# Constant-Round Interactive Consistency in Dishonest Majority, or with Quadratic Communication in Honest Majority

Pierre Civit
pierre.civit@epfl.ch
EPFL
(École polytechnique fédérale de Lausanne)
Lausanne, Vaud, Switzerland

Grigorii Emdin
grigorii.emdin@epfl.ch
EPFL
(École polytechnique fédérale de Lausanne)
Lausanne, Vaud, Switzerland

Rachid Guerraoui
rachid.guerraoui@epfl.ch
EPFL
(École polytechnique fédérale de Lausanne)
Lausanne, Vaud, Switzerland

## Abstract

We present the first constant-expected-latency protocols for Interactive Consistency (IC), also known as Parallel Byzantine Broadcast, that achieve either: (1) security in dishonest majority, namely $t \leq (1-\varepsilon)n$ for any constant $\varepsilon \in \Omega(1)$; or (2) quadratic communication $O\big(n^2(L_{in}+\kappa)\big)$ in honest majority (i.e., $\varepsilon > 1/2$). In IC, $n$ processes must agree on a vector that maps every honest process to its input of size $L_{in}$, despite up to $t$ dishonest (Byzantine) processes that may collude and behave arbitrarily. IC is the strongest one-shot distributed task: by fully determining the honest input configuration, it subsumes every other solvable one-shot task in the same model. Moreover, most multiparty computation protocols rely on IC as a building block.

These guarantees are qualitatively optimal. First, Garay, Katz, Koo, and Ostrovsky (FOCS 2007) rule out constant-round protocols unless the honest fraction is constant. Second, Pease, Shostak, and Lamport (JACM 1980) rule out setup-free information-theoretic solutions once $n \leq 3t$. Thus, to overcome this barrier, we rely on cryptographic objects of size $\kappa$, obtaining correctness with all but negligible probability in $\kappa$ against any adversary running in time polynomial in $\kappa$. This includes digital signatures, for which the corresponding public keys must be published on a bulletin-board public key infrastructure before the protocol begins. Third, IC trivially requires $\Omega(n^2 L_{in})$ communication, since every honest process must learn the inputs of all honest processes.

Our results follow from a single generic compiler that transforms any constant-expected-latency Byzantine Broadcast protocol into an IC protocol with the same latency profile.

## CCS Concepts

• **Theory of computation → Distributed algorithms**; • **Security and privacy → Distributed systems security**.

## Keywords

Parallel Byzantine Broadcast, Interactive Consistency, Constant Round Complexity, Quadratic Communication Complexity, Dishonest Majority, Honest Majority

## 1 Introduction

***Interactive Consistency: the strongest distributed (agreement) task.*** Secure distributed computing [18] studies how $n$ processes can cooperate despite the presence of up to $t$ malicious ones. Agreement is typically necessary and sufficient for many services of interest, such as blockchain systems, where disagreement may be catastrophic, for example through double-spending. An agreement task requires two basic properties: *Consistency* and *Termination*, meaning that (after providing their inputs), processes should eventually return a common output. The task becomes meaningful because of a third property, *Validity*, which rules out the trivial solution where processes always return a meaningless default value, independently of their inputs.

The strongest agreement problem is *Interactive Consistency* (IC). In IC, processes decide a vector vec of $n$ entries, where $\text{vec}[i]$ must equal the input of $p_i$ whenever $p_i$ is honest. In other words, all honest inputs are explicitly included in the decided vector, removing any ambiguity about the input configuration. This makes IC the strongest (one-shot) distributed task.

***Computability and historical perspective.*** IC was introduced in [74], together with a tight computability landscape: resiliency requires $n > 3t$ in the information-theoretic setting (this lower bound holds even for the weakest validity property [27, 45]); with transferable authentication, IC becomes solvable for any $n \geq t$[1]. Already in [74], IC is solved through $n$ parallel instances of *Byzantine Broadcast* (BB), an agreement task parameterized by a designated source and satisfying source-validity[2]: if the source is honest, only its input can be decided. This explains the other name of IC: *parallel broadcast*. The original [74]'s solution relies on Exponential Information Gathering trees and incurs exponential computation in $n$, whereas Dolev–Strong [41] gives the first polynomial-time BB protocol (with $t+1$ rounds and cubic communication).

As the strongest one-shot distributed task, IC (and thus BB) is among the most fundamental problems in secure distributed computing. It appears as a subroutine in long-lived atomic broadcast or state machine replication, and in multi-party computation (MPC), where additional privacy and input-independence concerns arise. Cleve's impossibility result [28] rules out general MPC with guaranteed output delivery when $n \leq 2t$, but this barrier can be mitigated through MPC with abort [11, 32, 64], which still requires IC in the dishonest-majority setting. For this reason, IC and BB have been intensively studied.

[1] Readers may instead recall an $n/2$-resiliency lower bound [4]. This lower bound applies to Byzantine agreement with *Strong Unanimity Validity* (SBA), where if all processes propose the same value, then only that value may be returned.

[2] In honest majority, BB reduces to SBA by letting every process submit to SBA the value heard from the source in the first round. Conversely, SBA can be obtained from IC, and therefore from BB, by returning the most represented value in the vector output by IC. In dishonest majority, however, SBA is impossible, even if IC is given for free, by a simple indistinguishability argument, whereas BB, and therefore IC, remains solvable.

This does not come without challenges. Deterministic agreement protocols necessarily incur linear latency [41], even under crash failures only [5] (so regardless of cryptographic assumptions). Randomized Monte Carlo protocols, that is, protocols with a fixed number of rounds before halting, face another barrier: by the lower bound of [24], any agreement protocol running for $r$ rounds still has failure probability $O(cr)^{-r}$ for some constant $c$. This is true both in honest majority [47, 58, 59] and in dishonest majority [14, 34, 84]. Hence, one cannot obtain overwhelming success probability together with constant latency in the Monte Carlo setting. This still leaves open the possibility of Las Vegas protocols, with non-deterministic latency but constant expected latency.

***Efficient agreement within honest-majority.*** Constant-round Las Vegas protocols for SBA and BB have been known for almost four decades, starting from the information-theoretic work of [42]. The first protocol achieving simultaneously $n/2$-resiliency, constant expected latency, and quadratic communication is due to [66]; its rather large expected round complexity was later improved to 10 in [3] (under static corruption). These protocols follow the classical blueprint of alternating adopt-commit primitives [49], also known as graded consensus or gradecast, with random leader election. The adopt-commit part ensures safety, while termination occurs once processes converge on a common honest leader. When the cost of leader election is amortized over $n$ parallel BB instances, one obtains IC with constant expected latency and cubic expected communication, either with information-theoretic security and $n/3$-resiliency [10], or with computational security and $n/2$-resiliency [82]. Dragon [43] reduces this communication to $\Omega(n^{2.5})$ by means of a committee-based simulation paradigm, where the $n$ parties are partitioned into $\sqrt{n}$ committees of size $\sqrt{n}$. Ada-DARE [25] is the first protocol achieving IC with quadratic communication and linear resiliency $t$, in fact in strict honest majority for every $n > 2t$, but at the price of linear latency. We will return to Ada-DARE in the technical overview, since it is our starting point. In short, in honest majority, no known IC protocol achieves both constant expected latency, as in [82], and quadratic communication, as in [25].

***Efficient agreement within dishonest-majority.*** With a dishonest majority, the picture is even more limited. First, it has been shown [51] that even randomized protocols require $\Omega\big(2n/(n-t)\big)$ rounds. Hence expected constant-round protocols are impossible when the number of honest parties is sublinear in $n$. In other words, any constant-expected-latency protocol in dishonest majority must necessarily assume $t \leq (1-\varepsilon)n$ for some constant $\varepsilon > 0$. Within this necessary regime, the first BB protocols with sublinear latency are due to [51], later improved by [48], but only in the narrow setting $t \leq n/2 + o(n)$. The first sublinear result beyond this restricted setting appears in PKC'20 with [21]; following the same blueprint, [14] later improves the communication complexity. In TCC'20, Wan et al. [85] gave the first BB protocol with constant expected latency. Running this protocol $n$ times in parallel gives an IC protocol, but not a satisfactory one. First, since the non-termination probability has a geometric tail, the maximum of the $n$ parallel instances has $\Theta(\log n)$ expected latency. Second, this would incur a linear multiplicative communication overhead. Specific IC protocols have also been proposed [34, 84], but they still cannot achieve constant expected latency since they follow a strict Monte Carlo design, where the impossibility result of [24] holds. This raises the following natural question:

*Can one design an interactive consistency protocol among n processes, tolerating up to t Byzantine faults, with constant expected latency and either: (1) quadratic communication in the honest-majority regime $n > 2t$; or (2) resilience to dishonest majority, namely $t \leq (1-\varepsilon)n$ for some constant $\varepsilon > 0$?*

**Table 1: Honest-majority interactive consistency.**

| Protocol | Communication | Latency |
|---|---|---|
| ADA-Dare [25] | $\Theta\big(n^2(L_{in}+\kappa_a)\big)$ | $\Theta(t)$ |
| GLLY25 [57] | $\Theta\big(n^2 L_{in}\log n + n\kappa\kappa_a \log^3 n\big)$ | $\Theta(\kappa/\log\kappa)$ |
| DRAGON [43] | $\Omega\big(n^{2.5}(L_{in}+\kappa)\big)$ | $\Omega(1)$ |
| SAN25 [82] | $\Theta\big(n^2 L_{in} + n^3\kappa_a\big)$ | $\Theta(1)$ |
| **This-1**(VBA) | $\Theta\big(n^2(L_{in}+\kappa_a)\big) + Comm_{\mathrm{VBA}(\kappa)}$ | $\Theta(1) + 3Lat_{\mathrm{VBA}(\kappa)}$ |
| **This-1([3])** | $\Theta\big(n^2(L_{in}+\kappa_a)\big)$ | $\Theta(1)$ |

All protocols tolerate tight honest majority, namely any $n > 2t$. All protocols are secure against an adaptive adversary that can dynamically corrupt processes during the execution, however, our complexity metrics hold against a static adversary only, i.e. which performs corruptions upfront. We omit setup assumptions from the table: all these protocols rely on threshold-signature for compact certificates and/or common-coin machinery. For DRAGON, this is required by the constant-round seed protocol, not by the wrapper itself. Here $L_{in}$ denotes the input size, $\kappa$ the security parameter, and $\kappa_a$ the accumulator size ($\kappa_a = \kappa$ or $\kappa_a = \kappa \log n$, depending on the setup).

***Our contribution.*** This paper answers both questions positively, through a common approach. Assuming a bulletin-board public key infrastructure (bb-PKI), where processes publish their public-key material before the protocol starts, we construct two randomized Las Vegas protocols for IC with constant expected latency against a static adversary.

Both solutions are obtained from the same compiler, which relies on three efficient subroutine functionalities: (1) an Agreement primitive, either Byzantine Broadcast, or in honest majority only, Validated Byzantine Agreement, (2) leader election, which returns at each new view an unpredictable common leader, and (3) a Learn-Or-Expose primitive, which allows an honest leader to collect a provably valid vector of inputs, despite potentially unresponsive dishonest processes.

The first protocol works in the honest-majority regime $n > 2t$ and achieves communication complexity $O\big(n^2(L_{\mathrm{in}}+\kappa)\big)$. This is asymptotically optimal for any $L_{\mathrm{in}} \in \Omega(\kappa)$, since every process must receive the inputs of all honest processes. See Table 1 for a detailed comparison with prior work in honest majority.

The second protocol works in the dishonest-majority, i.e., whenever $t \leq (1-\varepsilon)n$ for some constant $\varepsilon \in \Omega(1)$. For constant round protocols, this condition is motivated by the impossibility result of [51]. Moreover, its communication is a factor $n$ smaller than what one would obtain by running $n$ parallel instances of the state-of-the-art Las Vegas BB protocol [85]. See Table 2 for a detailed comparison with prior work in dishonest majority.

