure all processes decide, we can simply retry the Byzantine agreement instance until no faults are detected, before moving on to the next instance. As the number of faulty process is bounded, the number of retries is also bounded and thus, we will get a very efficient Byzantine agreement. To simplify the design, we would also like all honest processes to have the same view of the system as well (i.e, at step (3), they agree on which process to remove.)

Challenge 1: Difficulty of failure detection. However, a strong failure detection as described above can be hard to achieve, especially when we also want low latency and communication cost. To

[5]While omitted in our exposition, our protocol needs to tag each message with the consensus instance number; see Appendix A.

[6]In practice, if this is not the case, we can propose a $\perp$ input and carefully defining the validity property. Also, while our definition requires a valid decision per instance, we stress that the choice of this definition is not crucial. Indeed, we can observe that the variant of deciding a sequence of values that includes some $\perp$ decisions implies solving the variant with no $\perp$ decisions, by removing the $\perp$ decisions (e.g., as in the famous PBFT protocol [27]). Of course, the validity must be defined accordingly.

[7]If only a constant start-time offset can be guaranteed, we can use standard weak synchronization techniques [96] and standard simulation techniques such as in [78]. This requires a quadratic communication cost each time weak synchronization is needed, e.g., at the end of every epoch.

[8]We are aware that this use of "amortized" may not be standard—but it appears to be the term used in the distributed algorithms literature.

prove that a process is faulty, there must be at least 1 honest process that accuses the process of misbehaviour, which translates to at least $t + 1$ processes.[9] In other words, a faulty process may misbehave in front of t honest processes, and still cannot be proven to be faulty. Furthermore, this typically adds an $\Omega(nt)$ communication cost, which exceeds our budget. However, rather than using this strong notion of accountability, we can instead use a relaxed one as seen in the player-elimination framework. First, an important property in accountable distributed computing is *accuracy*, which says an honest process cannot accuse an honest process of being faulty. Put it differently, any accusation must involve at least one faulty process. If our Byzantine agreement satisfies accuracy, rather than figuring out which process is faulty, we can remove *both* processes involved in the accusation instead. As an honest process might get removed but still need to be able to decide, our (assumed) optimistic Byzantine agreement must be a committee-based one; agreement is done amongst the committee, then gets disseminated to all processes. Thus, removing a process now means removing it from the committee instead of the system. Importantly, as at least one of the removed process must be faulty, the committee would still have an honest majority.

When liveness fails, there might be multiple accusations made and thus, honest processes need to agree on which one needs to be used to update the committee. In particular, checking and removing faults boil down to another Byzantine agreement. This step, which hereafter we call *complain phase*, can be done by using single-shot IC, e.g., using the one from [34] that has $O(n)$ latency and $O(n^2\kappa)$ communication complexity for an $O(\log n)$-sized proposal (accusation). (The reason why we use IC, which will be clear at the end of this section, is because we need to be able to prioritize some accusations above others.) In this phase, all processes input the accusation they made to the IC, where the decided vector would be used by each process to (deterministically) update their view of the committee.

Challenge 2: Ensuring $O(1)$ latency and $O(nL_o)$ communication complexity on the composition with complain phase. Our current problem is that the complain phase is too costly; As long as there is a faulty process in the committee, we would need to run the complain phase, even if all honest processes have decided. However, what if we run multiple consensus instances first before doing that complain phase? That is, instead of just doing one instance, what if we do an *epoch* of V consensus instances before doing the complain phase?[10] Then, we can "charge" the communication complexity of the complain phase to those consensus instances. It would be the same as adding $O(\frac{n^2\kappa}{V})$ bits of of communication to each instance of consensus. When we set V to be (say) $\Omega(n)$, it means the complain phase would come *for free*!

Hereafter, we denote *payload phase* for an epoch e to be a phase where we do consensus instance which belongs to that epoch (instance $(e-1)V + 1$ to eV). One problem still remains, which is latency. Even if all honest processes have decided in epoch 1, consensus instance $V + 1$ cannot immediately start due to the complain phase. By our definition, the latency of instance $V + 1$ will be at least $\Omega(n)$. Moreover, this is not the only instance where it occurs; every first consensus instance of an epoch suffers from this.

To address this, we run both phases in parallel. Specifically, payload phase for epoch e is run in parallel with complain phase for epoch $e - 1$. When complain phase for epoch e detects failure, we then retry epoch e. This parallelization, along with the high-level view of RARECOMPLAIN, is depicted in Figure 2.

Challenge 3: Ensuring correctness and efficiency of payload phase. We now aim to realize the optimistic and efficient Byzantine agreement that we assumed to exist so far. First, each consensus

[9]To be more precise, after k failures have been masked, $t - k + 1$ accusations are sufficient to mask another failure.
[10]This idea of epoch is similar to segment in player-elimination framework.

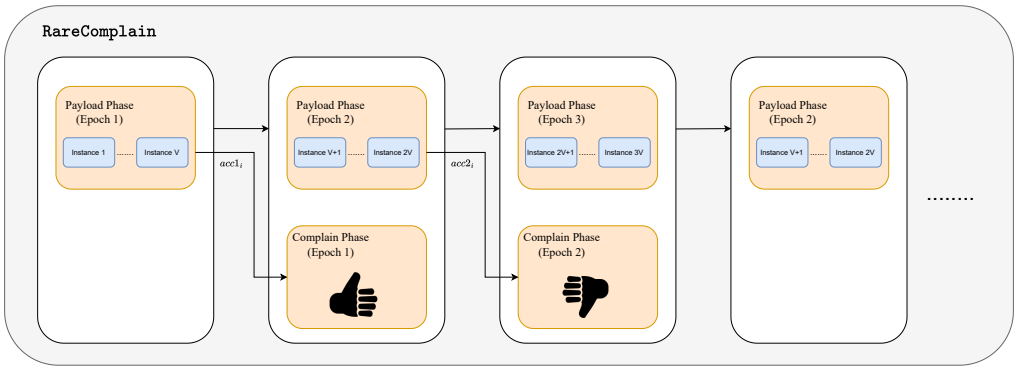

Fig. 2. RARECOMPLAIN. As complain phase for epoch 1 indicates there is no issue with epoch 1, processes can move on to the next epoch. Meanwhile, as complain phase for epoch 2 indicates there might be issues with epoch 2, processes have to retry epoch 2 (with an updated committee and trust graph). When retrying epoch 2, we do not do complain phase for epoch 3 because any accusation made in its payload phase might be inaccurate.

instance must satisfy validity, agreement, and accuracy. Furthermore, if an honest process fails to decide, there must be an honest committee that observes a misbehaving committee. Next, due to the structure of an epoch, we also need to take into account that honest processes which fail to decide in a consensus instance would not have the input for the next instances yet. While this may not be an issue for SMVBA (where committee inputs are enough to determine validity), this might be a problem for some validity property such as IC (which requires input from all, including non-commitee processes). Moreover, we have to ensure correctness while still being efficient, i.e. with $O(1)$ latency and $O(nL_o)$ communication complexity.

We put our agreement amongst the committee into an abstraction we call *alarm-gradecast*. In a nutshell, this module is a *leader-based* Byzantine agreement that always guarantee validity and agreement, and only guarantees decision when all committee are honest. Furthermore, when an honest process fails to decide, each honest process will be "alarmed" of a potentially faulty process. In our concrete implementation, we reduce the problem into certifying an admissible value and running validated Byzantine agreement on certified value, in a way similar to the DARE paradigm [34].

To take care of accusation, we apply the King-Fail-Detect (KFD) paradigm: when a committee member fails to decide, then (1) an honest process may accuse the leader, and (2) an honest leader will accuse a misbehaving committee it observed. Notice that an honest process may "wrongly" accuse an honest leader. However, in this case the honest leader will accuse a faulty committee. Hence, as long as in the complain phase, we always prioritize accusations made by the leader, any removed pair of processes will always include a faulty process. There are some additional details due to our usage of cryptography, which we discuss in Appendix A.

Notice that we have not taken into account when some honest processes do not have their proposal yet, which may raise the difficulty on determining whether a value is admissible or not. In a nutshell, we do this by having honest processes in the committee "blocks" agreement by not participating. This is fine as a faulty committee member will be caught, and we will retry the consensus instance. Let us note that while this is simple to do in SMVBA which only needs proposals from the committee, this becomes non-trivial in IC which also needs proposals from non-committee processes. We explain in more detail in §6.

Finally, we focus on how dissemination by the committee is done. First, there exists a committee dissemination with $O(nL_o + n^2\kappa)$ communication cost [35], which is sufficient for IC, but is too much for SMVBA. To solve this, we borrow the idea of trust graph from [98], which is also similar to the dispute set in dispute-control framework. In a nutshell, besides the committee set, all processes also maintain a trust graph, a directed graph where a directed edge from p_i to p_j means p_i still trusts p_j (p_i has not observed p_j of misbehaving). Dissemination is then done by assigning each process p_i with a *forwarder*, a committee process p_i still trusts. The forwarder of p_i has to send the decision, along with a proof of the decision (also provided by the alarm-gradecast). When an honest non-committee process fails to receive a decision, it means either (1) its forwarder is faulty, or (2) its forwarder also fails to decide and will accuse someone in the committee. Hence, complain phase also has to take care of trust graph maintenance as well. Importantly, as long as we prioritize complains made by the committee over the ones made by non-committee, the trust graph will always be accurate, i.e., an honest process never distrusts another honest process (see §9 for more details).

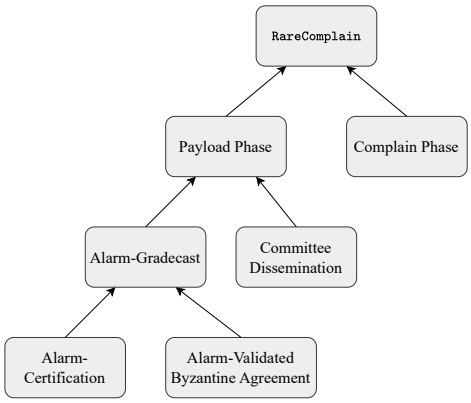

Fig. 3. An overview of how the subprotocols compose RARECOMPLAIN.

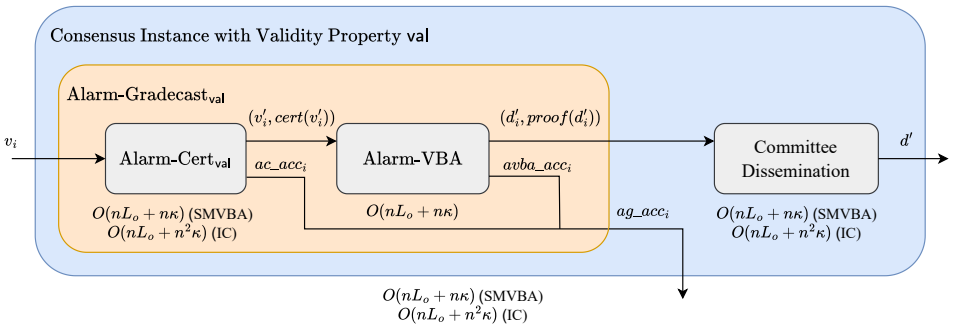

Fig. 4. Structure of the optimistic agreement for a consensus instance, with communications done in each subprotocol. Each subprotocol terminates in $O(1)$ rounds.

Overall, the composition of alarm-gradecast and committee dissemination allows an attempt of consensus instance with $O(1)$ latency and $O(nL_o)$ communication. How RARECOMPLAIN is composed can be seen in Figure 3, and the structure of a consensus instance within a payload phase can be seen in Figure 4. Our main results, which we formally prove in §10, are captured in the following theorems.

THEOREM 4.1. *There is an SMVBA protocol that tolerates $t < n/2$ faulty processes. Over $X \to \infty$ instances, the protocol exchanges $O\big((n^3+X)n(L+\kappa)\big)$ bits and ran over $O(X+n^3)$ rounds. Furthermore, except for $O(n^3)$ instances, each instance exchanges $O(n(L+\kappa))$ bits and terminates in $O(1)$ rounds.*

THEOREM 4.2. *There is an IC protocol that tolerates $t < n/2$ faulty processes. Over $X \to \infty$ instances, the protocol exchanges $O\big((n^2+X)n^2(L+\kappa)\big)$ bits and ran over $O(X+n^2)$ rounds. Furthermore, except for $O(n^2)$ instances, each instance exchanges $O(n^2(L+\kappa))$ bits and terminates in $O(1)$ rounds.*

The rest of the paper is devoted to explain the details and building blocks of RARECOMPLAIN.

5 Committee and Trust Graph

In our protocols, each honest process maintains a set of processes, the committee, and a trust graph, a directed graph among processes where an edge $p_i \to p_j$ indicates p_i still trusts p_j of being honest.[11] Importantly, we ensure each honest process has the same committee and trust graph, the committee has an honest majority, and removal of an edge $p_i \to p_j$ can only happen if p_i or p_j is faulty. As a shorthand, we may sometimes write an honest process in the committee as "honest committee" and a faulty process in the committee as "faulty committee". Similarly, we say "honest non-committee" (resp. "faulty non-committee") as an honest (resp. faulty) process that is not in the committee.

Throughout the rest of the paper, we will use the following definitions. Given a committee, the *leader* of this committee is the process with the smallest identifier in it. Then, given a process p_i, a committee, and a trust graph TG, the *forwarder* for p_i is the process with the smallest identifier in the committee that p_i trusts in TG. Based on the properties of the committee and the trust graph, the forwarder for any honest process p_i must be well-defined.

Both the leader and forwarder can be computed deterministically. Moreover, as long as all processes have the same committee and trust graph, for each process p_i, there will be at most one forwarder for p_i. Furthermore, for each honest committee p_i, p_i can compute on its own the set of processes whose forwarder is p_i. For completeness, we present the pseudocode to compute each definition at Algorithm 1.

Algorithm 1 Pseudocode to Compute Committee Leader and Forwarder

```
1:  ▷ Leader of the committee
2:  function Leader(Set(Process) Committee):
3:      return process with the smallest identifier in Committee
4:  ▷ Output the forwarder for p_i
5:  function Forwarder(Process p_i, Set(Process) Committee, TrustGraph TG):
6:      let Good_i ← {p_j | ⟨p_i → p_j⟩ ∈ TG} ∩ Committee
7:      return process with the smallest identifier in Good_i
8:  ▷ Output the set of processes which forwarder is p_i
9:  function ToForward(Process p_i, Set(Process) Committee, TrustGraph TG):
10:     return {p_j | Forwarder(p_j, Committee, TG) = p_i}
```

[11] Trust graph is similar to the dispute set in the dispute-control framework.