***Cryptographic and setup assumptions.*** The first protocol assumes: (i) a setup for a $(t+1)$-threshold signature scheme (TSS) with signature uniqueness [12]; (ii) an Erasure Coding Proof (ECP) system [7, 8]; and (iii) the random oracle model. The TSS is then used to realize: (iv) the standard 1-round all-to-all common coin, and therefore leader election, in the random oracle model [19];

**Table 2: Dishonest-majority interactive consistency.**

| Protocol | Communication | Latency |
|---|---|---|
| PBC$^{\text{static}}$ [34] | $\Theta\big((n^2\kappa + n\kappa^2)(L_{\text{in}} + \kappa) + \kappa^4\big)$ | $\Theta(\kappa)$ |
| BulletinPBC [84] | $\tilde{\Theta}(n^3\kappa^2 L_{in})$ | $\Theta(n \log n)$ |
| TrustedPBC [84] | $\tilde{\Theta}(n^2\kappa^4 L_{in})$ | $\Theta(\kappa \log n)$ |
| $n\times$ [14] | $\Theta\big(n^2(L_{in}\kappa^2 + \kappa^4)\big)$ | $\Theta(\kappa \log n)$ |
| $n\times$ [85] | $\Theta(n^3 L_{in} + n^5 + n^4\kappa)$ | $\Theta(\log n)$ |
| **This-2(BB)** | $\Theta(n^4 + n^2 L_{in} + n^3\kappa + Comm_{\text{BB}(L_{dm}^{vv})})$ | $O(Lat_{\text{BB}(L_{dm}^{vv})})$ |
| **This-2([85])** | $\Theta(n^3(L_{in} + \kappa) + n^4)$ | $\Theta(1)$ |

All protocols assume at least a bulletin-board PKI. Except for BulletinPBC [84], they additionally rely on a self-sortition or eligibility mechanism, often idealized as $\mathcal{F}_{\text{mine}}$. We compare only static-adversary complexity bounds. The bounds for [85] and [14] are obtained by the standard reduction from interactive consistency to $n$ parallel instances of Byzantine Broadcast, without optimization for long inputs. We were unable to verify the correctness of [34, Lemma 1], we thus state the result without the extension-wrapper. Similarly, we do not use the extension protocol of [7, Theorem 3]. We discuss the corresponding issues in Appendix D. Finally, $L_{dm}^{vv} = n^2 + n(L_{in} + \kappa)$ denotes the size of the verifiable vector used in our Learn-or-Expose primitive.

and (v) the Learn-or-Expose primitive, where the corresponding proof-of-misbehavior (PoM) against a set of processes $G$ is simply a $(t+1)$-threshold signature on an accusation message against each process in $G$. The underlying seed agreement protocol [2] we use in our instantiation in honest majority require any additional assumptions.

Leader election cannot be instantiated in dishonest majority [28]. We therefore assume a common random string revealed only after the adversarial corruption. This is standard in secure distributed computing and can be realized in practice from an external entropy source, for example the hash of a future Bitcoin block. Any bias in this randomness affects only the expected latency, not correctness. In dishonest majority, our Learn-or-Expose primitive relies on standard digital signatures; to reduce the size of the corresponding proofs of misbehavior, one may alternatively use aggregate signatures [17] or multi-signatures [78]. Finally, the Byzantine Broadcast protocol of [85], which underlies our dishonest-majority instantiation, achieves constant expected latency against a static adversary without further assumptions.[3]

***Roadmap.*** Section 2 gives the main ideas behind the two constructions. Section 3 fixes the model, setup assumptions, complexity measures, and idealized subroutine interfaces. Section 4 presents the generic compiler from leader election, Learn-or-Expose, and Byzantine Broadcast to Interactive Consistency, and proves its correctness and complexity bounds. Section 5 instantiates the compiler in honest majority and dishonest majority. Section 6 concludes.

The appendices contain the supporting material. Appendix A spells out the cryptographic details used by the instantiations. Appendix B gives the Disperse–Agree–Retrieve reduction to one short-input VBA instance used in the honest-majority extension. Appendix C expands the comparison with prior work. Appendix D discusses why we do not use recent constant-overhead long-input extensions as black boxes. Appendix F contains the detailed honest-majority instantiation, and Appendix G collects omitted proofs.

[3]To retain constant expected latency against an adaptive adversary, [85] relies on a self-sortition functionality $\mathcal{F}_{\text{mine}}$. This can be instantiated from Verifiable Random Functions and the bulletin-board PKI in the random-oracle model [77], or without ROM under a stronger setup [2].

## 2 Technical Overview

***ADA-DARE.*** Our starting point is ADA-DARE [25]. This protocol already achieves IC in the honest majority regime with the target quadratic communication complexity $O(n^2(L_{\text{in}} + \kappa))$, but it incurs linear latency. As its name suggests, it follows the Disperse–Agree–Retrieve (DARE) paradigm. Processes first disperse candidate values and must acquire them together with a proof of dispersal (PoD). This PoD certifies that the digest corresponds to a pre-image that is both valid, according to the validity condition of the target agreement task, and retrievable. In other words, the PoD consists of a proof of validity (PoV) and a proof of retrievability (PoR).

The Agree phase is connected to the Disperse and Retrieve phases through a *seed* protocol, namely a validated Byzantine Agreement protocol (VBA) that only guarantees external validity. The validation predicate simply checks the correctness of the PoD attached to a candidate digest. Once processes agree on a digest that has been provably dispersed, they enter the Retrieve phase. This phase can be realized through a standard double-echo technique on top of an ECP system [8], in 2 rounds and with communication $O(nL_o + \kappa n^2)$, where $L_o$ is the output size of the underlying agreement task. For IC, one has $L_o = nL_{\text{in}}$, whereas for BB, VBA, or SBA one has $L_o = L_{\text{in}}$. In [25], the seed protocol is instantiated with the adaptive-communication SVBA of [33]. For our purposes, this seed can instead be replaced by [3], yielding a constant-round Agree phase with quadratic communication.

The last ingredient of ADA-DARE is its Disperse phase, implemented by ADA-DISP. In honest majority, proving retrievability is typically easy. It suffices to send the value, wait for $(n-t) \geq t+1$ signed acknowledgements, and aggregate them into a $(t+1)$-threshold signature. This threshold signature proves that at least one honest process has observed the pre-image of the corresponding digest. This is enough to later run the quadratic double-echo Retrieve protocol on top of ECP.

The real difficulty lies in proving vector-validity for IC. A vector alone is not self-certifying. If it is assembled by a dishonest process, the inputs of honest processes may simply be modified. One may first think of attaching an auxiliary material $aux^{\in}$ such that, whenever $\text{vec}[i] = v_i$, the value $aux^{\in}[i]$ contains a digital signature on $v_i$ verifiable under $p_i$'s public key. However, this is not enough. A malicious node may simply discard the input of an honest process and pretend that this process was silent. Since Byzantine processes may indeed refuse to provide their signed inputs, the vector must also be backed by a second auxiliary material $aux^{\notin}$ proving misbehavior of the processes in the set $G = \{p_j \mid \text{vec}[j] = \perp\}$. Such a proof of misbehavior (PoM) may, for instance, consist of a $(t+1)$-threshold signature on an accusation message against each member of $G$. This is precisely the approach used in ADA-DARE, and it is also the one we adopt in our honest-majority protocol.

***Naive cubic constant-round IC within honest-majority.*** After (1) an initial quadratic all-to-all input dissemination round, where every process broadcasts its signed input, one could imagine the following direct approach. In (2) a second round, all processes broadcast: (a) a partial accusation signature against every process from which they did not receive a signed input, and (b) the signed inputs they did receive in the first round. Then, in (3) a third phase of three rounds, every process $p_i$ would: (a) broadcast a verifiable

vector $(\mathrm{vec}_i, aux_i^{\in}, aux_i^{\notin})$; (b) collect acknowledgements; and (c) broadcast the digest together with the corresponding PoD. The problem is that this incurs cubic communication $O\big(n^3(L_{\mathsf{in}} + \kappa)\big)$.

***Quadratic constant-round IC within honest-majority.*** The natural way to recover quadratic communication is to turn this second and third phase into a leader-based procedure. At each view, a fresh randomly elected leader drives the construction. It waits for accusations or forwarded signed inputs from the other replicas, assembles the corresponding verifiable vector, and then executes the three-round acknowledgement phase needed to build a PoD. Agreement is still not guaranteed at that point, since a malicious leader may send the resulting PoD only to a strict subset of the honest processes. Fortunately, there is no need to achieve agreement at this level. Processes can simply echo the first valid PoD they observe. This remains quadratic and yields acquisition-synchronicity: if one honest process acquires a valid PoD, then every honest process observes a valid PoD by the next round. At that point, all of them can invoke the underlying VBA, for example [3], while doubling the round duration in order to apply the standard simulation technique of [68].

Once a process obtains a value from the VBA subroutine, which happens after $2 \cdot 10 = 20$ expected rounds with [2], following $5 \cdot 2 = 10$ expected rounds to encounter an honest leader, omitting pipelining optimizations, it can safely stop participating in the Disperse phase without fearing future accusations for omission. It can then enter the asynchronous ECP-based double-echo Retrieve phase and finally decide a verifiable vector.

This already suffices to obtain the first constant-round IC protocol with quadratic communication $O\big(n^2(L_{\mathsf{in}} + \kappa)\big)$ and linear resilience $t \in \Theta(n)$, in fact under strict honest majority, that is, for every $n > 2t$. In dishonest majority, however, the situation is more delicate, and we now elaborate on it.

***Challenges within dishonest majority.*** Because of Cleve's impossibility result, we directly assume a random leader-election oracle. In practice, this can be realized by publishing a common random string after the adversary fixes its corruption set. Even with such an oracle, two difficulties remain: one concerns proofs of misbehavior, and the other concerns the seed agreement.

The first difficulty is that when $n \leq 2t$, one can no longer expect the $n - t$ honest processes to generate a $(t + 1)$-threshold signature on accusation messages. More fundamentally, one cannot hope for an externally verifiable PoM against a silent process: a Byzantine majority could always fabricate such a PoM against an honest process. Therefore, if a leader receives neither the signed input of some suspected process $p_s$, nor $t + 1 > (n - t)$ signed accusations against it, it is no longer clear how to construct a provably valid vector. Indeed, $p_s$ may well be honest and the leader malicious. This calls for a more refined primitive, called *Learn-or-Expose* in [25]. Its purpose is to let an honest leader handle non-cooperative processes despite dishonest majority: either it learns their signed input, which is then added to $aux^{\in}$, or it exposes them, which yields material to be added to $aux^{\notin}$.

The second difficulty is that, even assuming such a constant-round Learn-or-Expose primitive, the only known constant-round agreement protocol in dishonest majority is Byzantine Broadcast [85], which provides source-validity, not external validity. Thus, one cannot simply replace the seed VBA by a closed-box BB. Doing so would require the honest minority to emulate the honest source, in a way that remains robust against the dishonest majority, which is precisely the point at issue.

A natural reaction is therefore to replace the mechanism that triggers the seed VBA after the successful three-round acknowledgement phase by a Byzantine Broadcast invocation. However, constant-round BB does not provide simultaneous termination: honest processes may decide one round apart, and therefore start the next view with a one-round offset. The blow-up-free composition of [68] relies on Byzantine-agreement-style synchronization barriers, available in their honest-majority setting but not in dishonest majority, even with authentication [83]. Falling back to the naive round-expansion repair incurs a factor-2 slowdown repeatedly.

This would be still manageable if applied to a single seed agreement instance. Here, however, the number of views until the first honest leader is itself random. If $X$ denotes the number of iterations until the first honest leader, then $X$ is geometric with success probability $p = \Theta(\varepsilon)$. Under the naive doubling approach, the total latency grows on the order of $2^X$, and $\mathbb{E}[2^X] < \infty \iff p > 1/2$. Hence, in the regime of interest $t \leq (1-\varepsilon)n$ with a small constant $\varepsilon$, this naive exponential strategy yields infinite expected latency. In particular, an expected number of iterations of order $O(1/\varepsilon)$ does not imply constant expected total latency when each failed iteration incurs exponential backoff.

We address these two difficulties in the next two paragraphs.

***Constant-round Learn-or-Expose within dishonest majority.*** To address the first issue, the key tool is the notion of a *polarizer*, introduced in [69] and refined in [40]. A polarizer is a weaker, context-dependent analogue of a certificate. Its validity is not absolute, but *parametrized by the identity of the verifier*. Informally, suppose that the set of processes is partitioned into two parts $A$ and $C$, and that every process in $A$ accuses every process in $C$. Then any process in $A$ may interpret this family of accusations as valid evidence against any process in $C$. This evidence is not externally verifiable, since an outside observer does not know which side contains the honest processes. Nevertheless, it is still *transferable* inside the protocol: if one honest process accepts such a polarizer against $p_j$, then every honest process accepts it as well, since honest processes never accuse one another.

At a high level, LearnOrExpose works as follows. Suppose the leader failed to obtain the signed input of some target process $p^*$. The leader first accuses $p^*$. Then every process is expected either to forward a valid signed message from $p^*$, or to accuse $p^*$ as well. If all such responses arrive consistently, the leader either learns the missing signed input, or gathers enough accusations to expose $p^*$. The difficulty is that some intermediary process may itself refuse to respond. In that case, the protocol proceeds recursively: the non-responsive intermediary is accused in turn, and the same reasoning is applied again. This already captures the main idea of LearnOrExpose, but in its direct form it may require a linear number of rounds.

To obtain a constant-round protocol, it is useful to view the collected accusations as defining a graph, where an edge indicates that the two corresponding processes are still mutually compatible. In

this graph, if the leader still has neither learned nor exposed $p^*$ after one round, then $p^*$ must already be at distance at least two from the leader. If the same happens again in the next round, the distance increases again. Continuing in this way, after round $r$, if the leader still has neither learned nor exposed $p^*$, then the distance between the leader and $p^*$ is at least $r + 1$. A crucial observation is that all honest processes always remain in the same clique, since honest processes never accuse one another. Therefore, any edge that does not belong to a clique of the corresponding size cannot lie inside the honest core, and can in principle be removed for free[4]. After this pruning step, the distance-growth argument above becomes the key to the analysis: once the distance from the leader to $p^*$ exceeds the diameter of the refined graph, the process $p^*$ must be disconnected from the leader and is therefore exposed by a valid polarizer. Since in the dishonest-majority regime $t \le (1-\varepsilon)n$ the refined graph has constant diameter, after only a constant number of rounds the process $p^*$ must either be learned or become disconnected from the leader. Hence LearnOrExpose can be implemented in a deterministic constant number of rounds.

***Bypassing non-simultaneous termination of the seed-BB.*** To address the second issue, the key observation is that non-simultaneous termination is not an obstacle. Each view, whether driven by an honest leader or a dishonest one, is independent of the previous views and terminates after an expected constant number of rounds. Moreover, if processes obtain outputs from several consecutive views, they may simply retain the first verifiable vector among them.

This suggests a simple scheduling rule: instead of waiting for one BB instance to terminate before starting the next, processes launch a fresh instance every fixed number $c$ of rounds. Whenever a contiguous block of such views has terminated, and the last one of them returns a verifiable vector, processes decide that vector and abandon all remaining unfinished views. In this way, no explicit synchronization between views is needed, yet the overall protocol still achieves constant expected latency.

Some care is needed when abandoning unfinished views. We rely on a seed BB protocol, such as [85], with near-simultaneous termination: once one honest process terminates an instance, all honest processes terminate the same instance within $\Delta_{\text{sim}}^{\text{BB}}$ rounds. Hence, after deciding, a process waits $\Delta_{\text{sim}}^{\text{BB}}$ rounds before halting. At that point, IC has terminated at all honest processes. Thus halting cannot affect IC consistency or validity.

Combined with the deterministic constant-round Learn-or-Expose primitive, this suffices to obtain the first constant-round IC protocol in the dishonest-majority regime for every $t \le (1-\varepsilon)n$, for any constant $\varepsilon > 0$. As discussed earlier, such a restriction is necessary in light of the lower bound of [51].

# 3 Preliminaries

## 3.1 Model and Assumptions

***Processes and synchronous authenticated network.*** We consider a static system $\Pi = \{p_1, \ldots, p_n\}$ of $n$ processes. An adversary corrupts upfront a set $D \subseteq [n]$ of $|D| = f \le t$ processes, for some integer $t \le (1-\varepsilon)n$ and some constant $\varepsilon \in (0,1)$. The special case $\varepsilon > 1/2$ yields the honest-majority regime $n > 2t$. The corrupted processes are arbitrarily coordinated by the adversary and said *dishonest*. The remaining processes are said *honest*.

Processes communicate over a complete authenticated synchronous network: (i) the receiver of a message knows the sender's identity, (ii) every pair of processes can communicate directly, and (iii) if an honest process sends a message to an honest process, then the message is received within at most $\Delta$ time units, where $\Delta$ is known to all honest processes. We normalize time so that $\Delta = 1$. Equivalently, we may reason in rounds: any message sent by an honest process at the beginning of a round is received by every honest recipient by the end of the same round.

***Cryptographic and setup assumptions.*** We write $\kappa$ for the security parameter. Honest processes and the adversary are probabilistic polynomial-time in $\kappa$. We rely on standard cryptographic primitives, secure except with negligible probability in $\kappa$, and on a collision-resistant hash function [60, §6.2.5] $\mathsf{hash} : \{0,1\}^* \to \{0,1\}^{O(\kappa)}$.

We assume a bulletin-board public-key infrastructure (bb-PKI): before receiving inputs and starting the protocol, processes may publish public cryptographic material on a public append-only bulletin board accessible to all. In practice, one may think of a blockchain and require all public material to be posted before a fixed block height. In UC terminology, this is the certification-authority functionality $\mathcal{F}_{\text{CA}}$ of [20]. At a minimum, each process publishes the public key of an EUF-CMA secure digital-signature scheme [60, §6.1.2]: (Gen, Sign, Ver). The key of process $p_i$ is denoted $\mathsf{pk}_i$, and signatures are verified as $\mathsf{Ver}_{\mathsf{pk}_i}(m,\sigma) \in \{0,1\}$. This realizes the usual signature functionality $\mathcal{F}_{\text{sign}}$; combined with $\mathcal{F}_{\text{CA}}$, it realizes the certification functionality $\mathcal{F}_{\text{cert}}$, which binds signed messages to process identities. Processes whose public material is missing or invalid on the bulletin board are ignored.

***Honest-majority.*** In the honest-majority regime $t < n/2$, we additionally assume a setup for a unique $(t+1)$-threshold-signature scheme. Such a setup can be obtained from the distributed key-generation protocol of [12], with constant latency and quadratic communication. The scheme satisfies the standard robustness and unforgeability properties [38, §4.3]. Together with the random-oracle model, threshold-signature uniqueness is used to implement common coins and leader election, e.g., through the one-round all-to-all coin of [19]. We defer the details of the honest-majority threshold-cryptographic setup to Appendix A.2.

***Dishonest-majority.*** In the dishonest-majority regime, where only $t \le (1-\varepsilon)n$ for a constant $\varepsilon \le 1/2$, we do not assume a threshold-signature setup: indeed, $n - t$ may be smaller than $t + 1$, so a $(t+1)$-threshold signature can no longer be produced by honest processes alone. Instead, we assume an ideal random leader-election functionality. Concretely, this can be instantiated from public randomness revealed only after the static corruption takes place, for example by taking the hash of a future block of the blockchain already used to implement the bb-PKI.

For compact polarizers and related certificates, we use aggregate signatures [17] or multi-signatures [78]. Multi-signatures provide more efficient verification, at the cost of publishing a proof

[4] Deciding whether an edge belongs to a clique of a given size is NP-hard in general. Fortunately, [40, 85] show how to avoid this obstacle by replacing the ideal clique-based pruning with an efficient graph-construction rule that preserves the same distance-growth argument.

of possession together with each public key to prevent rogue-key attacks; recent constructions make this proof very cheap [78]. In both cases, we assume the standard robustness and unforgeability properties [78, §III]. We defer the details of the dishonest-majority cryptographic setup to Appendix A.1.

***complexity.*** $\mathsf{Execs}_\Pi$ denotes the set of executions of protocol $\Pi$. A *world* for $\Pi$ is a pair $\mathcal{W} = (\mathcal{A}, c_I)$, where $\mathcal{A}$ is an adversary for $\Pi$ and $c_I$ is an input configuration for $\Pi$, i.e. a partial mapping from honest processes to their inputs. Each world $\mathcal{W}$ induces a probability distribution $\Pr_{\mathcal{W}}$ over $\mathsf{Execs}_\Pi$. Accordingly, for any random variable $X$ over $\mathsf{Execs}_\Pi$ (e.g., latency or communication), we define $\mathbb{E}_{\mathcal{W}}[X] := \sum_{e \in \mathsf{Execs}_\Pi} X(e) \Pr_{\mathcal{W}}(e)$. For any event $V \subseteq \mathsf{Execs}_\Pi$, we write $\Pr(V) := \sup_{\mathcal{W}} \Pr_{\mathcal{W}}(V)$, where the supremum ranges over all admissible worlds for $\Pi$. Similarly, for any random variable $X$ over $\mathsf{Execs}_\Pi$, we write $\mathbb{E}[X] := \sup_{\mathcal{W}} \mathbb{E}_{\mathcal{W}}[X]$. We only consider protocols for which processes halt after at most $R^* \in poly(\kappa)$ rounds. Accordingly, for an execution $e$, let $\mathsf{Lat}_\Pi(e)$ be the smallest round $r$ such that all honest processes have returned and halted by round $r$, if such an $r$ exists; and $R^*$ otherwise. The latency complexity $\mathsf{lc}_\Pi$ of a protocol $\Pi$ is then the (worst-case) expected value of its corresponding latency random variable: $\mathbb{E}(\mathsf{Lat}_\Pi)$. Similarly, for an execution $e$, let $\mathsf{Com}_\Pi(e)$ denote the total number of bits sent by honest processes during $e$. The communication complexity $\mathsf{cc}_\Pi$ of a protocol $\Pi$ is then the (worst-case) expected value of its corresponding communication random variable: $\mathbb{E}(\mathsf{Com}_\Pi)$. When clear from context, we slightly abuse notation by extending random variables associated with a protocol $\Pi$ to executions of higher-level protocols invoking $\Pi$ as a subroutine.

***Presentation level and composition.*** For simplicity, and as is standard in secure distributed computing, we present the auxiliary cryptographic and distributed primitives below through their ideal property-based behavior rather than through a full simulation-based formalization. We assume that these primitives can be composed with each other in the natural way. Concretely, we only consider executions in which the assumed primitives are not broken or forged; the complementary bad executions occur with negligible probability in the security parameter $\kappa$, and this negligible error remains negligible under polynomial composition. This abstraction is well motivated by standard realizability results. For instance, EUF-CMA signatures can be used to UC-realize the corresponding signature functionality, which, combined with a bulletin-board PKI (captured through the certification-authority functionality $\mathcal{F}_{\mathrm{CA}}$ of [20]), yields the usual certification functionality ($\mathcal{F}_{\mathrm{cert}}$ in [20]) binding process identities to public keys and signatures. Likewise, the agreement-type subroutines used in this paper admit direct straight-line simulations between their real and ideal descriptions, so the properties relevant here, including safety and latency bounds such as expected running time, are preserved under composition. We therefore use the idealized interfaces below as a concise and faithful presentation layer.

## 3.2 Principal subroutines

We adopt a modular presentation in which each building block is specified as a one-shot distributed task [76]. In such a task $T$, every process may submit one input value from some input domain $\mathcal{V}_I^T$, eventually obtain one output value from some output domain $\mathcal{V}_O^T$, and, trigger a special halt request. Each module is then defined by the properties it guarantees. First, it satisfies *safety* properties, which specify the admissible output configurations as a function of the input configuration; here, an input (resp., output) configuration is simply a partial mapping from honest processes to their submitted inputs (resp., produced outputs). Second, it may satisfy *unforgeability* properties, expressing that a polynomially bounded adversary cannot compute some forbidden object, typically a proof falsely certifying that an honest process exhibited a certain behavior. Third, it satisfies *liveness* properties, requiring honest processes to eventually produce an output, possibly within a fixed time, or with some simultaneity guarantee.

Unless otherwise explicitly stated, these guarantees are only claimed under two execution preconditions. First, except if stated otherwise, honest processes must submit their inputs simultaneously, or at least near-simultaneously, e.g., within one round of each other. We note that simultaneous submission can be simulated from near-simultaneous submission via the standard round-synchronizing simulation of [68][5]. Second, no honest process may halt before every other honest process has obtained its output. In the protocols of this paper, these preconditions are either enforced by construction, or their violation is confined to a subroutine in a way that is harmless to the upper-level protocol. Likewise, all subroutines except agreement tasks are designed to terminate simultaneously after a fixed number of rounds $\Delta_{lat}^T$.

For agreement tasks such as IC, BB, or VBA, when one targets constant expected latency, neither Monte-Carlo-style fixed constant deterministic termination [24] nor exact simultaneous termination [68] is achievable. Accordingly, we do not specify any fixed latency bound $\Delta_{lat}^T$ . Instead, termination is required only to satisfy a weaker near-simultaneity condition: all honest processes terminate within a known constant number $\Delta_{sim}^T$ of rounds of one another. After this delay, the special halt request may safely be triggered for essentially all still-pending subroutines.

In fact, for the two agreement subroutines relevant to this paper, we have the stronger bound $\Delta_{sim}^T = 1$: this follows from [3, Section 3.1] in the honest-majority regime, and from [85, Fact 5.13] in the dishonest-majority regime. This property will in turn carry over to our two IC protocols. This weaker notion of simultaneity is sufficient for our composition.

Unless stated otherwise, every module below is interpreted under the generic execution preconditions introduced above: honest processes invoke the input interface simultaneously or near-simultaneously, and no honest process halts before all honest processes have obtained their output. We therefore omit the special halt request from the module interfaces, and only mention deviations from this convention when needed.

Agreement tasks (see Module 1) and leader election (see Module 2) are standard objects, so we only recall their abstract interfaces. The only non-standard primitive is Learn-Or-Expose, whose role is to let a leader obtain a verifiable vector suitable for a subsequent Byzantine Broadcast/Agreement invocation.

[5]The original simulation of [68]is stated for information-theoretic protocols and gives $\Delta_{\mathrm{new}} = \Delta_{\mathrm{shift}} + \Delta$, where $\Delta_{\mathrm{shift}}$ is the maximum skew between honest submissions and $\Delta = 1$ is the (normalized) delivery bound. For cryptographic protocols, the analogous simulation is made explicit in [26, §A], with $\Delta_{\mathrm{new}} = 2\Delta_{\mathrm{shift}} + \Delta$.

**Module 1** Generic Agreement Task

**Parameters:**
- A validity property [27] $\mathrm{val} : \mathcal{I}(\mathcal{V}_I) \to 2^{\mathcal{V}_O} \setminus \{\emptyset\}$, where $\mathcal{I}(\mathcal{V}_I)$ is the set of input configurations, where each $c_I \in \mathcal{I}(\mathcal{V}_I)$ is a partial map from honest processes to submitted values in $\mathcal{V}_I$.
- A near-simultaneity parameter $\Delta_{\mathsf{sim}}$.