6 Alarm-Gradecast

Alarm-Gradecast is a module that handles agreement amongst the committee with validity property val. Let us note that all processes still participate in the module, as some validity property such as IC requires input from all processes and not just the committee; still, the agreement itself would only be done amongst the committee. At a high-level, this module is a leader-based Byzantine agreement protocol that follows the KFD paradigm when making accusations. Moreover, a decision must be supported by a *decision proof*, a threshold signature signed by all processes in the committee that proves the legitimacy of the decision (see §6.2 for a formal definition of decision proof). Finally, let us stress that an instance of the module may be invoked multiple times (i.e., to retry an instance where some honest process fails to decide).

The module exposes the following interfaces:

- *request* propose($Committee \subseteq \Pi, v \in \mathsf{Value}_{\mathsf{In}} \cup \{\bot\}$): A process participates in the agreement while proposing v. $v = \bot$ means the process does not have its actual proposal yet.
- *indication* decide($v' \in \mathsf{Value}_{\mathsf{O}} \cup \{\bot\}, proof \in \mathsf{DecisionProof} \cup \{\bot\}, acc \in \Pi \cup \{\bot\}$): if $v' \neq \bot$, a process decides on a value v', accompanied with its decision proof. Otherwise, it is alarmed of a possibly faulty process acc.

Next, the module assumes the following:

- (*Valid Configuration*) Each honest process has the same committee. Moreover, the majority of the committee is honest.
- (*Subset Committee*) Let c, c' be the two committees that an honest process inputs in two consecutive invocations. Then, $c' \subseteq c$.

Lastly, the module guarantees the following properties:

- (*Validity*) If an honest process decides v', then it also receives a decision proof for v' and v' is admissible according to val.
- (*Termination*) All honest processes eventually receive decide indication at the same time.
- (*Decide-or-Accuse*) Each honest committee either receives decide($v' \neq \bot, proof, \bot$) or decide($\bot, \bot, acc \neq \bot$).
- (*Alarm-Priority*) If an honest committee proposes $\bot$, then all honest processes will be alarmed.
- (*Sender-Validity*) If all honest committees propose non-$\bot$ values and p_ℓ, the leader of the committee, is honest, then:
  - all honest committees decide on a value, or
  - p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.
- (*Agreement*) Suppose a decision proof for v' exists. Then, if a decision proof for v^* exists, $v' = v^*$.

In our implementation, alarm-gradecast is done by (1) certifying an admissible value via alarm-certification (§6.1), followed by (2) agreement on a certified value via alarm-VBA (§6.2). In short, validity came from alarm-certification, while agreement came from alarm-VBA. If any of the steps fail, then an honest committee will be "alarmed" of a potentially faulty committee. Some of the key challenges include (1) ensuring validity holds even if some honest processes do not have their input yet, and (2) ensuring agreement holds even if the module might be invoked (retried) multiple times, potentially with different committees. In a nutshell, (1) is done by ensuring decision can only happen if all honest committees already have their input e.g., by requiring every committee to participate, and (2) is done by ensuring the committee is always a subset of prior committees. Let us note that while our solution for (1) may cause the gradecast to fail even with just one faulty committee, this is not a problem as such failure will be detected and in the future, the module will

succeed when everyone in the committee is honest. Furthermore, in KFD paradigm, an honest process may "wrongly" accuse an honest leader, although in this case the leader surely would accuse a faulty process. While this is fine in our complain phase (see §9), there are some additional details due to the cryptographic tools we use which we discuss in Appendix A.

Before stating our results here, let us note that we also need to assume the following in our alarm-gradecast for IC. At a high-level, it is to ensure validity holds even if some honest process outside the committee do not have their input yet.

Assumption 1. If there is an honest non-committee process without its input (it inputs $\perp$), then there is also an honest committee process without its input.

The following are the key results related to this module; we will prove them at the end of this section (see §6.3). At a high-level, this is done via a direct composition of (efficient) alarm-certification and alarm-VBA.

Lemma 6.1. *There is an alarm-gradecast for SMVBA with $O(1)$ latency and $O(n(L+\kappa))$ communication.*

Lemma 6.2. *Assuming Assumption 1, there is an alarm-gradecast for IC with $O(1)$ latency and $O(n^2(L+\kappa))$ communication.*

We now explain the two building blocks of the module.

6.1 Alarm-Certification

Alarm-Certification is a module which helps in certifying a value admissible according to the validity property val it is concerned with. In a nutshell, this module is similar to the dispersal module from the DARE paradigm [34], with the main differing features being (1) the certification is on the actual value, rather than a (short) digest of the value, and (2) the capability to alarm a process of a potentially faulty process when certification fails.

The module exposes the following interfaces:

- *request* certify($Committee \subseteq \Pi, v \in \mathsf{Value}_{\mathsf{In}} \cup \{\perp\}$): a process participates in the certification while inputting v.
- *indication* acquire($v' \in \mathsf{Value}_{\mathsf{O}} \cup \{\perp\}, cert \in \mathsf{Certificate} \cup \{\perp\}, acc \in \Pi \cup \{\perp\}$): if $v' \neq \perp$, a process acquires a value v' along with a certificate proving v' is admissible according to val. Otherwise, it is alarmed of a possibly faulty process acc.

Formally, under a fixed committee of size n', a certificate for a value v, σ, is a threshold signature signed by all processes in the committee such that $\mathsf{CombinedVerify}^{n'}(\langle \textsc{certify}, v\rangle, \sigma)$ returns *true*. Furthermore, the way the certificate is signed, v must be admissible according to val.

Next, the module assumes the following:

- (*Valid Configuration*) Each honest process has the same committee. Moreover, the majority of the committee is honest.

Finally, the module guarantees the following properties:

- (*Validity*) If an honest process acquires v', it also receives a certificate for v' and v' is admissible according to val.
- (*Termination*) All honest processes eventually receive acquire indication at the same time.
- (*Acquire-or-Accuse*) Each honest committee either receives acquire($v' \neq \perp, cert, \perp$) or acquire($\perp, \perp, acc \neq \perp$).
- (*Alarm-Priority*) If an honest committee inputs $\perp$, then all honest processes will be alarmed.

- (*Sender-Validity*) If all honest committees input non-$\perp$ values and p_ℓ, the leader of the committee, is honest, then:
  - all honest committees acquire a value, or
  - p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.

In particular, our implementations closely resemble a view of dispersal from [34], which follows a leader-based pattern. Our key challenge here is to ensure only admissible values can be certified, even when some honest processes may not have their input yet. Our main idea is by making an honest committee sign only if they have their input and requiring everyone in the committee to sign the certificate (in contrast to dispersal from [34] which only needs $t' + 1$ processes to sign). Finally, accusation is made following the KFD paradigm: a non-leader may accuse the leader, while the leader accuses a committee it observed of being silent.

6.1.1 Alarm-Certification for SMVBA. Our implementation is very similar to the implementation of dispersal for SMVBA from [34], with some differences due to the committee and different thresholds used in the signatures (which also helps simplify the protocol). Let us describe the high-level idea of the algorithm; the detailed algorithm can be read at Algorithm 2. First, the leader collects the input from (only) everyone in the committee. If there is a value that appears $t' + 1$ times, then the leader builds a positive certificate via threshold signature for this value. Otherwise, the leader partitions the value domain into $O(1)$ disjoint intervals such that from the collected inputs, for each interval, there are at most t' processes which sent a value inside the interval.[12] The leader then leads the creation of a negative certificate, a $t' + 1$ threshold signature for each interval, where a process signs a value it does not belong into. Intuitively, a negative certificate proves that for each interval in the input domain, there is an honest process whose input does not belong to that interval. Observe that a positive certificate proves that the corresponding value is admissible by SMVBA, while a negative certificate proves that any externally-valid value is admissible by SMVBA. Hence, using the positive or negative certificate, a leader can lead the creation of a certificate for an admissible value.

We now prove that the algorithm indeed implements the alarm-certification for SMVBA.

Lemma 6.3 (Termination). *All honest processes eventually receive* acquire *indication at the same time.*

Proof. Clear from the algorithm. □

Lemma 6.4 (Acquire-or-Accuse). *Each honest committee either receives* acquire$(v' \neq \perp, cert, \perp)$ *or* acquire$(\perp, \perp, acc \neq \perp)$.

Proof. See line 2, line 41, and line 44. Furthermore, at line 44, *accused* must be well-defined. □

Lemma 6.5 (Alarm-Priority). *If an honest committee inputs $\perp$, then all honest processes will be alarmed.*

Proof. For a non-committee, this is ensured by the rule at line 2. Next, a certificate must be signed by all committees. As an honest committee is silent when it inputs $\perp$, all honest processes receive acquire$(\perp, \perp, acc \neq \perp)$. □

Lemma 6.6 (Validity). *If an honest process acquires v', it also receives a certificate for v' and v' is admissible according to SMVBA.*

[12] This can be done greedily by making each interval as large as possible while still containing at most t' vales from the committee.

Algorithm 2 Alarm-Certification for SMVBA: Pseudocode (for process p_i)

Rules:
* if $p_i \notin Committee_i$ or $v_i = \perp$, p_i stays silent and in the end, triggers acquire$(\perp, \perp, \text{Leader}(Committee_i))$
Input parameters:
Set(Process) $Committee_i$
Value $\cup \{\perp\}$ v_i ▷ might be $\perp$
Local variables:
Value $\cup \{\perp\}$ $certified_value_i \leftarrow \perp$
PositiveCertificate $\cup \{\perp\}$ $pos_cert_i \leftarrow \perp$
Round 1:
send $\langle \text{DISCLOSE}, v_i, \text{ShareSign}_i^{t'+1}(\langle \text{DISCLOSE}, v_i \rangle) \rangle$ to Leader$(Committee_i)$
Round 2:
if $p_i = \text{Leader}(Committee_i)$ and received $\langle \text{DISCLOSE}, \cdot, \cdot \rangle$ from all processes in $Committee_i$:
if there is v' with $t'+1$ discloses for v': ▷ positive certificate can be made
$certified_value_i \leftarrow v'$
$pos_cert_i \leftarrow \text{Combine}^{t'+1}(psig \mid psig \text{ is received from a disclose for } v')$
else: ▷ build a negative certificate
let $domain_partition \leftarrow \text{partition}(discloses)$ ▷ partition into $O(1)$ ranges each with $\leq t'$ discloses
multicast $\langle \text{PARTITION}, domain_partition \rangle$ to $Committee_i$
Round 3:
if received $\langle \text{PARTITION}, domain_partition \rangle$ from Leader$(Committee_i)$ with $O(1)$ ranges:
let $negatives \leftarrow \emptyset$
for each range $[l, r] \in domain_partition$ with $\neg(l \leq v_i \leq r)$: ▷ partially signs ranges not containing v_i
$negatives \leftarrow negatives \cup \{l, r, \text{ShareSign}_i^{t'+1}(\langle \text{OUTSIDE}, l, r \rangle)\}$
send $\langle \text{PARTITION_REPLY}, negatives \rangle$ to Leader$(Committee_i)$
Round 4:
if $p_i = \text{Leader}(Committee_i)$ and $certified_value_i \neq \perp$: ▷ if positive certificate exists,
multicast $\langle \text{VALID_VALUE}, certified_value_i, pos_cert_i \rangle$ to $Committee_i$ ▷ multicast the corresponding value
else if $p_i = \text{Leader}(Committee_i)$ and received $\langle \text{PARTITION_REPLY}, \cdot \rangle$ from all in $Committee_i$:
if there are $t'+1$ signatures for each range $[l, r] \in domain_partition$:
let $neg_cert \leftarrow \text{build_negative_certificate}(domain_partition, partition_replies)$
multicast $\langle \text{VALID_VALUE}, v_i, neg_cert \rangle$ to $Committee_i$ ▷ multicast own value
Round 5:
if received valid $\langle \text{VALID_VALUE}, v, cert \rangle$ from Leader$(Committee_i)$ and $cert$ proves the validity of v:
send $\langle \text{CERTIFY}, v, \text{ShareSign}_i^{n'}(\langle \text{CERTIFY}, v \rangle) \rangle$ to Leader$(Committee_i)$
Round 6:
if $p_i = \text{Leader}(Committee_i)$ and received $\langle \text{CERTIFY}, v, psig \rangle$ from all in $Committee_i$:
let $cert \leftarrow \text{Combine}^{n'}(\{psig \mid psig \text{ is a partial signatures from received messages}\})$
multicast $\langle \text{CERTIFIED_VALUE}, v, cert \rangle$ to $Committee_i$ ▷ multicast value and certificate
Round 7:
if received valid $\langle \text{CERTIFIED_VALUE}, v, cert \rangle$ from Leader$(Committee_i)$:
trigger acquire$(v, cert, \perp)$
else:
let $accused$ be the first misbehaving committee p_i observed
trigger acquire$(\perp, \perp, accused)$

Proof. A certificate is partially signed by all processes in the committee, which includes at least one honest process. Meanwhile, an honest committee only partially signs a certificate for a value v' upon receiving a positive or negative certificate. A positive certificate for v' means at least one honest committee disclosed the value v'. Hence, it must be admissible under SMVBA. Meanwhile, a negative certificate proves that any value satisfying external validity must be admissible. Thus, it suffices to show that a negative certificate cannot be formed when all honest processes input the

same value. In this case, at least $n' - t' \geq t' + 1$ honest committees input the same value. So, the value domain cannot be partitioned such that each interval contains at most t' discloses. □

To prove the Sender-Validity property, we need to prove some helper lemmas first.

Lemma 6.7. *If all processes in the committee are honest and input non-$\perp$ values, then each honest committee will acquire a value and its certificate.*

Proof. All honest committees will disclose their values to the leader. Then, the leader p_ℓ either observes $t' + 1$ discloses for some value or observes that there is no such value. In the former, p_ℓ can build a positive certificate, while in the latter, p_ℓ can partition the value domain into $O(1)$ interval with each interval having $\leq t'$ discloses. In the latter case, each honest committee will partially sign each interval in the partition to which its input does not belong. Observe that in each interval, at least $n' - t' \geq t' + 1$ committees will sign for that interval, allowing p_ℓ to build a negative certificate. In any case, p_ℓ will be able to build a positive or a negative certificate, proving that a certain value v' is admissible according to SMVBA. Hence, p_ℓ will be able to convince each committee process to partially sign a certificate for v', allowing p_ℓ to build a certificate for v'. p_ℓ then multicast the value and its certificate to all committees, ensuring they will acquire v' and its certificate. □

Lemma 6.8. *If all honest committees input non-$\perp$ values, the leader, p_ℓ, is honest, and p_ℓ acquires a value and its certificate, then all honest committees will do so as well.*

Proof. Before receiving the indication, p_ℓ must have previously multicast v' and its certificate and so, all honest committees will acquire them as well. □

The next lemma follows from the observation of the KFD paradigm.