**Events:**
- *request* $\mathsf{submit}(v \in \mathcal{V}_I)$.
- *indication* $\mathsf{deliver}(v' \in \mathcal{V}_O)$.

**Properties:**
- *Near-simultaneous termination:* all honest processes eventually deliver within $\Delta_{\mathsf{sim}}$ time units of each other.
- *Consistency:* no two honest processes deliver different values.
- *Validity:* if $c_I$ is the input configuration induced by the honest submissions, then every value delivered by an honest process belongs to $\mathrm{val}(c_I)$.
  The validity property can be instantiated as follows:
  - **Vector-Validity (IC).** Here $\mathcal{V}_O = \mathcal{V}_I^n$, and

$$\mathrm{val}_{IC}(c_I) := \left\{ \vec{v} \in \mathcal{V}_I^n \;\middle|\; \forall p_j \in \mathrm{dom}(c_I),\ \vec{v}[j] = c_I[p_j] \right\}.$$

  - **Source-Validity (BB).** Here $\mathcal{V}_O = \mathcal{V}_I$, a source process $p_s$ is fixed, and

$$\mathrm{val}_{BB}^{p_s}(c_I) := \begin{cases} \{c_I[p_s]\}, & \text{if } p_s \in \mathrm{dom}(c_I), \\ \mathcal{V}_O, & \text{otherwise.} \end{cases}$$

  - **External-Validity (VBA).** Here $\mathcal{V}_I = \mathcal{V}_O = \{\, v \in \{0,1\}^* \mid Q(v) \,\}$, where $Q : \{0,1\}^* \to \{\mathrm{true}, \mathrm{false}\}$ is a globally verifiable predicate, and

$$\mathrm{val}_{EV}^{Q}(c_I) := \begin{cases} \{c_I[p_1]\}, & \text{if } \forall i \in [n],\ p_i \in \mathrm{dom}(c_I) \ \wedge\ c_I[p_i] = c_I[p_1], \\ \mathcal{V}_O, & \text{otherwise.} \end{cases}$$

**Module 2** Leader Election Oracle

**Events:**
- *request* $\mathsf{inquire}(v \in \mathsf{View})$.
- *indication* $\mathsf{inform}(v, \ell_v \in [n])$.

**Properties:**
- *Termination:* if all honest processes invoke $\mathsf{inquire}(v)$ simultaneously, then all honest processes receive $\mathsf{inform}(v, \ell_v)$ simultaneously: after one round in the honest-majority regime, and immediately in the dishonest-majority regime.
- *Agreement:* for every view $v$, no two honest processes receive $\mathsf{inform}(v, \ell_v)$ and $\mathsf{inform}(v, \ell'_v)$ with $\ell_v \neq \ell'_v$.
- *Fairness:* for every view $v$, the probability that the returned (common) leader $\ell_v$ is honest is at least $\frac{n-t}{n}$.

**Module 3** Learn-Or-Expose

**Parameters:**
- A distinguished leader $p_\ell$.
- A proof domain $\Pi_{\mathsf{LoE}}$.
- $\forall i \in [n]$, local predicates $\mathsf{VerifyVec}_i^{in}, \mathsf{VerifyVec}_i^{out} : (\mathcal{V}_I \cup \{\perp\})^n \times \Pi_{\mathsf{LoE}} \to \{\mathrm{true}, \mathrm{false}\}$.

**Events:**
- *request* $\mathsf{start}(\vec{u} \in (\mathcal{V}_I \cup \{\perp\})^n, \pi^{in} \in \Pi_{\mathsf{LoE}})$.
- *indication* $\mathsf{return}(\vec{v} \in (\mathcal{V}_I \cup \{\perp\})^n, \pi^{out} \in \Pi_{\mathsf{LoE}})$.

**Assumed Preconditions:**
- *Honest Input Agreement:* for every honest process $p_j$, we have $\vec{u}_{p_j}[j] \neq \perp$; let us call $\vec{u}_{p_j}[j]$ the value submitted by $p_j$. Moreover, for every three honest processes $p_i, p_j, p_k$, we have $\vec{u}_{p_i}[k] = \vec{u}_{p_j}[k]$, and for every honest process $p_i$, $\mathsf{VerifyVec}_i^{in}(\vec{u}_{p_i}, \pi_i^{in}) = \mathrm{true}$.
- *Input Soundness:* no PPT adversary can produce $(\vec{v}, \pi^{in}, p_i)$, with $p_i$ honest, such that $\mathsf{VerifyVec}_i^{in}(\vec{v}, \pi^{in}) = \mathrm{true}$, unless, for every honest process $p_j$, the entry $\vec{v}[j]$ is exactly the input submitted by $p_j$ or $\perp$.

**Properties:**
- *Termination:* all honest processes simultaneously terminate after fixed known latency.
- *Progress:* if $p_\ell$ is honest, then it eventually returns some $(\vec{v}, \pi^{out})$ such that $\mathsf{VerifyVec}_\ell^{out}(\vec{v}, \pi^{out}) = \mathrm{true}$.
- *Transferability:* no PPT adversary can produce $(\vec{v}, \pi^{out}, p_i, p_j)$, with $p_i, p_j$ honest, such that $\mathsf{VerifyVec}_i^{out}(\vec{v}, \pi^{out}) \neq \mathsf{VerifyVec}_j^{out}(\vec{v}, \pi^{out})$.
- *Output Soundness:* no PPT adversary can produce $(\vec{v}, \pi^{out}, p_i)$, with $p_i$ honest, such that $\mathsf{VerifyVec}_i^{out}(\vec{v}, \pi^{out}) = \mathrm{true}$, unless, for every honest process $p_j$, the entry $\vec{v}[j]$ is exactly the input submitted by $p_j$.

The soundness properties are the validity part of this abstraction. Input soundness prevents the protocol from starting from a vector that already forges an honest input. Output soundness ensures that any output vector accepted by an honest process contains the true input of every honest process, and therefore cannot violate IC vector-validity. Transferability is the agreement part: although verification may be local, no adversary can make two honest processes disagree on whether the same output pair $(\vec{v}, \pi^{out})$ is valid. Hence, once Byzantine Broadcast delivers the same pair to all honest processes, they either all accept it or all reject it, which prevents a consistency violation at the compiler level.

## 4 Generic Compiler

In this section, we present a generic transformation from Leader Election, Learn-or-Expose, and Byzantine Broadcast to Interactive Consistency. The transformation is captured by Algorithm 1. The parties first execute an input-dissemination step, in which each party sends its signed input to everyone. They then proceed in pipelined views: in each view the parties elect a leader, the leader completes the missing coordinates of the candidate vector using Learn-or-Expose, and then Byzantine-broadcasts the resulting verifiable vector. The theorem below states that, under the corresponding guarantees of the three underlying primitives, Algorithm 1 solves Interactive Consistency and inherits the resulting latency and communication bounds.

For any protocol $\Pi$, recall that $Lat_\Pi, Comm_\Pi : \mathsf{Execs}_\Pi \to \mathbb{N} \cup \{0\}$ are the random variables corresponding to the latency and communication complexity of $\Pi$, respectively. We write $\mathsf{lc}_\Pi := \mathbb{E}[Lat_\Pi]$ and $\mathsf{cc}_\Pi := \mathbb{E}[Comm_\Pi]$ for the corresponding expectations.

Assume the following primitives: a Leader Election oracle $\mathsf{LE}$, a LearnOrExpose protocol $\mathsf{LoE}$ with proof size $\lambda_{\mathsf{LoE}}$ and input predicate $\mathsf{VerifyVec}^{in}$ defined by $\mathsf{VerifyVec}_i^{in}(\vec{v}, \pi) = \mathrm{true}$ if for every $j \in [n]$ with $\vec{v}[j] \neq \perp$ one has $\mathsf{ver}_{\mathsf{pk}_j}(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \sigma[j]) = \mathrm{true}$, and a Byzantine Broadcast protocol $\mathsf{BB}$ for inputs of bit-length $n(L_{\mathsf{in}} + \kappa) + \lambda_{\mathsf{LoE}}$, with near-simultaneous termination within $\Delta_{sim}$ rounds, and with per-message complexity $\mathsf{pmc}_{\mathsf{BB}}$, that is, in any round the communication sent from one party to another does not exceed $\mathsf{pmc}_{\mathsf{BB}}$.

Then the following theorem holds.

Theorem 4.1 (Generic IC compiler). *Fix $\varepsilon \in (0,1)$. Algorithm 1 with subroutines $\mathsf{LE}$, $\mathsf{LoE}$, and $\mathsf{BB}$, and for any constant $c$, solves Interactive Consistency against any static adversary corrupting at most $t \leq (1-\varepsilon)n$ parties. Moreover,*

$$\mathsf{lc}_{IC} \leq 1 + \frac{c}{\varepsilon} + \Delta_{sim} + \mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \mathbb{E}(\max_{u \in [X \sim \mathrm{Geom}(\varepsilon)]} Lat_{\mathsf{BB}_u}) = O\big(\mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \mathsf{lc}_{\mathsf{BB}}\big).$$

*and*

$$\mathsf{cc}_{IC} \leq \left\lceil \frac{\mathsf{lc}_{IC}}{c} \right\rceil \cdot \big(\mathsf{cc}_{\mathsf{LE}} + \mathsf{cc}_{\mathsf{LoE}} + \mathsf{cc}_{\mathsf{BB}}\big) + n^2(L_{\mathsf{in}} + \kappa + \Delta_{sim}\mathsf{pmc}_{\mathsf{BB}}).$$

**Algorithm 1** InteractiveConsistency at process $P_i$

**Black-box subprotocols:**
1: LE, LoE, BB

**Parameters:**
2: $c$ ▷ a new view starts every $c$ rounds
3: $\Delta_{sim}$ ▷ termination/halting window

**Local state at process $P_i$:**
4: $doneTime \leftarrow \infty$
5: $out \leftarrow \perp$
6: **for** $j \in [n]$ **do**
7: $candVec[j] \leftarrow \perp$
8: $candSign[j] \leftarrow \perp$
9: **for** $v \in \mathbb{N}_{\geq 1}$ **do**
10: $leader[v] \leftarrow \perp$
11: $completed[v] \leftarrow$ false
12: $accepted[v] \leftarrow$ false
13: $vec[v] \leftarrow (\perp, \ldots, \perp)$

14: **upon** IC.submit($m_i$)
15: send $(\langle \text{disclose}, m_i \rangle, \text{Sign}_{P_i}(\langle \text{disclose}, m_i \rangle))$ to all parties
16: start timer

17: **upon** receiving for the 1st time a valid message $(\langle \text{disclose}, x_j \rangle, \sigma_j)$ from $P_j$
18: $candVec[j] \leftarrow x_j$
19: $candSign[j] \leftarrow \sigma_j$

20: **upon** timer reaches time $1 + (v-1) \cdot c$ for some view $v \geq 1$
21: **if** currentTime $<$ $doneTime + \Delta_{sim}$ **then**
22: request $\text{LE}_v$.inquire($v$)

23: **upon** $\text{LE}_v$informs($\ell$)
24: **if** currentTime $<$ $doneTime + \Delta_{sim}$ **then**
25: $leader[v] \leftarrow \ell$
26: request $\text{LoE}_v$.start($candVec$, $candSign$)

27: **upon** LearnOrExpose$_v$ returns $(\tilde{vec}, \pi)$
28: **if** currentTime $<$ $doneTime + \Delta_{sim}$ **then**
29: **if** $i = leader[v]$ **then**
30: $\text{BB}_v$.submit$((\tilde{vec}, \pi))$
31: **else**
32: $\text{BB}_v$.submit$(\perp)$

33: **upon** $\text{BB}_v$ delivers some value $y$
34: **if** $doneTime = \infty$ **then**
35: $completed[v] \leftarrow$ true
36: **if** $y \neq \perp$ **then**
37: parse $y$ as $(\tilde{vec}, \pi)$
38: **if** $\text{LoE.VerifyVec}_i^{out}(\tilde{vec}, \pi)$ **then**
39: $accepted[v] \leftarrow$ true
40: $vec[v] \leftarrow \tilde{vec}$
41: $u \leftarrow 1$
42: **while** $completed[u] =$ true **do**
43: **if** $accepted[u] =$ true **then**
44: **if** $doneTime = \infty$ **then**
45: $out \leftarrow vec[u]$
46: indicate IC.deliver($out$)
47: $doneTime \leftarrow$ currentTime
48: **break**
49: $u \leftarrow u + 1$

50: **upon** $doneTime \neq \infty$ and timer reaches time $doneTime + \Delta_{sim}$
51: **for** each started $\text{LoE}_v$ and $\text{BB}_v$ **do**
52: request $\text{LoE}_v$.halt() and $\text{BB}_v$.halt()
53: request IC.halt()

*Remark.*

(1) Note that the term $\mathbb{E}\big(\max_{u \in [X \sim \text{Geom}(\varepsilon)]} Lat_{\text{BB}_u}\big)$ can always be upper bounded by $\frac{1}{\varepsilon} \cdot \text{lc}_{\text{BB}}$. Moreover, if $Lat_{\text{BB}}$ is stochastically dominated by a geometric distribution, then it can be bounded by $\log(1/\varepsilon) \cdot \text{lc}_{\text{BB}}$. This is the case for the concrete instantiations that follow. More generally, we believe that the latency of most Las Vegas protocols is stochastically dominated by a geometric distribution. We keep the expectation in this form for the sake of generality.

(2) In our concrete instantiations, $\text{pmc}_{\text{BB}} \leq \text{cc}_{\text{BB}}/n^2$, and therefore the term $n^2 \cdot \Delta_{sim} \cdot \text{pmc}_{\text{BB}}$ can be replaced by $\Delta_{sim} \cdot \text{cc}_{\text{BB}}$. More generally, we believe that most constant-round protocols can be made balanced, in the sense that their per-message complexity does not exceed a $1/n^2$ fraction of the total communication complexity. Again, we write the theorem in the more explicit form for the sake of generality.

We first present two concrete instantiations of the compiler in Corollaries 4.2 and 4.3, and then prove Theorem 4.1.

Corollary 4.2 (Dishonest-majority instantiation). *For any constant $\varepsilon \in (0,1)$, one can solve Interactive Consistency in the dishonest-majority setting $t \leq (1-\varepsilon)n$ with $O(1)$ expected latency. This is achieved by instantiating Algorithm 1 with:*

*(1) the polarizer-based Learn-or-Expose primitive $\text{LoE}^{\text{dm}}$, instantiated by Algorithm 2,*
*(2) a leader-election primitive $\text{LE}_{\varepsilon}^{\text{dm}}$ that elects an honest leader with probability at least $\varepsilon$,*
*(3) the Byzantine Broadcast protocol of [85] for messages of length[6] $O(n(L_{\text{in}} + \kappa) + n^2)$.*

Corollary 4.3 (Honest-majority instantiation). *One can solve Interactive Consistency in the honest-majority setting $t < n/2$ with $O(1)$ expected latency and expected communication complexity $O(n^2(L_{in} + \kappa))$. This is achieved by instantiating Algorithm 1 with:*

*(1) a TSS-based Learn-or-Expose primitive $\text{LoE}^{\text{hm}}$, instantiated by Algorithm 9,*
*(2) a leader-election primitive $\text{LE}^{\text{hm}}$,*
*(3) the Byzantine Broadcast protocol obtained from Algorithm 8, instantiated with source-validity $\text{val}^*_{\text{BB}}$ as described in Appendix B.3 and Corollary B.5. This protocol extends the protocol of [3] with standard erasure-code-based techniques to accommodate long values[7] of size $n(L_{in} + \kappa)$.*

Proof of Theorem 4.1. We first show that the assumed input behavior of LearnOrExpose is satisfied. After the initial input-dissemination step in line 15, every honest process $p_j$ stores in $(candVec, candSign)$ a pair such that $\text{VerifyVec}_j^{in}(candVec, candSign)$ is true. Indeed, every honest process signs and broadcasts only its own input, and therefore the Honest Input Agreement property holds. Moreover, Input Soundness follows from the unforgeability of the underlying signature scheme together with the bb-PKI setup.

Now, fix an execution $\epsilon \in \text{Exec}_{IC}$. For each view $v \geq 1$, let $\ell_v$ be the leader elected in view $v$. By the agreement property of LE, all honest processes obtain the same leader $\ell_v$ in view $v$. Let $X := \min\{v \geq 1 : \ell_v \text{ is honest}\}$ be the index of the first view with an honest leader.

We say that a process $p_i$ *halts before the end of view* $v$ if it halts before $\text{BB}_v$ returns. We say that a process $p_i$ *accepts a pair* $(\tilde{vec}, \pi)$ *from an instance* $\text{BB}_v$ if the check $\text{LoE.VerifyVec}_i^{out}(\tilde{vec}, \pi)$ in line 38 passes, and is followed by the assignments $accepted[v] \leftarrow$ true in line 39 and $vec[v] \leftarrow \tilde{vec}$ in line 40.

[6] Using the extension protocol of [7], which we were not able to verify (see Appendix D), one can improve the communication complexity by running Byzantine Broadcast only on an input of size $\kappa$. This yields total communication $O(n^4 + n^3\kappa + n^2 L_{\text{in}})$, compared with the current $O(n^4 + n^3(L_{\text{in}} + \kappa))$.

[7] Due to minor technicalities, we cannot use existing extension protocols [7, 25, 71] in a closed-box manner. We thus provide a new extension protocol in the Appendix.

We will use the following claims in the proof of the theorem.

**Claim 1 (view-wise agreement on acceptance).** For every fixed view $v$, if no honest process halts before the end of view $v$, then either all honest processes accept the pair delivered by $\mathsf{BB}_v$, or none of them do. Moreover, if they accept, then they all store the same vector in $vec[v]$.

*Proof of Claim 1.* Fix a view $v$, and assume that no honest process halts before the end of view $v$. By consistency of BB, all honest processes deliver the same pair $y_v = (\tilde{vec}, \pi)$ from $\mathsf{BB}_v$. Thus every honest process applies its acceptance test to the same pair $(\tilde{vec}, \pi)$.

By transferability of LoE, for every two honest processes $p_i, p_j$, $\mathsf{LoE.VerifyVec}_i^{out}(\tilde{vec}, \pi) = \mathsf{LoE.VerifyVec}_j^{out}(\tilde{vec}, \pi)$. Thus, the check in line 38 has the same outcome at every honest process. Therefore, either all honest processes accept $(\tilde{vec}, \pi)$ from $\mathsf{BB}_v$, or none of them do. If they do, then all of them execute lines 39 and 40, and therefore store the same vector $\tilde{vec}$ in $vec[v]$. ◁

**Claim 2 (any accepted view is valid on honest coordinates).** If an honest process accepts a pair $(\tilde{vec}, \pi)$ from $\mathsf{BB}_v$, and no honest process halts before the end of view $v$, then for every honest process $p_j$ one has $\tilde{vec}[j] = m_j$.

*Proof of Claim 2.* Fix an honest process $p_i$ that accepts $(\tilde{vec}, \pi)$ from $\mathsf{BB}_v$. By definition of acceptance, the check $\mathsf{LoE.VerifyVec}_i^{out}(\tilde{vec}, \pi)$ passes. By Output Soundness of LoE, this implies that for every honest process $p_j$, the entry $\tilde{vec}[j]$ equals the input submitted by $p_j$. Hence $\tilde{vec}[j] = m_j$ for every honest process $p_j$. ◁

**Claim 3 (existence of an accepted view).** There exists a view $V \le X$ such that no honest process halts before the end of view $V$ and every honest process accepts the pair delivered by $\mathsf{BB}_V$.

*Proof of Claim 3.* We distinguish two cases.

*Case 1: no honest process halts before the end of view $X$.* Since the leader of view $X$ is honest, by the progress property of LearnOrExpose, it obtains some pair $(\tilde{vec}, \pi)$ such that $\mathsf{VerifyVec}_{\ell_X}^{out}(\tilde{vec}, \pi) = \text{true}$. Hence it submits $(\tilde{vec}, \pi)$ to $\mathsf{BB}_X$. Since the sender of $\mathsf{BB}_X$ is honest, by the validity of BB, every honest process delivers $(\tilde{vec}, \pi)$. By Claim 1, either all honest processes accept this pair or none do. Since the honest leader accepts it, all honest processes accept it. Thus, the claim holds with $V = X$.

*Case 2: some honest process halts before the end of view $X$.* Let $p_i$ be the first honest process that outputs, let $\tau$ be its output time, and let $V$ be the index of the view whose value it outputs. By definition of the algorithm, once an honest process outputs at time $\tau$, it halts only at time $\tau + \Delta_{sim}$. Hence, no honest process halts before time $\tau + \Delta_{sim}$. In particular, no honest process halts before the end of view $V$, since all honest processes finish view $V$ by time $\tau + \Delta_{sim}$. Since $p_i$ outputs $vec[V]$, it must have $accepted[V] = \text{true}$. By Claim 1, every honest process accepts the pair delivered by $\mathsf{BB}_V$.

It remains to show that $V \le X$. Suppose for contradiction that $V > X$. Since $p_i$ is the first honest process that outputs, no honest process halts before time $\tau$. Hence, in particular, no honest process halts before the end of view $X$. Therefore Case 1 applies, and every honest process accepts the pair delivered by $\mathsf{BB}_X$. Moreover, because $p_i$ outputs the value of view $V$ at time $\tau$, the scan in lines 41–49 has already reached $V$. Thus all views $1, \dots, V$ are completed at $p_i$ by time $\tau$. In particular, view $X < V$ is already completed and accepted at $p_i$. But then $p_i$ would output the value of view $X$ rather than that of view $V$, a contradiction. Hence $V \le X$. ◁

**Near-simultaneous termination.** By Claim 3, there exists a view $V \le X$ such that no honest process halts before the end of view $V$ and every honest process accepts the pair delivered by $\mathsf{BB}_V$.

Let $U^\star \le V$ be the smallest view such that some honest process accepts the pair delivered by $\mathsf{BB}_{U^\star}$. We claim that every honest process outputs the value of view $U^\star$.

First, since $U^\star \le V$ and no honest process halts before the end of view $V$, no honest process halts before the end of view $U$ either. Hence Claim 1 applies to view $U^\star$, and therefore every honest process accepts the pair delivered by $\mathsf{BB}_{U^\star}$ and stores the same value in $vec[U^\star]$.

Next, by minimality of $U^\star$, no view $u < U^\star$ is accepted by any honest process. By Claim 1, this implies that no honest process accepts the pair delivered by $\mathsf{BB}_u$ for any $u < U^\star$. Therefore, when honest processes scan completed views in increasing order, the first accepted one is exactly $U^\star$. As soon as all views $u \le U^\star$ are completed, every honest process outputs $vec[U^\star]$.

Thus, every honest process eventually outputs the value of the same view $U^\star$, and therefore all honest processes terminate within the near-simultaneity window. Hence, near-simultaneous termination is inherited from the near-simultaneity of the instances $\{\mathsf{BB}_u\}_{u \le U^\star}$.

**Consistency.** By the proof of near-simultaneous termination, there exists a view $U^\star$ such that every honest process outputs the value stored in $vec[U^\star]$. By Claim 1, all honest processes store the same vector in $vec[U^\star]$. Therefore, all honest processes output the same value. This proves consistency.

**Validity.** By the proof of near-simultaneous termination, there exists a view $U^\star$ such that every honest process outputs the value stored in $vec[U^\star]$. By Claim 2, for every honest process $P_j$, $vec[U^\star][j]$ is equal to $m_j$. Therefore, the common output vector contains the input of every honest process in the corresponding coordinate. This proves validity.

**Expected latency.** For each view $v$, define $Y_v : \mathsf{Exec}_{IC} \to \mathbb{N} \cup \{0\}$ to be the random variable that maps each execution $\epsilon \in \mathsf{Exec}_{IC}$ to the latency spent by view $v$ in execution $\epsilon$. Fix an execution $\epsilon \in \mathsf{Exec}_{IC}$, and assume that no honest process halts before the end of view $v$ in execution $\epsilon$. Hence, $Y_v(\epsilon) \le Lat_{\mathsf{LE}_v}(\epsilon) + Lat_{\mathsf{LoE}_v}(\epsilon) + Lat_{\mathsf{BB}_v}(\epsilon) + \Delta_{sim}$.

Now let $X' : \mathsf{Exec}_{IC} \to \mathbb{N}$ be the random variable corresponding to the first view with an honest leader. Then $X'$ is stochastically dominated by a random variable $X \sim \mathsf{Geom}(\varepsilon)$.

Since a new view starts every $c$ rounds, the end time of view $u$ is at most $1 + (u-1)c + Y_u(\epsilon)$. Therefore, $Lat_{\mathsf{IC}}(\epsilon) \le \max_{1 \le u \le U^\star} \big(1 + (u-1)c + Y_u(\epsilon)\big)$. Since LE and LoE have deterministic latency bounds, the random variables $Lat_{\mathsf{LE}_v}$ and $Lat_{\mathsf{LoE}_v}$ are constant. Hence,

$$\begin{aligned} Lat_{\mathsf{IC}}(\epsilon) &\le 1 + U^\star c + \mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \Delta_{sim} + \max_{1 \le u \le U^\star} \big(Lat_{\mathsf{BB}_u}(\epsilon)\big) \\ &\le 1 + X(\epsilon)c + \mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \Delta_{sim} + \max_{1 \le u \le X(\epsilon)} \big(Lat_{\mathsf{BB}_u}(\epsilon)\big). \end{aligned}$$

Taking the expectation over all executions, we obtain

$$\begin{aligned}\mathsf{lc}_{IC} &\le 1 + c\,\mathbb{E}[X] + \mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \Delta_{sim} + \mathbb{E}\left[\max_{1\le u\le X(\epsilon)} \left(Lat_{\mathsf{BB}_u}\right)\right] \\ &= 1 + \frac{c}{\varepsilon} + \mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \Delta_{sim} + \mathbb{E}\left[\max_{1\le u\le X(\epsilon)} \left(Lat_{\mathsf{BB}_u}\right)\right] \\ &= O(\mathsf{lc}_{\mathsf{LE}} + \mathsf{lc}_{\mathsf{LoE}} + \mathsf{lc}_{\mathsf{BB}})\,.\end{aligned}$$

**Expected communication.** For each view $v$, define $Z_v : \mathsf{Exec}_{IC} \to \mathbb{N} \cup \{0\}$ to be the random variable that maps each execution $\epsilon \in \mathsf{Exec}_{IC}$ to the communication spent by view $v$ in execution $\epsilon$. Fix an execution $\epsilon \in \mathsf{Exec}_{IC}$, and assume that no honest process halts before the end of view $v$ in execution $\epsilon$. Then $Z_v(\epsilon) \le Comm_{\mathsf{LE}}(\epsilon) + Comm_{\mathsf{LoE}}(\epsilon) + Comm_{\mathsf{BB}}(\epsilon)$. Hence $\mathbb{E}[Z_v] = cc_{\mathsf{LE}} + cc_{\mathsf{LoE}} + cc_{\mathsf{BB}}$.