Lemma 6.9. *If all honest committees input non-$\perp$ values, the leader, p_ℓ, is honest, and p_ℓ receives* $\mathsf{acquire}(\perp, \perp, acc_\ell \neq \perp)$*, then, acc_ℓ is a faulty committee.*

We are now ready to prove Sender-Validity.

Lemma 6.10 (Sender-Validity). *If all honest committees input non-$\perp$ values and p_ℓ, the leader of the committee, is honest, then:*

- *all honest committees acquire a value, or*
- *p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.*

Proof. From Lemma 6.7, it is possible for p_ℓ to acquire a value v' and its certificate. Furthermore, as shown in Lemma 6.8, in this case, all honest committees will also acquire v' and its certificate. Otherwise, p_ℓ will be alarmed of a process acc_ℓ which, from Lemma 6.9, must be a faulty committee. □

Hence, the algorithm correctly implements the alarm-certification for SMVBA. Next, observe that in each round, honest processes collectively send $O(n(L + \kappa))$ bits (for L-bits inputs). Thus, we obtain the following.

Lemma 6.11. *There is an alarm-certification for SMVBA with $O(1)$ latency and $O(n(L + \kappa))$ communication.*

6.1.2 Alarm-Certification for IC. Unlike the alarm-certification for SMVBA, alarm-certification for IC is non-trivial because we also need the input from non-committee processes. However, an honest non-committee that does not have its input yet cannot prevent certification, as in the first place, its signature is not needed for the certificate. To tackle this, in our implementation, we assume if such process exist, then there is an honest committee without input as well (Assumption 1). In the

full picture, this assumption would later be ensured by how we disseminate the decision from the committee during the payload phase for IC (see §7.2 and §8).

We now briefly describe our implementation; the algorithm is presented at Algorithm 3. Initially, all processes disclose their input to everyone (if they have one). Then, each process in the committee build a vector where the i-th entry is either (a) a value disclosed by p_i (along with p_i's signature), or (b) an accusation against p_i (along with a partial signature for this accusation). Each committee member then sends this vector to the leader. From the received vectors, the leader attempts to build a vector admissible according to IC along with its proof. For the i-th entry, the leader either put a disclose from p_i where the proof is a signature by p_i, or a threshold signature of accusation from all (n') processes in the committee. If such vector and proof can be formed, the leader can then lead the creation of a certificate for the vector, using the proof to prove the validity of the vector. We say that the proof validates the vector if, for each process p_i:

- If p_i is in the committee, the i-th entry of the vector is a value and the proof contains a signature by p_i disclosing the value.
- If p_i is not in the committee, the i-th entry of the vector is a value, and the proof contains a signature by p_i disclosing the value, or the i-th entry of the vector is $\perp$ and the proof is a threshold signature accusing p_i signed by everyone in the committee (hereafter referred to as *PoM*, i.e., *proof-of-malice*).

Only non-committee processes can be accused, which is fine because we only need to ensure certification to succeed when all committees are honest. Observe that an accusation against silent honest committee that does not have its input yet might be made by an honest process, and, is possibly used by the adversary when an instance of this module is retried. We underline that this is not a problem; see Appendix A for a discussion on this.

We now prove the correctness of the algorithm.

Lemma 6.12 (Termination). *All honest processes eventually receive* acquire *indication at the same time.*

Proof. Straightforward from the algorithm. □

Lemma 6.13 (Acquire-or-Accuse). *Each honest committee either receives* acquire$(v' \neq \perp, cert, \perp)$ *or* acquire$(\perp, \perp, acc \neq \perp)$.

Proof. See line 3, line 38, and line 41. Furthermore, if a process receives the indication at line 41, it must have observed a misbehaving committee. □

Lemma 6.14 (Alarm-Priority). *If an honest committee inputs $\perp$, then all honest processes will be alarmed.*

Proof. For a non-committee process, this follows from Algorithm 3. Next, as an honest committee that inputs $\perp$ will not participate in the signing of a certificate, no certificate can be formed. □

We next prove Validity. Before that, we need to prove some helper lemmas first.

Lemma 6.15. *If an honest non-committee inputs $\perp$, then all honest processes receive* acquire$(\cdot, \cdot, acc \neq \perp)$.

Proof. From Assumption 1, there must be an honest committee that inputs $\perp$. The lemma is then implied from Lemma 6.14. □

Lemma 6.16. *If all honest processes input non-$\perp$ values, then no PoM can be formed against any honest process.*

Algorithm 3 Alarm-Certification for IC: Pseudocode (for process p_i)

Rules:
  * each process always execute round 1
  * then, if $p_i \notin Committee_i$ or $v_i = \perp$, p_i stays silent and in the end, triggers acquire$(\perp, \perp, \text{Leader}(Committee_i))$

Input parameters:
  Set(Process) $Committee_i$
  $\text{Value}_{\text{In}} \cup \{\perp\}$ v_i ▷ might be $\perp$

Round 1:
  if $v_i \neq \perp$:
    broadcast $\langle$DISCLOSE, $v_i\rangle$ ▷ broadcast a signed message disclosing the input

Round 2:
  let $vec_i \leftarrow \{\perp, \ldots, \perp\}$ be a vector of length n
  for each $1 \leq j \leq n$:
    if received signed $\langle$DISCLOSE, $v_j\rangle$ from p_j:
      $vec_i[j] \leftarrow \langle$DISCLOSE, $v_j\rangle$
    else if $p_j \notin Committee_i$:
      $vec_i[j] \leftarrow \langle$ACCUSE, p_j, $\text{ShareSign}_i^{n'}(\langle$ACCUSE, $p_j\rangle)\rangle$ ▷ accuse p_j of being faulty
  send $\langle$VECTOR, $vec_i\rangle$ to Leader$(Committee_i)$

Round 3:
  if $p_i = \text{Leader}(Committee_i)$ and received $\langle$VECTOR, $vec'\rangle$ from all in $Committee_i$:
    let $vec_i \leftarrow \{\perp, \ldots, \perp\}$ and $vec_proof_i \leftarrow \{\perp, \ldots, \perp\}$ both be a vector of size n
    for each Integer $1 \leq j \leq n$:
      if there is a disclose signed by p_j for some value v' from the received vectors:
        $vec_i[j] \leftarrow v'$
        $vec_proof_i \leftarrow \langle$DISCLOSE, $v'\rangle$ signed by p_j
      else if $p_j \notin Committee_i$ and all committee accuses p_j:
        $vec_proof_i[j] \leftarrow \text{Combine}^{n'}(\{psig \mid psig$ is from a received vector to accuse $p_j\})$
    if $vec_proof_i[j] \neq \perp$ for each $1 \leq j \leq n$:
      multicast $\langle$VERIFIABLE-VECTOR, vec_i, $vec_proof_i\rangle$ to $Committee_i$

Round 4:
  if received $\langle$VERIFIABLE-VECTOR, vec, $vec_proof\rangle$ from Leader$(Committee_i)$ and vec_proof validates vec :
    send $\langle$CERTIFY, vec, $\text{ShareSign}_i^{n'}(\langle$CERTIFY, $vec, \rangle)\rangle$ to Leader$(Committee_i)$

Round 5:
  if $p_i = \text{Leader}(Committee_i)$ and received $\langle$CERTIFY, vec, $psig\rangle$ from all in $Committee_i$:
    let $cert \leftarrow \text{Combine}^{n'}(\{psig \mid psig$ is partial signature from a received messages$\})$
    multicast $\langle$CERTIFIED_VALUE, vec, $cert\rangle$ to $Committee_i$

Round 6:
  if received valid $\langle$CERTIFIED_VALUE, vec, $cert\rangle$ from Leader$(Committee_i)$:
    trigger acquire$(vec, cert, \perp)$
  else:
    let *accused* be the first misbehaving committee p_i observed
    trigger acquire$(\perp, \perp, accused)$

Proof. A PoM must be partially signed by all processes in the committee, which contains some honest processes. Meanwhile, an honest committee only partially signs a PoM for a process p_j if p_j does not disclose its input. This is impossible when all honest processes input non-$\perp$ values. □

We are now ready to prove Validity.

Lemma 6.17 (Validity). *If an honest process acquires v', it also receives a certificate for v' and v' is admissible according to IC.*

Proof. From Lemma 6.14 and Lemma 6.15, if an honest process inputs $\perp$, then all honest processes will receive acquire($\perp, \perp, acc \neq \perp$). Hence, we only need to consider when all honest processes input non-$\perp$ value.

A certificate must be signed by every committee process, where at least one must be honest. Such process only signs a certificate of a vector if it is valid. Thus, it suffices to show that for each honest non-committee, no PoM can be formed against them. As this follows from Lemma 6.16, the vector v' must be admissible according to IC. □

To prove the Sender-Validity property, we prove some helper lemmas first.

Lemma 6.18. *If all processes in the committee are honest and input non-$\perp$ values, then each honest committee will acquire a value v' and its certificate.*

Proof. From Assumption 1, all honest non-committee processes must input non-$\perp$ values as well. Therefore, all honest processes disclose their value at round 1. Hence, at round 2, for each honest process p_j, each honest committee p_i must have received a disclosed value from p_j. Then, at round 3, the leader p_ℓ will receive messages from all committees. Furthermore, p_ℓ will be able to build a valid vector and its proof: each process in the committee has a value along with a disclose it signed, and each process that does not disclose anything will have a PoM signed by all (honest) committee processes. Thus, p_ℓ will be able to build a certificate for some vector v' and so, each honest committee will acquire v' and its certificate. □

Lemma 6.19. *If all honest committees input non-$\perp$ values, the leader of the committee, p_ℓ, is honest, and p_ℓ acquires a value v' and its certificate, then all honest committees will do as well.*

Proof. Previously p_ℓ must have multicast a vector v' along with its certificate to the committee and thus, all honest committees will acquire them as well. □

The following lemma is from the observation on KFD paradigm.

Lemma 6.20. *Suppose all honest committees input non-$\perp$ values, the leader of the committee, p_ℓ, is honest, and p_ℓ receives* acquire($\perp, \perp, acc_\ell \neq \perp$). *Then, acc_ℓ is a faulty committee.*

We are now ready to prove Sender-Validity.

Lemma 6.21 (Sender-Validity). *If all honest committees input non-$\perp$ values and p_ℓ, the leader of the committee, is honest, then:*

- *all honest committees acquire a value, or*
- *p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.*

Proof. As shown in Lemma 6.18, it is possible for p_ℓ to acquire some value v'. Furthermore, if p_ℓ does so, from Lemma 6.19, all honest committees will acquire v' as well. Meanwhile, if p_ℓ does not acquire some value, it must be alarmed of a process acc_ℓ at line 41. By Lemma 6.20, acc_ℓ must be a faulty committee. □

As all properties are satisfied, assuming Assumption 1, the module correctly implements alarm-certification for IC. Next, we can see that in each round, honest processes collectively send $O(n^2(L + \kappa))$ bits (for L-bits inputs). We then conclude with the following.

Lemma 6.22. *Assuming Assumption 1, there is an alarm-certification for IC with $O(1)$ latency and $O(n^2(L + \kappa))$ communication.*

6.2 Alarm-VBA

Alarm-VBA, in short, is validated Byzantine agreement (Byzantine agreement with external validity) amongst committee that also alarm a process with a potentially faulty process upon failing to decide. Here, a value is valid if it is accompanied by a certificate (which came from the alarm-certification). Next, we also need a decision to be accompanied with a *decision proof* – a threshold signature signed by everyone in the (current) committee proving that only that value can be decided. Let us stress that it is crucial to remember that (1) although the agreement is done amongst the committee, for technical reasons, we need all processes (including those not in the committee) to "participate", and (2) an instance of this module may be invoked (retried) multiple times.

This module exposes the following interfaces:

- *request* propose($Committee \subseteq \Pi, v \in \text{Value} \cup \{\bot\}, cert \in \text{Certificate} \cup \{\bot\}$): a process participates in the VBA, while inputting v with its certificate (if $v \neq \bot$).
- *indication* decide($v' \in \text{Value} \cup \{\bot\}, proof \in \text{Value} \cup \{\bot\}, acc \in \Pi \cup \{\bot\}$): if $v' \neq \bot$, a process decides on a value v' along with its decision proof. Otherwise, it is alarmed of a possibly faulty process acc.

Next, this module assumes the following:

- (*Valid Configuration*) Each honest process has the same committee. Moreover, there is at least one honest process in the committee.
- (*Subset Committee*) Let c, c' be the two committees that an honest process inputs in two consecutive invocations. Then, $c' \subseteq c$.

Lastly, This module guarantees the following properties:

- (*External-Validity*) If an honest process decides v', it also receives the decision proof for v' and certificate for v' exists.
- (*Termination*) All honest processes eventually receive decide indication at the same time.
- (*Decide-or-Accuse*) Each honest committee either receives decide($v' \neq \bot, proof, \bot$) or decide($\bot, \bot, acc \neq \bot$).
- (*Alarm-Priority*) If an honest committee proposes $\bot$, then all honest processes will be alarmed.
- (*Sender-Validity*) If all honest committees propose non-$\bot$ values and p_ℓ, the leader of the committee, is honest, then:
  - all honest committees receive decide on a value, or
  - p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.
- (*Agreement*) Suppose a decision proof for v' exists. Then, if a decision proof for v^* exists, $v' = v^*$.

Our implementation closely follows a single view of the algorithm from [39] (hereafter referred to as **CKS**), where the leader first collects the proposals and committed values from processes, and then leads the creation of commit proof and decision proof from the "freshest" commit value if one exists, or any proposal otherwise. A key challenge here is to ensure agreement, even if the module might be invoked (retried) multiple times, potentially with a different committee. Our approach to this is by having all n' processes to sign commit and decision proof, rather than just $\frac{n'+t'+1}{2}$ as in **CKS**. In a nutshell, this ensures that if a decision happens, an honest committee that commits to the decided value always exist in the committee. Lastly, accusation is made following KFD paradigm: a non-leader may accuse the leader, while the leader accuses a committee it observed of being silent.

Before describing our concrete implementation in more detail, let us first formally describe the two cryptographic objects we have mentioned above: *commit proof* and *decision proof*. Under a fixed committee of size n', a commit proof of *rank* r for a value v, σ, is a threshold signature signed by all

processes in the committee such that $\mathsf{CombinedVerify}^{n'}(\langle \textsc{commit}, v, r\rangle, \sigma)$ returns *true*. We say that a process is committed to a value v if the commit proof with the highest rank it has seen so far is the one for v. A decision proof σ is defined similarly, except it needs $\mathsf{CombinedVerify}^{n'}(\langle \textsc{decide}, v, r\rangle, \sigma)$ to return *true*. Let us note that the rank of a decision proof is only used internally in the alarm-VBA implementation.