Before the first honest output, every started view satisfies the assumed communication guarantees, since at most $t$ processes are corrupted. After the first honest output, some additional cost can be incurred during the next $\Delta_{sim}$ rounds before all honest processes halt. Indeed, in views that are still running at that point, one may observe more than $t$ *effective* crashes, since some honest processes may already have halted. For $\Delta_{sim} = 1$, which is the case in our instantiations, this overhead is immediately bound. More generally, for an arbitrary constant $\Delta_{sim}$, we use the assumed per-message complexity $\mathsf{pmc}_{\mathsf{BB}}$ of the underlying broadcast protocol: during these last $\Delta_{sim}$ rounds, the communication sent from one party to another in any round is at most $\mathsf{pmc}_{\mathsf{BB}}$. Therefore, even across all still-running views, the additional communication during these $\Delta_{sim}$ rounds is bounded by $n^2\Delta_{sim}\mathsf{pmc}_{\mathsf{BB}}$.

Since the protocol runs for $Lat_{IC}(\epsilon)$ rounds before all honest processes halt, and a new view starts every $c$ rounds, the total number of views started in execution $\epsilon$ is at most $\lceil Lat_{IC}(\epsilon)/c \rceil$. Therefore, the total communication in execution $\epsilon$ is upper bounded by $\sum_{v=1}^{\lceil Lat_{IC}(\epsilon)/c\rceil} Z_v(\epsilon) + n^2\Delta_{sim}\mathsf{pmc}_{\mathsf{BB}} + n^2(L_{\mathsf{in}} + \kappa)$, where the last term accounts for the initial dissemination of the inputs.

Taking the expectation over all executions, we obtain

$$\begin{aligned}cc_{IC} &= \mathbb{E}\left[\sum_{v=1}^{\lceil Lat_{IC}/c\rceil} Z_v\right] + n^2\Delta_{sim}\mathsf{pmc}_{\mathsf{BB}} + n^2(L_{\mathsf{in}} + \kappa) \\ &\le \left\lceil \frac{lc_{IC}}{c} \right\rceil \cdot \mathbb{E}[Z_1] + n^2\Delta_{sim}\mathsf{pmc}_{\mathsf{BB}} + n^2(L_{\mathsf{in}} + \kappa) \\ &= \left\lceil \frac{lc_{IC}}{c} \right\rceil \cdot \left(cc_{\mathsf{LE}} + cc_{\mathsf{LoE}} + cc_{\mathsf{BB}}\right) + n^2(\Delta_{sim}\mathsf{pmc}_{\mathsf{BB}} + L_{\mathsf{in}} + \kappa).\end{aligned}$$

□

## 5 Instantiations

In this section, we prove Corollaries 4.2 and 4.3 by providing concrete instantiations of the underlying primitives.

### 5.1 Dishonest Majority

*5.1.1 Learn-or-Expose.* Below we describe the protocol for LearnOrExpose. It follows the spirit of [40, 69]. We refer to Appendix E for additional intuition and more details.

**Algorithm 2** LearnOrExpose$^{dm}_{p_\ell}(\vec{v}, \vec{\sigma})$ at process $p_i$

**Local procedure:** Aggregate(Acc)
  for each $j \in [n]$, let $\mathrm{Acc}_j = \{\, \alpha \in \mathrm{Acc} : \alpha \text{ is a valid accusation against } p_j \,\}$
  return $\Pi$, where $\Pi[j] = cb(\mathrm{Acc}_j)$ for every $j \in [n]$ ▷ create a multi-signature
Acc $\leftarrow \emptyset$
Unresolved $\leftarrow \{p_j \in [n] : \vec{v}[j] = \perp\}$
**for** $j \in [n]$ **do**
  **if** $\vec{v}[j] \neq \perp$ **then**
    broadcast $(\langle \text{disclose}, \vec{v}[j]\rangle, \vec{\sigma}[j])$
**for** $r = 1$ to $R = \lceil 2n/(n-t)\rceil + 1$ **do**
  receive all newly delivered messages
  add all valid signed accusations to Acc
  forward all newly received valid accusations
  **for** each $p_j \in$ Unresolved **do**
    **if** received a valid signed message $(\langle \text{disclose}, m_j\rangle, \sigma_j)$ **then**
      $\vec{v}[j] \leftarrow m_j$
      $\vec{\sigma}[j] \leftarrow \sigma_j$
      broadcast $(\langle \text{disclose}, m_j\rangle, \sigma_j)$
      remove $p_j$ from Unresolved
  $G \leftarrow$ MakeGraph(Acc)
  **for** each $p_j \in$ Unresolved **do**
    **if** $p_i$ is disconnected from $p_j$ in $G$ **then**
      remove $p_j$ from Unresolved
  **if** Unresolved $= \emptyset$ **then**
    **if** $p_i = p_\ell$ **then**
      **return** $\vec{v}, (\vec{\sigma}, \text{Aggregate(Acc)})$
    **else**
      **return**
  Frontier $\leftarrow \emptyset$
  **for** each $p_j \in$ Unresolved **do**
    Frontier $\leftarrow$ Frontier $\cup$ $\left(N_G(p_i) \cap B_G(p_j, r-1)\right)$
  **for** each $p_j \in$ Frontier **do**
    broadcast $\langle \text{accuse}, p_j\rangle$ with signature

**Algorithm 3** MakeGraph(Acc) (EfficientMakeGraph from [40])

$G \leftarrow K_n$ ▷ initialize the complete graph on $\{p_1, \ldots, p_n\}$
**for** each valid accusation $\mathrm{acc}_{ij} \in$ Acc **do**
  remove edge $\{p_i, p_j\}$ from $G$
**while** there exists an edge $e = \{p_i, p_j\}$ in $G$ such that $|N_G(p_i) \cap N_G(p_j)| < n - t$ **do**
  remove $e$ from $G$
**return** $G$

Each party invokes LearnOrExpose on the candidate values and signatures collected during the initial input-dissemination stage. Throughout the protocol, parties keep forwarding all newly seen valid accusations, as well as all newly obtained valid disclose messages.

More precisely, each party $p_i$ maintains its own set of collected accusations Acc and, from it, constructs a local accusation graph $G_i :=$ MakeGraph(Acc). The goal is to resolve every party $p_j$ that is still missing from the current candidate vector. Informally, for every unresolved party $p_j$ in round $r$, process $p_i$ is in one of the following two situations: either $p_i$ has already obtained a valid disclose message from $p_j$, or the distance between $p_i$ and $p_j$ in $G_i$ is at least $r$. This invariant allows $p_i$ to accuse the neighbors that could keep $p_j$ too close in the graph.

Concretely, in each round, after incorporating all newly received accusations and disclose messages, process $p_i$ updates its graph and checks whether some unresolved node $p_j$ has become disconnected from $p_i$. If so, $p_j$ is removed from Unresolved. Otherwise, $p_i$ computes a frontier of nodes that lie on paths that are still too short and accuses them.

The key structural fact is that, for any two honest parties $p_i$ and $p_s$, the graph $G_i$ in round $r$ is a subgraph of the graph $G_s$ in round $r-1$. Intuitively, honest parties forward all newly seen accusations, so an honest process cannot lag behind another honest process by more than one round. Therefore, if in round $r$ some node $p_s$ is still at distance less than $r$ from $p_j$ in $G_i$, then in round $r-1$, process $p_s$ should already have obtained the disclose message of $p_j$. If it did not, then $p_s$ must itself be accused in round $r$.

Repeating this argument for $R = \lceil \frac{2n}{n-t} \rceil + 1$ rounds implies that, for every pair of parties $p_i, p_j$, either $p_i$ receives a valid disclose message from $p_j$, or the distance between them in $G_i$ exceeds $R$. The latter means that they are disconnected, since the diameter of $G_i$ is at most $R$. Therefore, after a constant number of rounds, every unresolved party is either learned or exposed.

Lemma 5.1. *Let* MakeGraph *be Algorithm 3.*

*Define* $\mathsf{VerifyVec}_i^{in}(\vec{v}, \vec{\sigma})$ *by* $\mathsf{VerifyVec}_i^{in}(\vec{v}, \vec{\sigma}) = \mathsf{true}$ *iff for every* $j \in [n]$ *with* $\vec{v}[j] \neq \perp$, $\mathsf{ver}_{\mathsf{pk}_j}(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \vec{\sigma}[j]) = \mathsf{true}$.

*Define* $\mathsf{VerifyVec}_i^{out}(\vec{v}, (\vec{\sigma}, \Pi))$ *by* $\mathsf{VerifyVec}_i^{out}(\vec{v}, (\vec{\sigma}, \Pi)) = \mathsf{true}$ *iff the following hold:*

1. *for every* $j \in [n]$ *with* $\vec{v}[j] \neq \perp$, $\mathsf{ver}_{\mathsf{pk}_j}(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \vec{\sigma}[j]) = \mathsf{true}$;
2. *from* $\Pi$ *one can extract an accusation set* Acc *such that, if* $G = \mathsf{MakeGraph}(\mathsf{Acc})$, *then* $p_i$ *is disconnected in* $G$ *from every process* $p_j$ *with* $\vec{v}[j] = \perp$.

*Then Algorithm 2 realizes Module 3 with latency* $\lceil \frac{2n}{n-t} \rceil + 1$ *rounds. Moreover, if accusations are aggregated per target party, then the proof size satisfies* $\lambda_{\mathsf{LoE}} = O(n^2 + n\kappa)$.

Proof. We prove the four properties of Module 3.

**Claim 1.** Honest parties never accuse honest parties.

*Proof of Claim 1.* The proof is essentially identical to the proof of Lemma 4 (Accusation Soundness) in [40]. For the sake of completeness, we provide the full proof in Appendix G. ◁

**Claim 2.** Every connected component of every graph obtained by Algorithm 3 has diameter at most $\lceil \frac{2n}{n-t} \rceil$.

*Proof of Claim 2.* This is exactly Lemma 13 in [40]. For the sake of completeness, we provide the full proof in Appendix G. ◁

*Termination.* Algorithm 2 executes exactly $R = \lceil \frac{2n}{n-t} \rceil + 1$ synchronous rounds. Hence, all honest processes terminate simultaneously after a fixed known latency $R$.

*Progress.* Assume that the leader $p_\ell$ is honest. First note that, for every coordinate $j$ with $\vec{v}[j] \neq \perp$, the definition of $\mathsf{VerifyVec}^{in}$ guarantees that $\vec{v}[j]$ is already accompanied by a valid disclose signature. Moreover, every disclose message sent by an honest process during the protocol carries a valid signature, by the Input Soundness precondition and the unforgeability of signatures.

Now, fix a party $p_j$ such that initially $\vec{v}[j] = \perp$, and for every process $p_i$ and round $r$, let $G_i^{(r)}$ be the graph computed by $p_i$ in line 16 in round $r$.

We use the same distance-growth argument as in [40]. We claim that if, in round $r$, before computing the graph $G_\ell^{(r)}$, the leader has not yet received a valid disclose message from $p_j$, then $\mathrm{dist}_{G_\ell^{(r)}}(p_\ell, p_j)$ is at least $r$. Indeed, in round 1, before computing the graph, the distance is trivially at least 1. Then the leader accuses $p_j$, and therefore in round 2 we have $p_j \notin N_{G_\ell^{(2)}}(p_\ell)$, so $\mathrm{dist}_{G_\ell^{(2)}}(p_\ell, p_j) \geq 2$. Inductively, suppose that in round $r$, before line 16, the party $p_j$ is still unresolved and $\mathrm{dist}_{G_\ell^{(r)}}(p_\ell, p_j) \geq r$. Then, after computing $G_\ell^{(r)}$, the leader accuses all nodes in $N_{G_\ell^{(r)}}(p_\ell) \cap B_{G_\ell^{(r)}}(p_j, r-1)$. This removes every path of length at most $r$ from $p_\ell$ to $p_j$. Therefore, if $p_j$ remains unresolved in the next round, then $\mathrm{dist}_{G_\ell^{(r+1)}}(p_\ell, p_j) \geq r+1$.

On the other hand, by Claim 2, every graph output by MakeGraph has diameter at most $\lceil \frac{2n}{n-t} \rceil$. Hence, after $R = \lceil \frac{2n}{n-t} \rceil + 1$ rounds, the leader either receives a valid disclose message from $p_j$, or $p_j$ becomes disconnected from the leader in $G_\ell^{(R)}$. In the latter case, the corresponding accusation proof is a valid polarizer against $p_j$ for the leader.