Finally, our concrete implementation is presented at Algorithm 4. Let us note that the implementation is similar to a view of **CKS**, except that we also need to handle the committee aspect, a different threshold for signatures, and the capability to "alarm". All honest processes participate in each invocation, although a non-committee process would just stay silent (see the rule at line 3). As an instance of the module may be retried, we need to have variables which persist through multiple invocations of the instance. They are listed under persistent variables, which mirrors the variables that may change in **CKS** as we go through views. For example, $rank_i$, which stores how many times we have invoked the instance, is similar to the current view's number in **CKS**.

We now prove that the algorithm satisfies all properties guaranteed by the alarm-VBA module.

Lemma 6.23 (Termination). *All honest processes eventually receive* decide *indication at the same time.*

Proof. Immediate from the protocol. □

Lemma 6.24 (Decide-or-Accuse). *Each honest committee either receives* $\mathsf{decide}(v' \neq \bot, proof, \bot)$ *or* $\mathsf{decide}(\bot, \bot, acc \neq \bot)$.

Proof. See line 3, line 41, and line 44. Observe that, at line 44, *accused* must be well-defined. □

Lemma 6.25 (Alarm-Priority). *If an honest committee proposes* $\bot$*, then all honest processes will be alarmed.*

Proof. For a non-committee, this is ensured by the rule at line 3. Next, a decision proof needs all committees to sign. As an honest committee that proposes $\bot$ will not participate in the signing, no decision proof can be formed. □

Lemma 6.26 (External-Validity). *If an honest process decides* v'*, it also receives the decision proof for* v' *and certificate for* v' *exists.*

Proof. A process deciding v' must also receive the decision proof for v'. Next, a decision proof for v' must be signed by everyone in the committee. Via the assumptions, at least one honest committee must have signed it. Meanwhile, an honest committee only signs a partial signature for $proof$ if it received a commit proof for v'. Using the same observation, some honest process in the committee must have signed a partial signature for the commit certificate, which it only does so if there is a certificate for v'. □

Lemma 6.27 (Agreement). *Suppose a decision proof for* v' *exists. Then, if a decision proof for* v^* *exists,* $v' = v^*$.

Proof. Observe that each honest process p_i always has the same $rank_i$.

Let us focus on the first invocation where a decision proof is formed, and let v' be the value it corresponds to. Let r be the number of times this instance of the module has been invoked, i.e., $rank_i = r$ for each honest process p_i. As there is at least one honest committee, there is at least one honest process that signed the decision proof. Furthermore, such process must have stored a commit proof for v'. Let us note that during this invocation, only decision proof for v' can be formed.

Algorithm 4 Alarm-VBA: Pseudocode (for process p_i)

Rules:
  * when p_i invokes this primitive, it increments $rank_i$ by one
  * then, if $p_i \notin Committee_i$ or $v_i = \perp$, p_i stays silent and in the end, triggers decide($\perp, \perp$, Leader($Committee_i$))

Input parameters:
  Set(Process) $Committee_i$
  Value $\cup \{\perp\}$ v_i ▷ can be $\perp$
  Certificate $\cup \{\perp\}$ $cert_i$ ▷ if $v_i \neq \perp$, a certificate for value v_i

Persistent variables: ▷ to store any commit from previous invocations of this instance
  CommitProof $\cup \{\perp\}$ $commit_proof_i \leftarrow \perp$
  Value $\cup \{\perp\}$ $commit_value_i \leftarrow \perp$
  Integer $commit_rank_i \leftarrow 0$
  Integer $rank_i \leftarrow 0$ ▷ how many times this instance has been invoked

Round 1:
  send $\langle commit_value_i, commit_rank_i, commit_proof_i \rangle$ to Leader($Committee_i$)

Round 2:
  if p_i = Leader($Committee_i$):
    let $v, rank, cp \leftarrow$ commit value, commit rank, and commit proof from the messages received by the end of the round 1 (pick any with the highest rank)
    if $rank > 0$: ▷ use committed value
      multicast $\langle$PROPOSAL, $v, rank, cp\rangle$ to $Committee_i$
    else: ▷ use own proposal
      multicast $\langle$PROPOSAL, $v_i, cert\rangle$ to $Committee_i$

Round 3:
  if received valid $\langle$PROPOSAL, $v, rank, cp\rangle$ from Leader($Committee_i$) and $rank \geq commit_rank_i$, or received valid $\langle$PROPOSAL, $v, cert\rangle$ from Leader($Committee_i$) and $commit_rank_i = 0$:
    send $\langle$COMMIT, $v, rank_i$, ShareSign$_i^{n'}$ ($\langle$COMMIT, $val, rank_i\rangle$)$\rangle$ to Leader($Committee_i$) ▷ for $rank_i$

Round 4:
  if p_i = Leader($Committee_i$) and received valid $\langle$COMMIT, $v', rank', psig\rangle$ from all in $Committee_i$:
    let $cp' \leftarrow$ Combine$^{n'}$ ($\{psig \mid psig$ is partial signatures from all received messages$\}$)
    multicast $\langle$COMMIT_VAL, $v', rank', cp'\rangle$ to $Committee_i$

Round 5:
  if received valid $\langle$COMMIT_VAL, $v', rank', cp'\rangle$ from Leader($Committee_i$) and $rank' = rank_i$:
    $commit_value_i \leftarrow v'$
    $commit_rank_i \leftarrow rank'$
    $commit_proof_i \leftarrow cp'$
    send $\langle$DECIDE, $v', rank'$, ShareSign$_i^{n'}$ ($\langle$DECIDE, $v', rank'\rangle$)$\rangle$ to Leader($Committee_i$)

Round 6:
  if p_i = Leader($Committee_i$) and received valid $\langle$DECIDE, $v', rank', psig\rangle$ from all in $Committee_i$:
    let $decision_proof \leftarrow$ Combine$^{n'}$ ($\{psig \mid psig$ is partial signatures from all received messages$\}$)
    multicast $\langle$DECIDE_VAL, $v', rank', decision_proof\rangle$ to $Committee_i$

Round 7:
  if received valid $\langle$DECIDE_VAL, $decision, rank, proof\rangle$ from Leader($Committee_i$) and $rank = rank_i$:
    trigger decide($decision, proof, \perp$)
  else:
    let $accused$ be the first misbehaving committee p_i observed
    trigger decide($\perp, \perp, accused$)

We then argue that in any future invocation, no honest process ever partially signs a decision proof for $v^* \neq v'$. First, a decision proof for v^* requires a commit proof for v^* with rank $r^* > r$. Next, we note that when the decision proof for v' was formed, r was the highest rank possible for a commit proof. From the assumptions, there will always be an honest committee with a commit

proof for v' with rank r. Furthermore, such process will not sign a commit proof for $v^* \neq v'$ with rank $r^* > r$. Hence, no decision proof for some value $v^* \neq v'$ can be formed. □

To prove Sender-Validity, we start with some helper lemmas.

Lemma 6.28. *If all processes in the committee are honest and propose non-$\perp$ values, then each honest committee will receive* decide$(v' \neq \perp, proof, \cdot)$ *indication.*

Proof. No committees are silent. In round 2, p_ℓ will send proposal messages containing a value v' that will pass the check at line 23, and be accepted by all processes in the committee. Thus, the leader will be able to build a commit proof with the highest rank. Therefore, each honest committee will partially sign the decision proof for v', allowing p_ℓ to form a decision proof and ensuring all honest processes decide on v'. □

Lemma 6.29. *Suppose all honest committees propose non-$\perp$ values, the leader of the committee, p_ℓ, is honest, and p_ℓ receives* decide$(v' \neq \perp, proof, \cdot)$. *Then, all honest committees receive* decide$(v', proof, \cdot)$.

Proof. p_ℓ must have done so at line 41. Moreover, this means previously p_ℓ has multicast a value v' along with its decision proof to the committee at line 38. Thus, all honest committees will receive decide$(v', proof, \cdot)$. □

The next lemma follows from the idea of the KFD paradigm.

Lemma 6.30. *Suppose all honest committees propose non-$\perp$ values, the leader of the committee, p_ℓ, is honest, and p_ℓ receives* decide$(\perp, \perp, acc_\ell \neq \perp)$. *Then, acc_ℓ is a faulty committee.*

We are now ready to prove Sender-Validity.

Lemma 6.31 (Sender-Validity). *If all honest committees propose non-$\perp$ values and p_ℓ, the leader of the committee, is honest, then:*

- *all honest committees receive decide on a value, or*
- *p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.*

Proof. Lemma 6.28 shows that it is possible for p_ℓ to reach line 41 and decide on some value v'. Combined with Lemma 6.29, in this case, all honest committees will decide on a value.

Meanwhile, if p_ℓ fails to reach line 41, it must have reached line 44. By Lemma 6.30, p_ℓ will be alarmed of acc_ℓ and acc_ℓ is a faulty committee. □

As all properties are satisfied, the algorithm indeed implements the alarm-VBA module. Observe that the algorithm terminates in constant rounds and in each round, $O(n)$ messages are exchanged, each with $O(L + \kappa)$ bits (for L-sized proposals). We thus conclude with the following lemma.

Lemma 6.32. *There is an alarm-VBA with $O(1)$ latency and $O(n(L + \kappa))$ communication.*

6.3 Implementation & Analysis of Alarm-Gradecast

We now present our implementation of alarm-gradecast (Algorithm 5). The algorithm first tries to certify some value via alarm-certification, and then tries to agree on a certified value via alarm-VBA. When a process fails to decide, it is alarmed of a potentially faulty process that was received from any of the submodules. When there are multiple such processes, only the first one would be indicated by the alarm.

We now prove the correctness of the algorithm.

Lemma 6.33 (Termination). *All honest processes eventually receive* decide *indication at the same time.*

Algorithm 5 Alarm-Gradecast for Validity Property val: Pseudocode (for process p_i)

```
1:  Uses:
2:      AlarmCertification for val, instance 𝒜𝒞
3:      AlarmVBA, instance 𝒜𝒱ℬ𝒜
4:  Input parameters:
5:      Set(Process) Committee_i
6:      Value ∪ {⊥} v_i                                   ▷ might be ⊥
7:  Local variables:
8:      Process ∪ {⊥} accused_i ← ⊥                       ▷ first process that p_i suspects to be faulty
9:  invoke 𝒜𝒞.certify(Committee_i, v_i)
10: upon 𝒜𝒞.acquire(certified_value_i, cert_i, c_accused_i):
11:     accused_i ← c_accused_i
12:     invoke 𝒜𝒱ℬ𝒜.propose(Committee_i, certified_value_i, cert_i)
13: upon 𝒜𝒱ℬ𝒜.decide(decision_i, decision_proof_i, vba_accused_i):
14:     if accused_i = ⊥ : accused_i ← vba_accused_i
15:     if accused_i = ⊥: trigger decide(decision_i, decision_proof_i, ⊥)
16:     else: trigger decide(⊥, ⊥, accused_i)
```

Proof. Follows from the Termination property of the alarm-certification and alarm-VBA. □

Lemma 6.34 (Agreement). *Suppose a decision proof for v' exists. Then, if a decision proof for v^* exists, $v' = v^*$.*

Proof. Follows from the Agreement property of the alarm-VBA. □

Lemma 6.35 (Decide-or-Accuse). *Each honest committee either receives* decide$(v' \neq \bot, proof, \bot)$ *or* decide$(\bot, \bot, acc \neq \bot)$.

Proof. Follows from the Decide-or-Accuse property of the alarm-VBA along with the check at line 15. □

Lemma 6.36 (Alarm-Priority). *If an honest committee proposes $\bot$, then all honest processes will be alarmed.*

Proof. From the Alarm-Priority property of the alarm-certification, $accused_i \neq \bot$ for each honest process p_i, and all of them will be alarmed. □

Lemma 6.37 (Validity). *If an honest process decides v', then it also receives a decision proof for v' and v' is admissible according to* val.

Proof. From the External-Validity property of the alarm-VBA, a certificate for v' exists. The lemma then follows from the Validity of the alarm-certification. □

Lemma 6.38 (Sender-Validity). *If all honest committees propose non-$\bot$ values and p_ℓ, the leader of the committee, is honest, then:*

- *all honest committees decide on a value, or*
- *p_ℓ is alarmed of acc_ℓ and acc_ℓ is a faulty committee.*

Proof. By the Sender-Validity of the alarm-certification, if p_ℓ received certify$(\bot, \bot, acc_\ell)$, acc_ℓ must be a faulty committee. Furthermore, acc_ℓ must be a faulty committee.

Otherwise, Sender-Validity of the alarm-certification implies all honest committees receive an admissible value v' and its certificate. The lemma statement then follows from the Sender-Validity of the alarm-VBA. □

Thus, the algorithm correctly implements alarm-gradecast. We are now ready to prove the lemmas we promised at the start of the section.

Lemma 6.1. *There is an alarm-gradecast for SMVBA with $O(1)$ latency and $O(n(L+\kappa))$ communication.*

Lemma 6.2. *Assuming Assumption 1, there is an alarm-gradecast for IC with $O(1)$ latency and $O(n^2(L+\kappa))$ communication.*

Proof. We get Lemma 6.1 by instantiating the alarm-certification with Lemma 6.11 and the alarm-VBA with Lemma 6.32.

Next, by also assuming Assumption 1, the alarm-certification for IC from Lemma 6.22 will work following its specifications. Therefore, by instantiating the submodules with Lemma 6.22 and Lemma 6.32, we get Lemma 6.2. □

7 Committee Dissemination

Committee dissemination, as the name suggests, is a module that helps disseminating the committee's decision (which must be supported by a decision proof) to all processes.

The module exposes the following interfaces:

- *request* disseminate($Committee \subseteq \Pi$, $TG \subseteq \Pi \times \Pi$, $v \in \mathsf{Value} \cup \{\bot\}$, $proof \in \mathsf{DecisionProof} \cup \{\bot\}$): a process participates while inputting v' and its decision proof (if $v' \neq \bot$).
- *indication* acquire($v' \in \mathsf{Value} \cup \{\bot\}$): if $v' \neq \bot$, a process acquires a value v'.

Next, the module assumes the following:

- (*Valid Configuration*) Each honest process has the same committee and trust graph. Moreover, the committee has an honest majority and in the trust graph, no honest process distrusts any honest process.
- (*Unique Value*) If a decision proof for v and v' exists, then $v = v'$.

Finally, the module guarantees the following properties:

- (*Validity*) If an honest process acquires a value v', then a decision proof for v' exists.
- (*Termination*) All honest processes eventually receive acquire indication at the same time.
- (*Accuracy*) Implementation-specific guarantee when an honest non-committee process fails to acquire a value.

A key challenge here is to implement the module efficiently while ensuring accuracy still holds, even if some honest committees have not decided yet (the agreement amongst the committee may fail to decide with a faulty committee, see §6). We provide two implementations: one that is efficient but weaker accuracy guarantee, and another with stronger accuracy guarantee but less efficient. Notably, the first one is suitable for SMVBA, while the second one is suitable for IC.

7.1 Committee Dissemination for SMVBA

Our first committee dissemination (Algorithm 6) with a weaker accuracy guarantee (Property 1) is highly inspired by the trust graph usage in [98]. Here, a non-committee expects its forwarder from the committee (see §5) to forward the decision (supported by its proof). If no decision is received, then either the forwarder is faulty or the forwarder does not have the decision yet.

Property 1 (Accuracy (Weak) of Committee Dissemination)**.** *Let p_i be an honest process not in the committee and p_j be its forwarder. If p_i acquires $\bot$, then p_j is faulty or p_j inputs $(\cdot, \cdot, \bot, \bot)$.*

We first prove the correctness of the algorithm.

Algorithm 6 Committee Dissemination for SMVBA: Pseudocode (for process p_i)

```
1:  Input parameters:
2:      Set(Process) Committee_i
3:      TrustGraph TG_i
4:      Value ∪ {⊥} v_i
5:      DecisionProof ∪ {⊥} proof_i
6:  Round 1:
7:      if p_i ∈ Committee_i and v_i ≠ ⊥:
8:          multicast ⟨v_i, proof_i⟩ to ToForward(p_i, Committee_i, TG_i)      ▷ see Algorithm 1
9:  Round 2:
10:     if p_i ∈ Committee_i:
11:         trigger acquire(v_i)
12:     else:
13:         if received ⟨v, proof⟩ from Forwarder(p_i, Committee_i, TG_i) and proof is a decision proof for v:
14:             trigger acquire(v)
15:         else:
16:             trigger acquire(⊥)
```

Lemma 7.1 (Termination). *All honest processes eventually receive* acquire *indication at the same time.*

Proof. Straightforward from the protocol. □

Lemma 7.2 (Validity). *If an honest process acquires a value v', then a decision proof for v' exists.*

Proof. If the process is in the committee, this is immediate. Otherwise, a process only acquires a value it received from its forwarder if it is accompanied by a decision proof for the value. □

Lemma 7.3 (Accuracy (Property 1)). *Let p_i be an honest process not in the committee and p_j be its forwarder. If p_i acquires $\perp$, then p_j is faulty or p_j inputs $(\cdot, \cdot, \perp, \perp)$.*

Proof. By contradiction, suppose p_j is honest and inputs $(\cdot, \cdot, v \neq \perp, proof)$. Hence, p_j will send v and its decision proof to p_i. Therefore, p_i will accept v and acquire $v \neq \perp$. As we have a contradiction, the lemma statement holds. □

Hence, the algorithm correctly implements committee dissemination with accuracy as defined in Property 1. From how forwarders are determined, we can see that in total, honest processes will send $O(n)$ messages, each with $O(L + \kappa)$ bits (for L-bits decisions). We thus conclude with the following.

Lemma 7.4. *There is a committee dissemination which accuracy follows Property 1 with $O(1)$ latency and $O(n(L + \kappa))$ communication.*

7.2 Committee Dissemination for IC

Our second implementation, presented at Algorithm 7, has a stronger accuracy guarantee (Property 2) and resembles the committee dissemination from [35]. Here, we also need to have at most $t < n/2$ faulty processes (and not just $t' < n'/2$ in the committee).

Let us briefly explain the algorithm. An honest committee that has decided first compute n symbols using Reed-Solomon encoding, then sends the i-th symbol along with its partial signature to p_i. When p_i receives $t' + 1$ partial signatures for a symbol, as an honest committee must have sent it, the symbol must indeed be the i-th symbol of the decision. p_i then aggregate the partial signatures into a threshold signature that proves the symbol's correctness before broadcasting

the symbol and the signature. When a process receives $n - t$ symbols supported with a threshold signature, the process can then rebuild the decision.

Property 2 (Accuracy (Strong) of Committee Dissemination). *Let p_i be an honest process not in the committee. If p_i acquires $\perp$, then there is an honest process in the committee which inputs $(\cdot, \cdot, \perp, \perp)$.*

Algorithm 7 Committee Dissemination for IC: Pseudocode (for process p_i)

```
1:  Input parameters:
2:      Set(Process) Committee_i
3:      TrustGraph TG_i                                  ▷ not used, only here for the interface
4:      Value ∪ {⊥} v_i
5:      DecisionProof ∪ {⊥} proof_i                      ▷ not used, only here for the interface
6:  Round 1:
7:      if p_i ∈ Committee_i and v_i ≠ ⊥:
8:          let [m_1, m_2, ..., m_n] ← RSEnc(v_i, n, t + 1)
9:          for Integer j ← 1 to n:
10:             send ⟨j, m_j, ShareSign_i^{t'+1}(⟨j, m_j⟩)⟩ to p_j     ▷ t' is the maximum number of faults in Committee_i
11: Round 2:
12:     if received ⟨i, m', psig⟩ from t' + 1 processes in Committee_i:
13:         let tsig ← Combine^{t'+1}({psig | psig is from some received message})
14:         broadcast ⟨i, m', tsig⟩
15: Round 3:
16:     if p_i ∈ Committee_i:
17:         trigger acquire(v_i)
18:     else:
19:         if received ⟨j, m', tsig⟩ from n − t processes and each tsig validates m':
20:             for each 1 ≤ j ≤ n, denote m_j as m' received from p_j (m_j = ⊥ if p_j does not send anything)
21:             trigger acquire(RSDec(n − t, 0, {(j, m_j) | 1 ≤ j ≤ n}))
22:         else:
23:             trigger acquire(⊥)
```

We now prove the correctness of the algorithm.

Lemma 7.5 (Termination). *All honest processes eventually receive* acquire *indication at the same time.*

Proof. Straightforward from the protocol. □

Lemma 7.6 (Accuracy (Property 2)). *Let p_i be an honest process not in the committee. If p_i acquires $\perp$, then there is an honest process in the committee which inputs $(\cdot, \cdot, \perp, \perp)$.*

Proof. By contradiction, suppose all honest processes in the committee input $(\cdot, \cdot, v \neq \perp, \cdot)$. From the assumptions, they must all input the same value v.

At round 2, each honest process p_j must have received at least $t' + 1$ messages for the same share m_j. Thus, each p_j will be able to form a threshold signature *tsig* which validates m_j, and broadcast it. Therefore, at round 3, each honest process must have received at least $n - t$ shares accompanied with a threshold signature. Thus, all honest processes not in the committee can decode $v \neq \perp$ at line 21, a contradiction. Hence, there is an honest process in the committee which inputs $(\cdot, \cdot, \perp, \perp)$. □

Lemma 7.7 (Validity). *If an honest process acquires a value v', then a decision proof for v' exists.*

Proof. This is clear for an honest committee. For honest non-committee processes, note that it only acquires v' upon receiving at least $n - t$ RS symbols supported by a threshold signature. As the threshold signature must be signed by at least one honest committee, this property is implied by the assumptions. □

Thus, the algorithm correctly implements committee dissemination with accuracy property following Property 2. Observe that in each round, $O(n^2)$ messages are sent, each with $O(\frac{L}{n} + \log n + \kappa) = O(\frac{L}{n} + \kappa)$ bits (for L-bits values). We can then conclude with the following.

Lemma 7.8. *There is a committee dissemination for $t < n/2$ faulty processes with the accuracy property as defined in Property 2 with $O(1)$ latency and $O(nL + n^2\kappa)$ communication.*

8 Payload Phase

Payload phase is where consensuses are done in rarecomplain. This phase is parameterized by validity property val, in which each consensus instance it handles also has the same validity property. For a (fixed) constant V, payload phase of epoch e handles consensus instance $e(V-1)+1$ to eV. At the end of the payload phase, a process could also accuse another process of being faulty. Let us note that an epoch might be entered (retried) multiple times, as some processes may fail to decide when there is a faulty committee.

In each invocation of the payload phase, we assume the following:

- (*Valid Configuration*) Each honest process has the same committee and trust graph. Moreover, the committee has an honest majority and in the trust graph, no honest process distrusts any honest process.
- (*Subset Committee*) Let c, c' be the two committees that an honest process inputs in two consecutive invocations. Then, $c' \subseteq c$.

The payload phase then guarantees the following:

- (*Termination*) All honest processes eventually terminate at the same time.
- (*Validity*) If an honest process decides d in a consensus instance, then d is admissible according to val for that instance.
- (*Agreement*) If two honest processes decide d and d' in a consensus instance (possibly in different attempts), $d = d'$.
- (*Integrity*) An honest process accuses a process of being faulty if and only if it fails to decide in a consensus instance inside the epoch.
- (*Accuracy*) Suppose all honest processes have decided in all consensus instances of prior epochs.
  - If an honest committee accuses a process, then the leader is faulty or the leader accuses a faulty committee.
  - If an honest non-committee accuses a process, then its forwarder is faulty or there is an honest committee that accuses a process.

Our implementation (Algorithm 8) is a straightforward composition of the alarm-gradecast (§6) and committee dissemination (§7) for each consensus instance. The phase goes through each consensus instance in increasing order. For each instance, agreement amongst the committee is done via alarm-gradecast. If a process does not have the input for the consensus instance yet, it inputs $\perp$ to the module (line 13). If a committee process manages to decide for instance v, it then indicates its decision (which must be accompanied by a decision proof) for the instance (line 17). Then, all processes participate in the committee dissemination. Here, an honest committee inputs its decision and decision proof to the dissemination module if it has decided (line 21), and $\perp$ otherwise (line 22). If an honest non-committee process acquires a value this way, it then indicates

this value as the decision of the consensus instance (line 25). Finally, an honest process will accuse a process of being faulty when it fails to decide in a consensus instance, either the process detected by the alarm-gradecast (line 19) or the process' forwarder if no misbehaviour is detected in the alarm-gradecast (line 27). If there are multiple such instances, the process will just accuse the first one it observed.

Algorithm 8 Payload Phase for Epoch e: Pseudocode (for process p_i)

```
1:  Constants:
2:      Integer V                                              ▷ length of an epoch
3:  Uses:
4:      AlarmGradecast, instance AG⟨v⟩ for (e − 1)V + 1 ≤ v ≤ eV
5:      CommitteeDissemination, instance CD⟨v⟩ for (e − 1)V + 1 ≤ v ≤ eV
6:  Input parameters:
7:      Set(Process) Committee_i
8:      TrustGraph TG_i
9:  Local variables:
10:     Process accused_i ← ⊥                                  ▷ a process that p_i accuses to be faulty
11: for Instance v ← (e − 1)V + 1 to eV:
12:     Step 1 of instance v:
13:         let proposal_{i,v} be p_i's proposal for instance v if it p_i has decided in all prior instances, and ⊥ otherwise
14:         invoke AG⟨v⟩.propose(Committee_i, proposal_{i,v})
15:     Step 2 of instance v, upon AG⟨v⟩.decide(d_i, d_proof_i, ag_accused_i):
16:         if p_i ∈ Committee_i, has not decided for instance v, and d_i ≠ ⊥:
17:             trigger decide(v, d_i)                         ▷ decide for instance v, as a committee
18:         if p_i ∈ Committee_i, and has not decided in instance v, and accused_i = ⊥:
19:             accused_i ← ag_accused_i
20:         if p_i has decided d'_i for instance v (with d_proof'_i being its proof):   ▷ potentially from different tries
21:             invoke CD⟨v⟩.disseminate(Committee_i, TG_i, d'_i, d_proof'_i)
22:         else: invoke CD⟨v⟩.disseminate(Committee_i, TG_i, ⊥, ⊥)
23:     Step 3 of instance v, upon CD⟨v⟩.acquire(cd_out_i):
24:         if p_i ∉ Committee_i, has not decided for instance v, and cd_out_i ≠ ⊥:
25:             trigger decide(v, cd_out_i)                    ▷ decide for instance v, as a non-committee
26:         if p_i ∉ Committee_i, accused_i = ⊥, and has not decided for instance v:
27:             accused_i ← Forwarder(p_i, Committee_i, TG_i)
28: return (accused_i)                                         ▷ accused process, to be used in the complain phase
```

In the payload phase for SMVBA, we use alarm-gradecast from Lemma 6.1 and committee dissemination from Lemma 7.4. Meanwhile, in the payload phase for IC, we use alarm-gradecast from Lemma 6.2 and committee dissemination from Lemma 7.8. Let us note that the alarm-gradecast for IC requires an additional assumption (Assumption 1). To address that, we add one more assumption to the payload phase for IC (Assumption 2) and state another property guarantee that will be provided (Property 3). Let us note that this assumption implies the assumption needed by the alarm-gradecast for IC would be satisfied. Importantly, Property 3 and Integrity implies Assumption 2 and thus, the assumption would be satisfied in consecutive runs of the payload phase for IC. Finally, let us underline that this new property is a stronger Accuracy guarantee when an honest committee accuses a process.

Assumption 2. If an honest non-committee has not decided in all consensus instances of prior epochs, then there is an honest committee that has not decided in all consensus instances of prior epochs.

Property 3. *Suppose all honest processes have decided in all consensus instances in prior epochs. If an honest non-committee p_i returns $(acc_i \neq \perp)$, then there is an honest committee p_j that returns $(acc_j \neq \perp)$.*

We first prove the correctness of the algorithm. We start by showing that the assumptions required by the alarm-gradecast and committee dissemination are always satisfied.

Lemma 8.1 (Assumptions, Payload Phase for SMVBA). *The assumptions required by the alarm-gradecast from Lemma 6.1 and committee dissemination from Lemma 7.4 are satisfied.*

Proof. The assumptions regarding the committee and trust graph are satisfied by the assumptions made by the paylod phase. Hence, the assumptions of the alarm-gradecast are satisfied. Finally, the assumptions regarding the input value and decision proof needed by the committee dissemination follow from the Agreement and Validity properties of the alarm-gradecast. □

Lemma 8.2 (Assumptions, Payload Phase for IC). *Assuming Assumption 2, the assumptions required by the alarm-gradecast from Lemma 6.2 and committee dissemination from Lemma 7.8 are satisfied.*

Proof. The assumptions from each submodule regarding the committee set and trust graph are satisfied due to the assumptions on the payload phase. We now consider two cases.

Case 1: There is an honest process that has not decided in some prior epochs. In this case, the assumption required by the alarm-gradecast (Assumption 1) will always be satisfied. Thus, the submodule will work following its specifications. Therefore, the assumptions on the committee dissemination regarding the input values will always be satisfied from the Agreement and Validity properties of the alarm-gradecast.