Since this holds for every initially unresolved coordinate, once $\mathsf{Unresolved} = \emptyset$ the honest leader returns some $(\vec{v}, \pi)$. This proves Progress.

*Transferability.* Suppose that for some honest $p_i$ we have $\mathsf{VerifyVec}_i^{out}(\vec{v}, \pi) = \mathsf{true}$. We show that every honest $p_{i'}$ also accepts.

For every coordinate $j$ with $\vec{v}[j] \neq \perp$, verification only checks a standard disclose signature by $p_j$, which is globally verifiable and therefore independent of the verifier. For the missing coordinates, by definition of $\mathsf{VerifyVec}_i^{out}$, from $\Pi$ one can extract an accusation set Acc such that, if $G = \mathsf{MakeGraph}(\mathsf{Acc})$, then $p_i$ is disconnected in $G$ from every process $p_j$ with $\vec{v}[j] = \perp$. By Claim 1, honest parties never accuse one another. Hence, all honest parties remain in the same clique, and therefore in the same connected component, of the graph output by MakeGraph. It follows that if one honest verifier $p_i$ is disconnected from some process $p_j$, then every honest verifier $p_{i'}$ is also disconnected from $p_j$. Therefore, the test for the missing coordinates has the same outcome at all honest processes. Thus $\mathsf{VerifyVec}_i^{out}(\vec{v}, \pi) = \mathsf{VerifyVec}_{i'}^{out}(\vec{v}, \pi)$ for all honest $p_i, p_{i'}$. This proves Transferability.

*Output Soundness.* Assume that an honest process $p_i$ accepts $(\vec{v}, \pi)$, that is, $\mathsf{VerifyVec}_i^{out}(\vec{v}, \pi) = \mathsf{true}$.

If $\vec{v}[j] \neq \perp$ and $p_j$ is honest, then, by Honest Input Agreement, Input Soundness, and the unforgeability of signatures, the value $\vec{v}[j]$ must be exactly the input submitted by $p_j$. If $\vec{v}[j] = \perp$ and $p_j$ is honest, then by definition of $\mathsf{VerifyVec}_i^{out}$ the proof $\Pi$ yields an accusation set Acc such that, if $G = \mathsf{MakeGraph}(\mathsf{Acc})$, then the honest verifier $p_i$ is disconnected in $G$ from $p_j$. But by Claim 1, honest parties never accuse one another, and therefore all honest parties remain in the same connected component of the graph produced by MakeGraph. This is impossible. Therefore, no honest process can be validly exposed.

Hence, for every honest party $p_j$, the entry $\vec{v}[j]$ is non-$\perp$ and equals the value submitted by $p_j$. This proves Soundness.

Finally, the latency is exactly $R$ rounds by construction. If accusations are aggregated per target party, then there are at most $n$ aggregated accusations, each of size $O(n+\kappa)$, so the total accusation part of the proof has size $O(n^2 + n\kappa)$. Together with the disclosed signatures, this yields $\lambda_{\mathsf{LoE}} = O(n^2 + n\kappa)$. □

*5.1.2 Leader Election.* For leader election, we use the static-adversary instantiation from [85]. The idea is the following. First, the adversary chooses the set of corrupted parties. Then, a common random

string $\mathrm{crs} \in \{0,1\}^\lambda$ is sampled, where $\lambda$ is the security parameter. After that, the protocol execution begins. In each epoch $e \geq 1$, every party locally computes the leader as $L_e := (\mathsf{PRF}_{\mathrm{crs}}(e) \bmod n) + 1$. Since the adversary is static and the common random string is chosen only after the corrupted set is fixed, the leader of each epoch is honest with probability at least $(n-t)/n$. In particular, in the dishonest-majority setting $t \leq (1-\varepsilon)n$, this probability is at least $\varepsilon$.

*5.1.3 Byzantine Broadcast.* For Byzantine Broadcast, we use the protocol of [85]. This protocol achieves expected constant latency and communication complexity $\tilde{O}(n^4)$ in the dishonest-majority setting, ignoring the security parameter $\kappa$ in the asymptotic notation for simplicity. More precisely, in the regime $t \leq (1-\varepsilon)n$, the protocol terminates after an expected $O(1/\varepsilon^2)$ rounds, and its latency has a geometric tail. The protocol also has per-message complexity $\mathrm{pmc}_{\mathsf{BB}} = \tilde{O}(n^2)$, since in any round a party sends to any fixed recipient at most the information corresponding to the current trust graph and the accumulated accusations.

At a very high level, the protocol relies on a weaker dissemination primitive called *TrustCast*, supported by a local *trust graph* maintained by every party. Intuitively, each party keeps track of which other parties it still considers compatible with honest behavior. This mechanism is closely related in spirit to our use of polarizers: rather than relying only on globally transferable certificates, parties also reason through a local graph structure that evolves as misbehavior is exposed. The key guarantee of TrustCast is that, for every honest party, by the end of the primitive either it receives a valid message signed by the sender, or it removes the sender from its trust graph. Moreover, provided the underlying verification rule satisfies suitable monotonicity and validity-at-origin conditions, honest parties never distrust one another.

The protocol then proceeds in epochs. In epoch 1, the leader is the designated sender. In every later epoch, the leader is chosen uniformly at random by the leader-election oracle. Each epoch consists of three phases.

*Propose.* The leader sends a proposal consisting of a value together with the freshest commit evidence it has seen so far. If it has not seen any nontrivial commit evidence yet, then in epoch 1 it proposes the sender's input, whereas in later epochs it proposes $\perp$. The leader disseminates this proposal using TrustCast.

*Vote.* Each party considers the leader's proposal provided that the leader is still present in its trust graph. In that case, it TrustCasts a vote for the proposed value. If the leader has already been removed from its trust graph, it votes $\perp$.

*Commit.* If a party sees that every party still present in its trust graph voted for the same value, then it can form commit evidence for that value, namely the collection of signed vote messages from all parties in its current trust graph. It then TrustCasts a commit message carrying that evidence. Otherwise, it TrustCasts $\perp$. In the background, parties continuously check a termination condition: if a party has received commit messages from all parties in its current trust graph, and all these messages support the same valid commit evidence for some value, then it outputs that value and terminates.

Although the presentation in [85] is given for binary values, the proof extends directly to long messages, as also noted explicitly by the authors. One could therefore replace every occurrence of the bit $b$ by a message $m$. At first sight, this may seem to increase the communication substantially: in the binary case, the adversary can equivocate between at most two values, whereas for messages of length $L_{\mathsf{in}}$ it could in principle send up to $2^{L_{\mathsf{in}}}$ different signed messages. However, the proof does not rely on the number of candidate values being small. It relies only on the fact that two valid commit evidences for two different values cannot both be generated. Hence, even if a dishonest sender equivocates among many messages, honest processes can create commit evidence for at most one of them, and among all commit messages acceptable to honest processes, at most one value can propagate through the protocol.

In the protocol, every honest node sends to every other process, in each round, at most two messages originating from the sender. Indeed, two messages are sufficient to prove equivocation, while no more than one message is needed to disperse the value. We do not count here the communication of messages that do not contain information about the sender's value. Therefore, any message sent between two parties that carries information about the sender's value contains at most $O(2L_{\mathsf{in}}^{BB} + \kappa)$ bits. Since the expected number of rounds is $O(1/\varepsilon^2)$, the total expected communication of all such value-carrying messages is bounded by $O\left(\frac{1}{\varepsilon^2} \cdot n^2(L_{\mathsf{in}}^{BB} + \kappa)\right) = O(n^2(L_{\mathsf{in}}^{BB} + \kappa))$. Here $L_{\mathsf{in}}^{BB}$ denotes the input length of Byzantine Broadcast; in our application, $L_{\mathsf{in}}^{BB} = O(n^2 + n(L_{\mathsf{in}}^{IC} + \kappa))$.

All remaining messages, including the trustcast control messages, contain at most $O(n^2 + n\kappa)$ bits. Note that, in the protocol as stated, this can actually be $\Omega(n^2\kappa)$ in the worst case, since one may need to forward the whole trust graph together with all signatures. However, using the same aggregation technique as we use for polarizers, namely multi-signatures, one can save this extra $\kappa$ factor. Hence their total contribution is $O(n^4 + n^3\kappa)$. Therefore, the resulting communication complexity is $O(n^4 + n^3\kappa + n^2 L_{\mathsf{in}}^{BB})$.

## 5.2 Honest Majority

We defer the full proof of Corollary 4.3 to Appendix F.

In the honest-majority setting, we use a threshold-signature-based Learn-or-Expose primitive $\mathsf{LoE}^{\mathsf{hm}}$, instantiated by Algorithm 9 following [25]. For leader election, we use the approach underlying [19]. For Byzantine Broadcast, we use the protocol of [3]. For $\kappa$-bit inputs, it provides Byzantine Broadcast with expected $O(1)$ rounds and expected $O(n^2)$ communication.

# 6 Conclusion

We presented the first Interactive Consistency protocols achieving constant expected latency together with either dishonest-majority resilience or quadratic communication in honest majority. Our results follow from a common compiler that reduces IC to standard agreement primitives through leader election, Learn-or-Expose, and Byzantine Broadcast (or alternatively, Validated Byzantine Agreement in honest majority).

Several directions remain open. In dishonest majority, it would be desirable to reduce the communication complexity of the underlying Byzantine Broadcast primitive. In both honest and dishonest majority setting, one may ask whether the compiler can be refined to achieve similar complexity against an adaptive adversary.

## References


[1] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. 2024. Asynchronous Agreement on a Core Set in Constant Expected Time and More Efficient Asynchronous VSS and MPC. In *Theory of Cryptography - 22nd International Conference, TCC 2024, Milan, Italy, December 2-6, 2024, Proceedings, Part IV (Lecture Notes in Computer Science)*, Elette Boyle and Mohammad Mahmoody (Eds.). Springer, 451–482. doi:10.1007/978-3-031-78023-3_15

[2] Ittai Abraham, T.-H. Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. 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*, Peter Robinson and Faith Ellen (Eds.). ACM, 317–326. doi:10.1145/3293611.3331629

[3] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. 2019. Synchronous Byzantine Agreement with Expected O(1) Rounds, Expected $O(n^2)$ Communication, and Optimal Resilience. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers (Lecture Notes in Computer Science)*, Ian Goldberg and Tyler Moore (Eds.). Springer, 320–334. doi:10.1007/978-3-030-32101-7_20

[4] Ittai Abraham, Danny Dolev, Alon Kagan, and Gilad Stern. 2022. Brief Announcement: Authenticated Consensus in Synchronous Systems with Mixed Faults. In *36th International Symposium on Distributed Computing, DISC 2022, Augusta, Georgia, USA, October 25-27, 2022 (LIPIcs)*, Christian Scheideler (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 38:1–38:3. doi:10.4230/LIPICS.DISC.2022.38

[5] Marcos Kawazoe Aguilera and Sam Toueg. 1999. A Simple Bivalency Proof that $t$-Resilient Consensus Requires $t + 1$ Rounds. *Inf. Process. Lett.* 71, 3-4 (1999), 155–158. doi:10.1016/S0020-0190(99)00100-3

[6] Andreea B. Alexandru, Julian Loss, Charalampos Papamanthou, Giorgos Tsimos, and Benedikt Wagner. 2025. Sublinear-Round Broadcast without Trusted Setup. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, Yossi Azar and Debmalya Panigrahi (Eds.). SIAM, 4132–4171. doi:10.1137/1.9781611978322.141

[7] Nicolas Alhaddad, Sisi Duan, Mayank Varia, Haochen Wang, and Haibin Zhang. 2025. Efficient Byzantine Broadcast From Succinct Erasure Coding Proof System. *IEEE Trans. Inf. Forensics Secur.* 20 (2025), 8583–8596. doi:10.1109/TIFS.2025.3592564

[8] Nicolas Alhaddad, Sisi Duan, Mayank Varia, and Haibin Zhang. 2021. Succinct Erasure Coding Proof Systems. *IACR Cryptol. ePrint Arch.* 2021 (2021), 1500. https://eprint.iacr.org/2021/1500

[9] Marcin Andrychowicz and Stefan Dziembowski. 2015. PoW-Based Distributed Cryptography with No Trusted Setup. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer, 379–399. doi:10.1007/978-3-662-48000-7_19

[10] Gilad Asharov and Anirudh Chandramouli. 2024. 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 (Lecture Notes in Computer Science)*, Marc Joye and Gregor Leander (Eds.). Springer, 310–339. doi:10.1007/978-3-031-58740-5_11

[11] Yonatan Aumann and Yehuda Lindell. 2007. Security Against Covert Adversaries: Efficient Protocols for Realistic Adversaries. In *Theory of Cryptography, 4th Theory of Cryptography Conference, TCC 2007, Amsterdam, The Netherlands, February 21-24, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4392)*, Salil P. Vadhan (Ed.). Springer, 137–156. doi:10.1007/978-3-540-70936-7_8

[12] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2024. GRandLine: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 941–955. doi:10.1145/3658644.3690287

[13] Zuzana Beerliová-Trubíniová and Martin Hirt. 2006. 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 (Lecture Notes in Computer Science)*, Shai Halevi and Tal Rabin (Eds.). Springer, 305–328. doi:10.1007/11681878_16