Case 2: All honest processes have decided in all prior epochs. Observing the arguments in the previous case, it suffices to show that if an honest process fails to decide in an instance inside the epoch, there must be an honest committee that fails as well in that instance. Let v be the smallest such instance (thus, all honest processes have decided in all consensus instances before v). Hence, they all have the proposal for instance v. The assumptions needed for the alarm-gradecast of that instance are satisfied and similarly, the committee dissemination as well. If an honest committee fails to decide, we are done. Otherwise, suppose that an honest non-committee fails to decide. In this case, due to the Accuracy of the committee dissemination employed (see Property 2), there must be an honest committee that fails to decide. □

Hence, the submodules always work following their specifications. We are now ready to prove the algorithm satisfies each property.

Lemma 8.3 (Termination). *All honest processes eventually terminate at the same time.*

Proof. From the Termination property of each alarm-gradecast and committee dissemination. □

Lemma 8.4 (Agreement). *If two honest processes decide d and d' in a consensus instance (possibly in different attempts), $d = d'$.*

Proof. An honest committee can only receive this indication at line 17. From the Agreement property of the alarm-gradecast, for a consensus instance, only one value d^* may have a decision proof. Meanwhile, an honest non-committee can only receive the indication at line 25. By the Validity property of the committee dissemination, d^* is the only non-$\perp$ value that can be acquired. □

Lemma 8.5 (Validity). *If an honest process decides d in a consensus instance, then d is admissible according to* val *for that instance.*

Proof. As argued before, d must have a decision proof. The lemma then follows from the Validity property of the alarm-gradecast. □

Lemma 8.6 (Integrity). *An honest process accuses a process of being faulty if and only if it fails to decide in a consensus instance inside the epoch.*

Proof. Follows from line 19 and line 27. □

We now prove Accuracy and Property 3 in the following lemmas.

Lemma 8.7 (Accuracy, Honest Committee). *Suppose all honest processes have decided in all consensus instances of prior epochs. If an honest committee accuses a process, then the leader is faulty or the leader accuses a faulty committee.*

Proof. An honest committee sets its accused process when it first fails to decide within the epoch, at line 19. Let v be the first instance where a committee sets its accused process. Hence, all honest committees have decided in all instances before v. This means they all have the proposal for instance v. The lemma then follows from the Sender-Validity of the alarm-gradecast. □

Lemma 8.8 (Accuracy, Honest Non-Committee, Payload Phase for SMVBA). *Suppose all honest processes have decided in all consensus instances of prior epochs, and committee dissemination guarantees Property 1. If an honest non-committee accuses a process, then its forwarder is faulty or there is an honest committee that accuses a process.*

Proof. An honest non-committee p_i sets its accused process when it first fails to decide within the epoch, at line 27. By contradiction, assume there is no honest committee p_j that returns $(acc_j \neq \bot)$ and p_i's forwarder is honest. Therefore, all honest committees decide in all instances, and thus, input a value to the committee dissemination. However, from Property 1, p_i's forwarder is faulty. As we reach a contradiction, there must be an honest committee p_j that returns $(acc_j \neq \bot)$ or p_i's forwarder is faulty. □

Lemma 8.9 (Property 3, Payload Phase for IC). *Suppose all honest processes have decided in all consensus instances in prior epochs, and committee dissemination guarantees Property 2. If an honest non-committee p_i returns $(acc_i \neq \bot)$, then there is an honest committee p_j that returns $(acc_j \neq \bot)$.*

Proof. An honest non-committee p_i sets its accused process when it first fails to decide within the epoch, at line 27. Seeking contradiction, assume there is no honest committee p_j that returns $(acc_j \neq \bot)$. Thus, all honest committees decide in all consensus instances and always input a value to the committee dissemination. However, from Property 2, as p_i is unable to decide, there must be an honest committee that inputs $\bot$. As we reach a contradiction, there must be an honest committee p_j that returns $(acc_j \neq \bot)$. □

Thus, the implementations satisfy all properties guaranteed by the respective payload phase. Then, observe that messages are only sent inside the submodules. From the latency and communication complexities of the submodules, we get the following key results on the payload phase.

Lemma 8.10. *There is a payload phase protocol for SMVBA with V consensus instances that terminates in $O(V)$ rounds and exchanges $O(Vn(L+\kappa))$ bits.*

Lemma 8.11. *Assuming Assumption 2, there is a payload phase protocol for IC with V consensus instances that tolerates $t < n/2$ faulty processes which also guarantees Property 3. The protocol terminates in $O(V)$ rounds and exchanges $O(Vn^2(L+\kappa))$ bits.*

9 Complain Phase

Complain phase is where any accusations (complains) made during the payload phase is resolved by updating the committee and trust graph.

When running the complain phase, each honest process p_i starts with (1) the committee set, (2) the trust graph, and (3) a process it accuses in the last payload phase (or $\perp$, if p_i is not accusing anyone). In each complain phase execution, we assume the following:

- (*Valid Configuration*) Each honest process has the same committee and trust graph. Moreover, the committee has an honest majority and in the trust graph, no honest process distrusts any honest process.
- (*Accuracy*) Let p_ℓ be the leader of the committee. For each honest process p_i that accuses a process:
  - If p_i is in the committee, then p_ℓ is faulty or p_ℓ accuses a faulty committee.
  - If p_i is not in the committee, then its forwarder is faulty or there is an honest committee that accuses a process in the committee.

Let us note that the Accuracy assumption above would be satisfied by the Accuracy property of the payload phase.

Lastly, the complain phase guarantees the following properties:

- (*Termination*) All honest processes eventually terminate at the same time.
- (*Agreement*) Honest processes agree on the new committee and trust graph.
- (*Progress*) If an honest process complains, then the new committee or trust graph is different from the previous one.
- (*Accuracy*) The majority of the new committee is honest and no edge between honest processes is removed in the new trust graph.
- (*Containment*) The new committee (resp. trust graph) is a subset (resp. subgraph) of the previous committee (resp. trust graph).

In our implementation (Algorithm 9), complain phase is done via single-shot IC where each process inputs an accusation (if it has one). Specifically, (1) a leader can accuse anyone in the committee, (2) a committee process can accuse the leader of the committee, and (3) a non-committee process can accuse its forwarder in the committee. The new committee and trust graph is obtained from the previous one by applying one of the following rules based on the decided vector, prioritizing the earlier one:

(1) If the leader accuses a process in the committee, then both processes are removed from the committee.
(2) Else, if a process in the committee accuses the leader, then both processes are removed from the committee. (if there are multiple, pick one deterministically.)
(3) Else, for each process p_i that accuses its forwarder p_j, remove $p_i \longrightarrow p_j$ from the trust graph. (We can skip this rule in RARECOMPLAIN for IC which does not use trust graph.)

We now prove that the algorithm indeed satisfies all the properties.

LEMMA 9.1 (TERMINATION). *All honest processes eventually terminate at the same time.*

PROOF. Follows from the Termination property of the single-shot IC. □

LEMMA 9.2 (AGREEMENT). *Honest processes agree on the new committee and trust graph.*

PROOF. From the Agreement property of the single-shot IC, each honest process will receive the same vector. The output computation is fully deterministic, so all processes will output the same updated committee and trust graph. □

Algorithm 9 Complain Phase: Pseudocode (for process p_i)

```
1:  Uses:
2:      InteractiveConsistency, instance IC
3:  Input parameters:
4:      Set(Process) Committee_i
5:      TrustGraph TG_i
6:      Process ∪ {⊥} accused_i          ▷ if not ⊥, p_i failed to decide in some instances of the epoch
7:  Round 1:
8:      if accused_i = ⊥:                ▷ has decided in all instances of corresponding epoch
9:          invoke IC.propose(ALL_GOOD)
10:     else if p_i = Leader(Committee_i):   ▷ for leader, accuse accused_i as being faulty
11:         invoke IC.propose(⟨ACCUSE, accused_i⟩)
12:     else if p_i ∈ Committee_i:       ▷ for committee, accuse leader as being faulty
13:         invoke IC.propose(SACRIFICE)
14:     else:                            ▷ for non-committee, accuse forwarder
15:         invoke IC.propose(BAD_FORWARDER)
16: Round 1 + r^ic, upon IC.decide(vec):   ▷ r^ic ∈ O(n) is the latency of IC
17:         let Committee' ← Committee_i
18:         let TG' ← TG_i
19:         let ℓ be an integer such that p_ℓ = Leader(Committee_i)
20:         if vec[ℓ] = ⟨ACCUSE, p_j⟩ for some p_j ∈ Committee_i:   ▷ rule 1
21:             Committee' ← Committee' \ {p_ℓ, p_j}
22:         else if there exists j such that p_j ∈ Committee_i and vec[j] = SACRIFICE:   ▷ rule 2
23:             let j' be the smallest such j
24:             Committee' ← Committee' \ {p_ℓ, p_j'}
25:         else:                        ▷ rule 3
26:             for each p_j ∉ Committee_i:
27:                 if vec[j] = BAD_FORWARDER:
28:                     let p_k ← Forwarder(p_j, Committee_i, TG_i)
29:                     TG' ← TG' \ {⟨p_j → p_k⟩}
30:         return (Committee', TG')
```

Lemma 9.3 (Containment). *The new committee (resp. trust graph) is a subset (resp. subgraph) of the previous committee (resp. trust graph).*

Proof. The output is computed from the input where some processes may be removed from the committee and some edges may be removed from the trust graph. □

Lemma 9.4 (Progress). *If an honest process complains, then the new committee or trust graph is different from the previous one.*

Proof. Let the honest process be p_i. We consider two cases:

- p_i is in the committee. From the assumption, either of the first 2 rules will activate. Thus, the new committee will be different.
- p_i is not in the committee. From the assumptions, some honest committee accuses another committee or p_i's forwarder is faulty. In the first case, using the previous argument, the new committee will be different. Otherwise, the third rule will activate. Let p_j be p_i's forwarder. By the definition of forwarder, $\langle p_i \rightarrow p_j \rangle$ is in the input trust graph. Hence, the new trust graph will be different.

□

Lemma 9.5 (Accuracy). *The majority of the new committee is honest and no edge between honest processes is removed in the new trust graph.*

Proof. We prove each part of the lemma statement separately.

We start with proving the majority of the new committee is honest. From how the new committee may differ from the previous one, it suffices to show that when two processes p_i, p_j are removed from the committee due to the first 2 rules, p_i or p_j is faulty. By contradiction, assume that both are honest. However, by assumption on the inputs' Accuracy, either the leader p_ℓ is faulty or p_ℓ is accusing a faulty committee. In any case, both removed processes cannot be honest and thus, we reach a contradiction.

We now prove that no edge between honest processes is removed in the new trust graph. From how the new trust graph may differ from the previous one, it suffices to show that when an edge $\langle p_i \rightarrow p_j \rangle$ is removed from the trust graph due to the third rule, p_i or p_j is faulty. Next, observe that if the third rule activates, no committee is accusing anyone. Hence, from the inputs' Accuracy assumption, p_i or p_j must be faulty. □

Thus, all properties are satisfied. Next, notice that except in the single-shot IC used by the protocol, the protocol does not send any messages. Hence, by instantiating the single-shot IC with the one from [34] that can tolerate $t < n/2$ faulty processes, has $O(n)$ latency, and exchanges $O(n^2\kappa)$ bits (for $O(\log n)$-bits proposals), we conclude this section with the following result.

Lemma 9.6. *There is a complain phase protocol for $t < n/2$ faulty processes that terminates in $O(n)$ rounds and exchanges $O(n^2\kappa)$ bits.*

10 RARECOMPLAIN

In this section, we present the implementation details of RARECOMPLAIN along with its formal analysis.

We present the algorithm in Algorithm 10. In the algorithm, each process maintains its view of the committee, trust graph, and the current epoch everyone is currently at. Furthermore, a process also keeps track of whether it needs to run a complain phase as the last payload phase it executed may have issues (i.e., not satisfying the assumptions needed by the payload phase).

In a nutshell, the algorithm runs as follow: it consists of infinite iterations, where in each iteration, a process runs the payload phase for the current epoch. If it needs to run complain phase for the previous epoch, it runs the complain phase in parallel. After that, if the process does not run complain phase then it simply move to the next epoch. Otherwise, if the output of complain phase indicates issues with the previous epoch, i.e., the new committee or trust graph is different, then the process will go back to the previous epoch. Importantly, when going back, the process knows that in the next iteration, it does not need to run the complain phase of the current iteration's epoch since some assumptions needed by the payload phase might be not satisfied and so, the accusations made might not be accurate. (Regardless, validity and agreement would always be satisfied.)

Throughout the execution, the algorithm maintains the following invariants:

- (*Valid Configuration*) Each honest process has the same committee and trust graph. Moreover, the committee has an honest majority and in the trust graph, no honest process distrusts any honest process. Lastly, honest processes agree on what the current epoch is and whether they should run the complain phase. (For any two honest process p_i and p_j, $e_i = e_j$ and $do_complain_i = do_complain_j$.)
- (*Containment*) Let c, c' and tg, tg' be the two committees and trust graphs that an honest process have in two consecutive iterations. Then, $c' \subseteq c$ and tg' is a subgraph of tg.
- (*Finished Epochs*) If $do_complain_i = false$ for an honest process p_i, then every epoch up to $e_i - 1$ has finished. Otherwise, every epoch up to $e_i - 2$ has finished.

Algorithm 10 RARECOMPLAIN: Pseudocode (for process p_i)

```
1:  Uses:
2:      PayloadPhase, instance PP⟨e⟩ for each e ∈ ℕ                ▷ decisions are emitted here
3:      ComplainPhase, instance CP⟨e⟩ for each e ∈ ℕ
4:  Local variables:
5:      Set(Process) Committee_i ← Π                               ▷ current committee. initially contains everyone
6:      TrustGraph TG_i ← Π × Π                                    ▷ current trust graph. initially, each process trusts everyone
7:      Epoch e_i ← 1                                              ▷ current epoch
8:      Boolean do_complain_i ← false                              ▷ whether we need to run complain phase for previous epoch
9:      Process ∪ {⊥} accused_i ← ⊥                                ▷ process that p_i accuses of being faulty in the previous payload phase
10: for ever:
11:     do the followings in parallel:
12:         invoke PP⟨e_i⟩ with input (Committee_i, TG_i)          ▷ for consensus instances (e_i − 1)V + 1 to e_i V
13:         if do_complain_i = true:                               ▷ handle complains made in previous iteration
14:             invoke CP⟨e_i − 1⟩ with input (Committee_i, TG_i, accused_i)
15:     after PP⟨e_i⟩ (and CP⟨e_i − 1⟩ if invoked) terminate:
16:         accused_i ← PP⟨e_i⟩                                    ▷ potentially used in the next complain phase
17:         do_complain_i ← true
18:         if did not invoke complain phase in this iteration:
19:             e_i ← e_i + 1
20:         else:
21:             Committee′, TG′ ← CP⟨e_i − 1⟩
22:             if Committee_i = Committee′ and TG_i = TG′:        ▷ epoch e_i − 1 has finished
23:                 e_i ← e_i + 1
24:             else:                                              ▷ epoch e_i − 1 may not be finished yet
25:                 do_complain_i ← false                          ▷ no need to do complain phase as epoch e_i − 2 has finished
26:                 e_i ← e_i − 1
27:             Committee_i ← Committee′; TG_i ← TG′
```

We say that an epoch e is finished if all honest processes have decided in all consensus instances of epoch e. Observe that these invariants guarantee that the assumptions needed by the payload phase and complain phase would be satisfied. Moreover, the second invariant ensures there will be bounded number of epochs that we have to retry.

Next, beside the invariants above, the following invariant also hold in RARECOMPLAIN for IC:

- (*Undecided Honest Committee*) If an epoch $e' < e_i$ is not finished for an honest process p_i, then there is an honest committee which has not finished epoch e'.

In a nutshell, this invariant ensure the additional assumption needed by the payload phase for IC (Lemma 8.11) is satisfied.

We now prove that the invariants always hold in RARECOMPLAIN for SMVBA (i.e., without the additional invariant needed by RARECOMPLAIN for IC). Importantly, they all hold at the start of the execution.

PROOF. We consider two cases, based on whether the complain phase needs to be executed or not (the value of $do_complain_i$ for some honest process p_i).

Case 1: $do_complain_i = false$. Complain phase will not be invoked in this iteration. Therefore, the committee and trust graph would not change. Moreover, at the start of the next iteration, $do_complain_j = true$ for each honest process p_j. Hence, the Finished Epochs invariant will still hold. Finally, we can see that each honest process will have the same configuration.

Case 2: do_complain$_i$ = true. Complain phase will be invoked in this iteration. Importantly, the committee and trust graph in this iteration is the same as the previous iteration. Therefore, the payload phase of epoch $e_i - 1$ works following its specifications. Moreover, from the invariant, epoch $e_i - 2$ has finished and thus, the preconditions of Accuracy of the payload phase of epoch $e_i - 1$ are satisfied. Hence, the assumptions required for the complain phase invoked in this iteration will be satisfied. In any case, from the guarantees of the complain phase, the new committee and trust graph will be the subset and subgraph of the previous ones. Moreover, from the Agreement property of the complain phase, each honest process will start the next iteration with the same configuration. We are left to show that the Finished Epochs invariant still holds. We consider two cases:

- There is no change on the committee and trust graph. From the Integrity property of the payload phase and contrapositive of Progress property of the complain phase, all honest processes have decided in epoch $e_i - 1$. As line 23 is entered, the Finished Epochs invariant will hold in the next iteration.
- There is some change on the committee or trust graph. In this case, line 25 and line 26 will be entered, setting e_i into $e_i - 1$ and *do_complain*$_i$ to *false* for the next iteration. Hence, the Finished Epochs invariant still holds in the next iteration.

In any case, if all invariants hold at the start of an iteration, they will also hold at the start of the next iteration. □

Let us note that the proof above can be slightly modified to show that all invariants of RARECOMPLAIN for IC always hold, by taking into consideration the added invariant and Property 3 from the payload phase for IC (Lemma 8.11).

Next, we bound how many times RARECOMPLAIN goes back to the previous epoch. Recall that when RARECOMPLAIN does so, the committee loses two processes or at least one edge is removed from the trust graph. Furthermore, the trust graph is not used in RARECOMPLAIN for IC. Thus, we conclude the following.

LEMMA 10.1. *There are $O(n^2)$ (resp., $O(n)$) iterations where RARECOMPLAIN for SMVBA (resp., IC) needs to return to the previous epoch.*

Next, we instantiate the payload phases and complain phase with Lemma 8.10 (payload phase for SMVBA), Lemma 8.11 (payload phase for IC), and Lemma 9.6 (complain phase). Observe that, by setting the length of epoch, V, to be $\Theta(n)$, then the latency of both the complain and payload phase will be $O(n)$, and we can charge $O(\frac{n^2\kappa}{n}) = O(n\kappa)$ communication complexity to each consensus instance in an epoch. Therefore, we get the results that we promised upfront.

THEOREM 4.1. *There is an SMVBA protocol that tolerates $t < n/2$ faulty processes. Over $X \to \infty$ instances, the protocol exchanges $O\big((n^3+X)n(L+\kappa)\big)$ bits and ran over $O(X+n^3)$ rounds. Furthermore, except for $O(n^3)$ instances, each instance exchanges $O(n(L+\kappa))$ bits and terminates in $O(1)$ rounds.*

THEOREM 4.2. *There is an IC protocol that tolerates $t < n/2$ faulty processes. Over $X \to \infty$ instances, the protocol exchanges $O\big((n^2+X)n^2(L+\kappa)\big)$ bits and ran over $O(X+n^2)$ rounds. Furthermore, except for $O(n^2)$ instances, each instance exchanges $O(n^2(L+\kappa))$ bits and terminates in $O(1)$ rounds.*

11 Conclusion

In this paper, we present RARECOMPLAIN, a multi-shot Byzantine agreement protocol. By instantiating its submodules with different implementations, we achieve SMVBA and IC that is near-optimal which, except for a finite number of instances,

- the latency is optimal $O(1)$, and

- the communication complexity is near-optimal $O(n(L+\kappa))$ for SMVBA and $O(n^2(L+\kappa))$ for IC.

An interesting future direction is to reduce the number of "bad" instances and to reduce the additive communication complexity and latency from those instances. Another interesting direction is to see whether similar results can be obtained with little or no cryptography.

References

[1] Abraham, I., Asharov, G., Patra, A., and Stern, G. Perfectly secure asynchronous agreement on a core set in constant expected time. *IACR Cryptol. ePrint Arch.* (2023), 1130.

[2] Abraham, I., Chan, T. H., Dolev, D., Nayak, K., Pass, R., Ren, L., and Shi, E. Communication Complexity of Byzantine Agreement, Revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019* (2019), P. Robinson and F. Ellen, Eds., ACM, pp. 317–326.

[3] Abraham, I., Chan, T. H., Dolev, D., Nayak, K., Pass, R., Ren, L., and Shi, E. Communication complexity of byzantine agreement, revisited. *Distributed Comput. 36*, 1 (2023), 3–28.

[4] Abraham, I., and Dolev, D. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015* (2015), R. A. Servedio and R. Rubinfeld, Eds., ACM, pp. 605–614.

[5] Abraham, I., Malkhi, D., Nayak, K., Ren, L., and Yin, M. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), pp. 106–118.

[6] Abraham, I., Nayak, K., Ren, L., and Xiang, Z. Good-case Latency of Byzantine Broadcast: A Complete Categorization. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing* (2021), pp. 331–341.

[7] Abraham, I., Nayak, K., and Shrestha, N. Communication and round efficient parallel broadcast protocols. *IACR Cryptol. ePrint Arch.* (2023), 1172.

[8] Alhaddad, N., Duan, S., Varia, M., and Zhang, H. Succinct Erasure Coding Proof Systems. *Cryptology ePrint Archive* (2021).

[9] Arun, B., Li, Z., Suri-Payer, F., Das, S., and Spiegelman, A. Shoal++: High throughput dag bft can be fast!, 2024.

[10] Asharov, G., and Chandramouli, A. Perfect (parallel) broadcast in constant expected rounds via statistical VSS. In *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V* (2024), M. Joye and G. Leander, Eds., vol. 14655 of *Lecture Notes in Computer Science*, Springer, pp. 310–339.

[11] Bandarupalli, A., Ji, X., Kate, A., Liu-Zhang, C., and Song, Y. Concretely efficient asynchronous MPC from lightweight cryptography. *IACR Cryptol. ePrint Arch.* (2024), 1666.

[12] Bar-Noy, A., Deng, X., Garay, J. A., and Kameda, T. Optimal amortized distributed consensus. In *Distributed Algorithms* (Berlin, Heidelberg, 1992), S. Toueg, P. G. Spirakis, and L. Kirousis, Eds., Springer Berlin Heidelberg, pp. 95–107.

[13] Bar-Noy, A., Dolev, D., Dwork, C., and Strong, H. R. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1987* (1987), F. B. Schneider, Ed., ACM, pp. 42–51.

[14] Beerliová-Trubíniová, Z., and Hirt, M. Efficient multi-party computation with dispute control. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings* (2006), S. Halevi and T. Rabin, Eds., vol. 3876 of *Lecture Notes in Computer Science*, Springer, pp. 305–328.

[15] Ben-Or, M., Canetti, R., and Goldreich, O. Asynchronous Secure Computation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing* (1993), pp. 52–61.

[16] Ben-Or, M., Goldwasser, S., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA* (1988), J. Simon, Ed., ACM, pp. 1–10.

[17] Ben-Sasson, E., Fehr, S., and Ostrovsky, R. Near-linear unconditionally-secure multiparty computation with a dishonest minority. In *Advances in Cryptology - CRYPTO 2012 - 32nd Annual Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2012. Proceedings* (2012), R. Safavi-Naini and R. Canetti, Eds., vol. 7417 of *Lecture Notes in Computer Science*, Springer, pp. 663–680.

[18] Berman, P., Garay, J. A., and Perry, K. J. Towards optimal distributed consensus (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989* (1989), IEEE Computer Society, pp. 410–415.

[19] Bhangale, A., Liu-Zhang, C., Loss, J., and Nayak, K. Efficient adaptively-secure byzantine agreement for long messages. In *Advances in Cryptology - ASIACRYPT 2022 - 28th International Conference on the Theory and Application*

of Cryptology and Information Security, Taipei, Taiwan, December 5-9, 2022, Proceedings, Part I (2022), S. Agrawal and D. Lin, Eds., vol. 13791 of *Lecture Notes in Computer Science*, Springer, pp. 504–525.

[20] Bhangale, A., Liu-Zhang, C. D., Loss, J., and Nayak, K. Efficient Adaptively-Secure Byzantine Agreement for Long Messages. In *Advances in Cryptology - ASIACRYPT - 28th International Conference on the Theory and Application of Cryptology and Information Security* (Taipei, Taiwan, 2022), vol. 13791 LNCS, pp. 504–525.

[21] Blahut, R. E. *Theory and practice of error control codes*, vol. 126. Addison-Wesley Reading, 1983.

[22] Boyle, E., Cohen, R., and Goel, A. Breaking the o($\sqrt{}$ n)-bit barrier: Byzantine agreement with polylog bits per party. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021* (2021), A. Miller, K. Censor-Hillel, and J. H. Korhonen, Eds., ACM, pp. 319–330.

[23] Boyle, E., Cohen, R., and Goel, A. Breaking the $o(\sqrt{n})$-bit barrier: Byzantine agreement with polylog bits per party. *J. Cryptol. 37*, 1 (2024), 2.

[24] Boyle, E., Goldwasser, S., and Tessaro, S. Communication locality in secure multi-party computation - how to run sublinear algorithms in a distributed setting. In *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings* (2013), A. Sahai, Ed., vol. 7785 of *Lecture Notes in Computer Science*, Springer, pp. 356–376.

[25] Braud-Santoni, N., Guerraoui, R., and Huc, F. Fast byzantine agreement. In *ACM Symposium on Principles of Distributed Computing, PODC '13, Montreal, QC, Canada, July 22-24, 2013* (2013), P. Fatourou and G. Taubenfeld, Eds., ACM, pp. 57–64.

[26] Camenisch, J., Drijvers, M., Hanke, T., Pignolet, Y. A., Shoup, V., and Williams, D. Internet Computer Consensus. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing 2021* (2022), 81–91.

[27] Castro, M., and Liskov, B. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS) 20*, 4 (2002), 398–461.

[28] Chen, J. Optimal error-free multi-valued byzantine agreement. In *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)* (2021), S. Gilbert, Ed., vol. 209 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 17:1–17:19.

[29] Chen, J., and Micali, S. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci. 777* (2019), 155–183.

[30] Cheng, H., Lu, Y., Lu, Z., Tang, Q., Zhang, Y., and Zhang, Z. Jumbo: Fully asynchronous bft consensus made truly scalable. *arXiv preprint arXiv:2403.11238* (2024).

[31] Chongchitmate, W., and Ostrovsky, R. Information-theoretic broadcast with dishonest majority for long messages. In *Theory of Cryptography - 16th International Conference, TCC 2018, Panaji, India, November 11-14, 2018, Proceedings, Part I* (2018), A. Beimel and S. Dziembowski, Eds., vol. 11239 of *Lecture Notes in Computer Science*, Springer, pp. 370–388.

[32] Choudhury, A., Orsini, E., Patra, A., and Smart, N. P. Linear overhead optimally-resilient robust MPC using preprocessing. In *Security and Cryptography for Networks - 10th International Conference, SCN 2016, Amalfi, Italy, August 31 - September 2, 2016, Proceedings* (2016), V. Zikas and R. D. Prisco, Eds., vol. 9841 of *Lecture Notes in Computer Science*, Springer, pp. 147–168.

[33] Choudhury, A., and Patra, A. On the communication efficiency of statistically secure asynchronous MPC with optimal resilience. *J. Cryptol. 36*, 2 (2023), 13.

[34] Civit, P., Dzulfikar, M. A., Gilbert, S., Guerraoui, R., Komatovic, J., and Vidigueira, M. DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024* (2024), R. Gelles, D. Olivetti, and P. Kuznetsov, Eds., ACM, pp. 145–156.

[35] Civit, P., Dzulfikar, M. A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M., and Zablotchi, I. Efficient signature-free validated agreement. In *38th International Symposium on Distributed Computing, DISC 2024, October 28 to November 1, 2024, Madrid, Spain* (2024), D. Alistarh, Ed., vol. 319 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 14:1–14:23.

[36] Civit, P., Gilbert, S., Guerraoui, R., Komatovic, J., Monti, M., and Vidigueira, M. Every bit counts in consensus. In *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy* (2023), R. Oshman, Ed., vol. 281 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 13:1–13:26.

[37] Civit, P., Gilbert, S., Guerraoui, R., Komatovic, J., Paramonov, A., and Vidigueira, M. All byzantine agreement problems are expensive. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024* (2024), R. Gelles, D. Olivetti, and P. Kuznetsov, Eds., ACM, pp. 157–169.

[38] Cohen, R., Forghani, P., Garay, J. A., Patel, R., and Zikas, V. Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV* (2023), G. N. Rothblum and H. Wee, Eds., vol. 14372 of *Lecture Notes in Computer Science*, Springer, pp. 422–451.

[39] Cohen, S., Keidar, I., and Spiegelman, A. Make every word count: Adaptive BA with fewer words. *CoRR abs/2202.09123* (2022).

[40] Damgård, I., and Nielsen, J. B. Scalable and unconditionally secure multiparty computation. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings* (2007), A. Menezes, Ed., vol. 4622 of *Lecture Notes in Computer Science*, Springer, pp. 572–590.

[41] Danezis, G., Kokoris-Kogias, L., Sonnino, A., and Spiegelman, A. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems* (2022), pp. 34–50.

[42] Das, S., Duan, S., Liu, S., Momose, A., Ren, L., and Shoup, V. Asynchronous consensus without trusted setup or public-key cryptography. *Cryptology ePrint Archive* (2024).

[43] Das, S., Krishnan, V., Isaac, I. M., and Ren, L. Spurt: Scalable distributed randomness beacon with transparent setup. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022* (2022), IEEE, pp. 2502–2517.

[44] Das, S., Xiang, Z., and Ren, L. Asynchronous Data Dissemination and its Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (2021), pp. 2705–2721.

[45] Doidge, I., Ramesh, R., Shrestha, N., and Tobkin, J. Moonshot: Optimizing block period and commit latency in chain-based rotating leader bft. In *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (2024), pp. 470–482.

[46] Dolev, D., and Reischuk, R. Bounds on information exchange for Byzantine agreement. *Journal of the ACM (JACM) 32*, 1 (1985), 191–204.

[47] Dolev, D., and Strong, H. R. Authenticated Algorithms for Byzantine Agreement. 656–666.

[48] Duan, S., Zhang, H., and Zhao, B. Waterbear: Information-theoretic asynchronous BFT made practical. *IACR Cryptol. ePrint Arch.* (2022), 21.

[49] Elsheimy, F., Tsimos, G., and Papamanthou, C. Deterministic byzantine agreement with adaptive $O(n \cdot f)$ communication. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024, Alexandria, VA, USA, January 7-10, 2024* (2024), D. P. Woodruff, Ed., SIAM, pp. 1120–1146.

[50] Feng, H., Lu, Z., Mai, T., and Tang, Q. Making hash-based mvba great again. *Cryptology ePrint Archive* (2024).

[51] Fernando, R., Gelles, Y., and Komargodski, I. Scalable distributed agreement from LWE: byzantine agreement, broadcast, and leader election. In *15th Innovations in Theoretical Computer Science Conference, ITCS 2024, January 30 to February 2, 2024, Berkeley, CA, USA* (2024), V. Guruswami, Ed., vol. 287 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 46:1–46:23.

[52] Fischer, M. J., Lynch, N. A., and Merritt, M. Easy Impossibility Proofs for Distributed Consensus Problems. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985* (1985), M. A. Malcolm and H. R. Strong, Eds., ACM, pp. 59–70.

[53] Gągol, A., Leśniak, D., Straszak, D., and Świętek, M. ALEPH: Efficient atomic broadcast in asynchronous networks with Byzantine nodes. *AFT 2019 - Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (2019), 214–228.

[54] Ganesh, C., and Patra, A. Optimal extension protocols for byzantine broadcast and agreement. *Distributed Comput. 34*, 1 (2021), 59–77.

[55] Garay, J. A., and Moses, Y. Fully polynomial byzantine agreement in t+1 rounds. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA* (1993), S. R. Kosaraju, D. S. Johnson, and A. Aggarwal, Eds., ACM, pp. 31–41.

[56] Gelles, Y., and Komargodski, I. Optimal load-balanced scalable distributed agreement. In *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024* (2024), B. Mohar, I. Shinkar, and R. O'Donnell, Eds., ACM, pp. 411–422.

[57] Giridharan, N., Suri-Payer, F., Abraham, I., Alvisi, L., and Crooks, N. Autobahn: Seamless high speed bft. In *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles* (New York, NY, USA, 2024), SOSP '24, Association for Computing Machinery, p. 1–23.

[58] Goldreich, O., Micali, S., and Wigderson, A. How to play any mental game or A completeness theorem for protocols with honest majority. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA* (1987), A. V. Aho, Ed., ACM, pp. 218–229.

[59] Goyal, V., Liu, Y., and Song, Y. Communication-efficient unconditional MPC with guaranteed output delivery. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part II* (2019), A. Boldyreva and D. Micciancio, Eds., vol. 11693 of *Lecture Notes in Computer Science*, Springer, pp. 85–114.

[60] Goyal, V., Song, Y., and Zhu, C. Guaranteed output delivery comes free in honest majority MPC. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II* (2020), D. Micciancio and T. Ristenpart, Eds., vol. 12171 of *Lecture Notes in Computer Science*, Springer, pp. 618–646.

[61] Haeberlen, A., and Kuznetsov, P. The fault detection problem. In *Principles of Distributed Systems, 13th International*

Conference, OPODIS 2009, Nîmes, France, December 15-18, 2009. Proceedings (2009), T. F. Abdelzaher, M. Raynal, and N. Santoro, Eds., vol. 5923 of *Lecture Notes in Computer Science*, Springer, pp. 99–114.

[62] Hendricks, J., Ganger, G. R., and Reiter, M. K. Verifying distributed erasure-coded data. In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Principles of Distributed Computing, PODC 2007, Portland, Oregon, USA, August 12-15, 2007* (2007), I. Gupta and R. Wattenhofer, Eds., ACM, pp. 139–146.

[63] Hirt, M., and Maurer, U. M. Robustness for free in unconditional multi-party computation. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings* (2001), J. Kilian, Ed., vol. 2139 of *Lecture Notes in Computer Science*, Springer, pp. 101–118.

[64] Hirt, M., Maurer, U. M., and Przydatek, B. Efficient secure multi-party computation. In *Advances in Cryptology - ASIACRYPT 2000, 6th International Conference on the Theory and Application of Cryptology and Information Security, Kyoto, Japan, December 3-7, 2000, Proceedings* (2000), T. Okamoto, Ed., vol. 1976 of *Lecture Notes in Computer Science*, Springer, pp. 143–161.

[65] Hirt, M., and Nielsen, J. B. Robust multiparty computation with linear communication complexity. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings* (2006), C. Dwork, Ed., vol. 4117 of *Lecture Notes in Computer Science*, Springer, pp. 463–482.

[66] Hirt, M., and Raykov, P. Multi-valued byzantine broadcast: The t < n case. In *Advances in Cryptology - ASIACRYPT 2014 - 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, R.O.C., December 7-11, 2014, Proceedings, Part II* (2014), P. Sarkar and T. Iwata, Eds., vol. 8874 of *Lecture Notes in Computer Science*, Springer, pp. 448–465.

[67] Huang, S., Pettie, S., and Zhu, L. Byzantine agreement in polynomial time with near-optimal resilience. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing, Rome, Italy, June 20 - 24, 2022* (2022), S. Leonardi and A. Gupta, Eds., ACM, pp. 502–514.

[68] Huang, S., Pettie, S., and Zhu, L. Byzantine Agreement with Optimal Resilience via Statistical Fraud Detection. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023* (2023), N. Bansal and V. Nagarajan, Eds., SIAM, pp. 4335–4353.

[69] Keidar, I., Kokoris-Kogias, E., Naor, O., and Spiegelman, A. All you need is DAG. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021* (2021), A. Miller, K. Censor-Hillel, and J. H. Korhonen, Eds., ACM, pp. 165–175.

[70] King, V., and Saia, J. From almost everywhere to everywhere: Byzantine agreement with $\tilde{o}(n^{3/2})$ bits. In *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings* (2009), I. Keidar, Ed., vol. 5805 of *Lecture Notes in Computer Science*, Springer, pp. 464–478.

[71] King, V., and Saia, J. Breaking the $O(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. In *Proceedings of the 29th Annual ACM Symposium on Principles of Distributed Computing, PODC 2010, Zurich, Switzerland, July 25-28, 2010* (2010), A. W. Richa and R. Guerraoui, Eds., ACM, pp. 420–429.

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

[73] Kotla, R., Alvisi, L., Dahlin, M., Clement, A., and Wong, E. L. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007* (2007), T. C. Bressoud and M. F. Kaashoek, Eds., ACM, pp. 45–58.

[74] Küsters, R., Truderung, T., and Vogt, A. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010* (2010), E. Al-Shaer, A. D. Keromytis, and V. Shmatikov, Eds., ACM, pp. 526–535.

[75] Lewis-Pye, A. Quadratic worst-case message complexity for State Machine Replication in the partial synchrony model. *arXiv preprint arXiv:2201.01107* (2022).

[76] Liang, G., and Vaidya, N. Error-free multi-valued consensus with byzantine failures. In *Proceedings of the 30th Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing* (New York, NY, USA, 2011), PODC '11, Association for Computing Machinery, p. 11–20.

[77] Libert, B., Joye, M., and Yung, M. Born and Raised Distributively: Fully Distributed Non-Interactive Adaptively-Secure Threshold Signatures with Short Shares. *Theoretical Computer Science 645* (2016), 1–24.

[78] Lindell, Y., Lysyanskaya, A., and Rabin, T. Sequential composition of protocols without simultaneous termination. In *Proceedings of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing, PODC 2002, Monterey, California, USA, July 21-24, 2002* (2002), A. Ricciardi, Ed., ACM, pp. 203–212.

[79] Lu, Y., Lu, Z., Tang, Q., and Wang, G. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. *Proceedings of the Annual ACM Symposium on Principles of Distributed Computing* (2020), 129–138.

[80] Momose, A. Practical asynchronous MPC from lightweight cryptography. *IACR Cryptol. ePrint Arch.* (2024), 1717.

[81] Moses, Y., and Waarts, O. Coordinated traversal: (t + 1)-round byzantine agreement in polynomial time. In *29th*

Annual Symposium on Foundations of Computer Science, White Plains, New York, USA, 24-26 October 1988 (1988), IEEE Computer Society, pp. 246–255.

[82] Nayak, K., and Abraham, I. The Dolev and Reischuk Lower Bound: Does Agreement need Quadratic Messages?, 2019.

[83] Nayak, K., Ren, L., Shi, E., Vaidya, N. H., and Xiang, Z. Improved extension protocols for byzantine broadcast and agreement. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference* (2020), H. Attiya, Ed., vol. 179 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 28:1–28:17.

[84] Patra, A., and Rangan, C. P. Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience. In *Information Theoretic Security - 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings* (2011), S. Fehr, Ed., vol. 6673 of *Lecture Notes in Computer Science*, Springer, pp. 206–226.

[85] Pfitzmann, B., and Waidner, M. *Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$*. Citeseer, 1996.

[86] Prabhu, B., Srinathan, K., and Rangan, C. P. Trading players for efficiency in unconditional multiparty computation. In *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers* (2002), S. Cimato, C. Galdi, and G. Persiano, Eds., vol. 2576 of *Lecture Notes in Computer Science*, Springer, pp. 342–353.

[87] Rambaud, M. Adaptively secure consensus with linear complexity and constant round under honest majority in the bare PKI model, and separation bounds from the idealized message-authentication model. *IACR Cryptol. ePrint Arch.* (2023), 1757.

[88] Rambaud, M., Tonkikh, A., and Abspoel, M. Linear View Change in Optimistically Fast BFT. In *Proceedings of the 2022 ACM Workshop on Developments in Consensus* (2022), pp. 67–78.

[89] Reed, I. S., and Solomon, G. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics 8*, 2 (1960), 300–304.

[90] Shoup, V. Practical threshold signatures. In *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding* (2000), B. Preneel, Ed., vol. 1807 of *Lecture Notes in Computer Science*, Springer, pp. 207–220.

[91] Shoup, V. A theoretical take on a practical consensus protocol. *Cryptology ePrint Archive* (2024).

[92] Shoup, V., and Smart, N. P. Lightweight asynchronous verifiable secret sharing with optimal resilience. *J. Cryptol. 37*, 3 (2024), 27.

[93] Shrestha, N., Shrothrium, R., Kate, A., and Nayak, K. Sailfish: Towards improving the latency of DAG-based BFT. Cryptology ePrint Archive, Paper 2024/472, 2024.

[94] Song, Y., and Ye, X. Perfectly-secure MPC with constant online communication complexity. In *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part IV* (2024), E. Boyle and M. Mahmoody, Eds., vol. 15367 of *Lecture Notes in Computer Science*, Springer, pp. 329–361.

[95] Spiegelman, A. In Search for an Optimal Authenticated Byzantine Agreement. In *35th International Symposium on Distributed Computing (DISC 2021)* (Dagstuhl, Germany, 2021), S. Gilbert, Ed., vol. 209 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 38:1–38:19.

[96] Srikanth, T. K., and Toueg, S. Optimal clock synchronization. In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985* (1985), M. A. Malcolm and H. R. Strong, Eds., ACM, pp. 71–86.

[97] Sui, X., and Duan, S. Signature-free atomic broadcast with optimal $O(n^2)$ messages and $O(1)$ expected time. *Cryptology ePrint Archive* (2023). https://eprint.iacr.org/2023/1549.

[98] Wan, J., Momose, A., Ren, L., Shi, E., and Xiang, Z. On the Amortized Communication Complexity of Byzantine Broadcast. In *Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing* (2023), pp. 253–261.

[99] Wan, J., Xiao, H., Shi, E., and Devadas, S. Expected constant round byzantine broadcast under dishonest majority. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I* (2020), R. Pass and K. Pietrzak, Eds., vol. 12550 of *Lecture Notes in Computer Science*, Springer, pp. 381–411.

[100] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., and Abraham, I. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (2019), pp. 347–356.

[101] Yurek, T., Luo, L., Fairoze, J., Kate, A., and Miller, A. hbacss: How to robustly share many secrets. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28, 2022* (2022), The Internet Society.

A Notes on Cryptography

In this section, we go through some subtle details on our usages of cryptography.

Threshold signatures in committee. As noted in §3, we also use threshold signatures in the committee. However, our committee set may also be updated over time. Thus, each time we update our committee, we have to set up a new threshold signature scheme, with the updated signer set and thresholds. As the committee set can only be updated $O(n)$ times, this only happens $O(n)$ times as well and thus, will not affect the amortized complexity.

On wrong accusation and multiple instances. In §4 and §6, we mentioned that an honest process may wrongly accuse another honest process, but this will not affect the correctness of the protocol. However, we assume that the accusation *cannot be used* by the adversary in other instances or invocations of the protocols. Moreover, this problem is not unique to accusations; for example, the adversary may use a signed message by p_i disclosing its proposal in a consensus instance to break the correctness in other instances.

In other words, for some signed messages we want them to be *only valid in a consensus instance* or even *only valid in an invocation of a module*. Both can be solved by adding a unique identifier to the signed messages. For example, if we want a message to be valid for a specific consensus instance, we can add the consensus instance number as the unique identifier.