[14] Erica Blum, Elette Boyle, Ran Cohen, and Chen-Da Liu-Zhang. 2025. Communication lower bounds for cryptographic broadcast protocols. *Distributed Comput.* 38, 1 (2025), 1–17. doi:10.1007/S00446-024-00473-5

[15] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2567)*, Yvo Desmedt (Ed.). Springer, 31–46. doi:10.1007/3-540-36288-6_3

[16] Dan Boneh, Manu Drijvers, and Gregory Neven. 2018. Compact Multi-signatures for Smaller Blockchains. In *Advances in Cryptology - ASIACRYPT 2018 - 24th International Conference on the Theory and Application of Cryptology and Information Security, Brisbane, QLD, Australia, December 2-6, 2018, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 11273)*, Thomas Peyrin and Steven D. Galbraith (Eds.). Springer, 435–464. doi:10.1007/978-3-030-03329-3_15

[17] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. 2003. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Advances in Cryptology - EUROCRYPT 2003, International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, Proceedings (Lecture Notes in Computer Science, Vol. 2656)*, Eli Biham (Ed.). Springer, 416–432. doi:10.1007/3-540-39200-9_26

[18] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. 2011. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer. doi:10.1007/978-3-642-15260-3

[19] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *J. Cryptol.* 18, 3 (2005), 219–246. doi:10.1007/s00145-005-0318-0

[20] Ran Canetti. 2004. Universally Composable Signature, Certification, and Authentication. In *17th IEEE Computer Security Foundations Workshop, (CSFW-17 2004), 28-30 June 2004, Pacific Grove, CA, USA*. IEEE Computer Society, 219. doi:10.1109/CSFW.2004.24

[21] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. 2020. Sublinear-Round Byzantine Agreement Under Corrupt Majority. In *Public-Key Cryptography - PKC 2020 - 23rd IACR International Conference on Practice and Theory of Public-Key Cryptography, Edinburgh, UK, May 4-7, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Aggelos Kiayias, Markulf Kohlweiss, Petros Wallden, and Vassilis Zikas (Eds.). Springer, 246–265. doi:10.1007/978-3-030-45388-6_9

[22] Wonseok Choi, Ran Cohen, Juan Garay, Nikos Skoumios, and Vassilis Zikas. 2025. Byzantine Broadcast with Unknown Participants. *IACR Cryptol. ePrint Arch.* 2025 (2025), 2133. https://eprint.iacr.org/2025/2133

[23] Wutichai Chongchitmate and Rafail Ostrovsky. 2018. 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 (Lecture Notes in Computer Science)*, Amos Beimel and Stefan Dziembowski (Eds.). Springer, 370–388. doi:10.1007/978-3-030-03807-6_14

[24] Benny Chor, Michael Merritt, and David B. Shmoys. 1985. Simple Constant-Time Consensus Protocols in Realistic Failure Models (Extended Abstract). In *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985*, Michael A. Malcolm and H. Raymond Strong (Eds.). ACM, 152–162. doi:10.1145/323596.323610

[25] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. 2024. DARE to agree: Byzantine Agreement with Optimal Resilience and Adaptive Communication. Cryptology ePrint Archive, Paper 2024/403. https://eprint.iacr.org/2024/403

[26] Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Manuel Vidigueira, and Igor Zablotchi. 2025. Partial Synchrony for Free: New Upper Bounds for Byzantine Agreement. In *Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2025, New Orleans, LA, USA, January 12-15, 2025*, Yossi Azar and Debmalya Panigrahi (Eds.). SIAM, 4227–4291. doi:10.1137/1.9781611978322.144

[27] Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. 2024. 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*, Ran Gelles, Dennis Olivetti, and Petr Kuznetsov (Eds.). ACM, 157–169. doi:10.1145/3662158.3662780

[28] Richard Cleve. 1986. Limits on the Security of Coin Flips when Half the Processors Are Faulty (Extended Abstract). In *Proceedings of the 18th Annual ACM Symposium on Theory of Computing, May 28-30, 1986, Berkeley, California, USA*, Juris Hartmanis (Ed.). ACM, 364–369. doi:10.1145/12130.12168

[29] Ran Cohen, Poulami Das, and Tal Moran. 2025. Setup-Free Committee Sampling with Subquadratic Communication. In *Advances in Cryptology - ASIACRYPT 2025 - 31st International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, VIC, Australia, December 8-12, 2025, Proceedings, Part II (Lecture Notes in Computer Science)*, Goichiro Hanaoka and Bo-Yin Yang (Eds.). Springer, 273–304. doi:10.1007/978-981-95-5096-8_9

[30] Ran Cohen, Jack Doerner, Yashvanth Kondi, and Abhi Shelat. 2022. Guaranteed Output in $O(\sqrt{n})$ Rounds for Round-Robin Sampling Protocols. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 241–271. doi:10.1007/978-3-031-06944-4_9

[31] Ran Cohen, Juan A. Garay, and Vassilis Zikas. 2020. Broadcast-Optimal Two-Round MPC. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 828–858. doi:10.1007/978-3-030-45724-2_28

[32] Ran Cohen and Yehuda Lindell. 2014. Fairness versus Guaranteed Output Delivery in Secure Multiparty Computation. 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 (Lecture Notes in Computer Science, Vol. 8874)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, 466–485. doi:10.1007/978-3-662-45608-8_25

[33] Shir Cohen, Idit Keidar, and Alexander Spiegelman. 2022. Brief Announcement: Make Every Word Count: Adaptive Byzantine Agreement with Fewer Words. In *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, Alessia Milani and Philipp Woelfel (Eds.). ACM, 421–423. doi:10.1145/3519270.3538458

[34] Daniel Collins, Sisi Duan, Julian Loss, Charalampos Papamanthou, Giorgos Tsimos, and Haochen Wang. 2025. Towards Optimal Parallel Broadcast Under a Dishonest Majority. In *Financial Cryptography and Data Security - 29th International Conference, FC 2025, Miyakojima, Japan, April 14-18, 2025, Revised Selected Papers, Part I (Lecture Notes in Computer Science)*, Christina Garman and Pedro Moreno-Sanchez (Eds.). Springer, 345–363. doi:10.1007/978-3-032-07024-1_20

[35] Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer. 2025. Validity in Network-Agnostic Byzantine Agreement. In *39th International Symposium on Distributed Computing, DISC 2025, Berlin, Germany, October 27-31, 2025 (LIPIcs)*, Dariusz R. Kowalski (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 24:1–24:23. doi:10.4230/LIPICS.DISC.2025.24

[36] Ivan Damgård, Bernardo Magri, Divya Ravi, Luisa Siniscalchi, and Sophia Yakoubov. 2021. Broadcast-Optimal Two Round MPC with an Honest Majority. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II (Lecture Notes in Computer Science)*, Tal Malkin and Chris Peikert (Eds.). Springer, 155–184. doi:10.1007/978-3-030-84245-1_6

[37] Sourav Das, Sisi Duan, Shengqi Liu, Atsuki Momose, Ling Ren, and Victor Shoup. 2024. Asynchronous Consensus without Trusted Setup or Public-Key Cryptography. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security, CCS 2024, Salt Lake City, UT, USA, October 14-18, 2024*, Bo Luo, Xiaojing Liao, Jun Xu, Engin Kirda, and David Lie (Eds.). ACM, 3242–3256. doi:10.1145/3658644.3670327

[38] Sourav Das and Ling Ren. 2024. Adaptively Secure BLS Threshold Signatures from DDH and co-CDH. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VII (Lecture Notes in Computer Science)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, 251–284. doi:10.1007/978-3-031-68394-7_9

[39] Sourav Das, Zhuolun Xiang, and Ling Ren. 2024. Powers of Tau in Asynchrony. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society. https://www.ndss-symposium.org/ndss-paper/powers-of-tau-in-asynchrony/

[40] Giovanni Deligios, Ivana Klasovita, and Chen-Da Liu-Zhang. 2025. Asymptotically Optimal Early Termination for Dishonest Majority Broadcast. In *Advances in Cryptology – EUROCRYPT 2025*, Serge Fehr and Pierre-Alain Fouque (Eds.). Springer Nature Switzerland, Cham, 116–144.

[41] Danny Dolev and H. Raymond Strong. 1983. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666. doi:10.1137/0212045

[42] Paul Feldman and Silvio Micali. 1988. Optimal Algorithms for Byzantine Agreement. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, Janos Simon (Ed.). ACM, 148–161. doi:10.1145/62212.62225

[43] Hanwen Feng, Zhenliang Lu, and Qiang Tang. 2024. Dragon: Decentralization at the cost of Representation after Arbitrary Grouping and Its Applications to Sub-cubic DKG and Interactive Consistency. In *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, Ran Gelles, Dennis Olivetti, and Petr Kuznetsov (Eds.). ACM, 469–479. doi:10.1145/3662158.3662771

[44] Hanwen Feng and Qiang Tang. 2025. Asymptotically Optimal Adaptive Asynchronous Common Coin and DKG with Silent Setup. In *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part III (Lecture Notes in Computer Science)*, Yael Tauman Kalai and Seny F. Kamara (Eds.). Springer, 642–675. doi:10.1007/978-3-032-01881-6_20

[45] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. 1985. 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*, Michael A. Malcolm and H. Raymond Strong (Eds.). ACM, 59–70. doi:10.1145/323596.323602

[46] Matthias Fitzi and Martin Hirt. 2006. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC 2006, Denver, CO, USA, July 23-26, 2006*, Eric Ruppert and Dahlia Malkhi (Eds.). ACM, 163–168. doi:10.1145/1146381.1146407

[47] Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. 2021. A New Way to Achieve Round-Efficient Byzantine Agreement. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen (Eds.). ACM, 355–362. doi:10.1145/3465084.3467907

[48] Matthias Fitzi and Jesper Buus Nielsen. 2009. On the Number of Synchronous Rounds Sufficient for Authenticated Byzantine Agreement. In *Distributed Computing, 23rd International Symposium, DISC 2009, Elche, Spain, September 23-25, 2009. Proceedings (Lecture Notes in Computer Science)*, Idit Keidar (Ed.). Springer, 449–463. doi:10.1007/978-3-642-04355-0_46

[49] Eli Gafni. 1998. Round-by-Round Fault Detectors: Unifying Synchrony and Asynchrony (Extended Abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing, PODC '98, Puerto Vallarta, Mexico, June 28 - July 2, 1998*, Brian A. Coan and Yehuda Afek (Eds.). ACM, 143–152. doi:10.1145/277697.277724

[50] David Galindo and Jia Liu. 2022. Robust Subgroup Multi-signatures for Consensus. In *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings (Lecture Notes in Computer Science, Vol. 13161)*, Steven D. Galbraith (Ed.). Springer, 537–561. doi:10.1007/978-3-030-95312-6_22

[51] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. 2007. Round Complexity of Authenticated Broadcast with a Dishonest Majority. In *48th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2007, Providence, RI, USA, October 20-23, 2007, Proceedings*. IEEE Computer Society, 658–668. doi:10.1109/FOCS.2007.44

[52] Juan A. Garay and Aggelos Kiayias. 2020. SoK: A Consensus Taxonomy in the Blockchain Era. In *Topics in Cryptology - CT-RSA 2020 - The Cryptographers' Track at the RSA Conference 2020, San Francisco, CA, USA, February 24-28, 2020, Proceedings (Lecture Notes in Computer Science)*, Stanislaw Jarecki (Ed.). Springer, 284–318. doi:10.1007/978-3-030-40186-3_13

[53] Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. 2018. Bootstrapping the Blockchain, with Applications to Consensus and Fast PKI Setup. In *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II (Lecture Notes in Computer Science)*, Michel Abdalla and Ricardo Dahab (Eds.). Springer, 465–495. doi:10.1007/978-3-319-76581-5_16

[54] Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. 2020. Resource-Restricted Cryptography: Revisiting MPC Bounds in the Proof-of-Work Era. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II (Lecture Notes in Computer Science)*, Anne Canteaut and Yuval Ishai (Eds.). Springer, 129–158. doi:10.1007/978-3-030-45724-2_5

[55] Juan A. Garay, Aggelos Kiayias, and Giorgos Panagiotakos. 2020. Blockchains from Non-idealized Hash Functions. In *Theory of Cryptography - 18th International Conference, TCC 2020, Durham, NC, USA, November 16-19, 2020, Proceedings, Part I (Lecture Notes in Computer Science)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 291–321. doi:10.1007/978-3-030-64375-1_11

[56] Juan A. Garay, Aggelos Kiayias, and Yu Shen. 2024. Proof-of-Work-Based Consensus in Expected-Constant Time. 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 III (Lecture Notes in Computer Science)*, Marc Joye and Gregor Leander (Eds.). Springer, 96–125. doi:10.1007/978-3-031-58734-4_4

[57] Ran Gelles, Christoph Lenzen, Julian Loss, and Sravya Yandamuri. 2025. Nearly Optimal Parallel Broadcast in the Plain Public Key Model. In *Advances in Cryptology - CRYPTO 2025 - 45th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2025, Proceedings, Part II (Lecture Notes in Computer Science)*, Yael Tauman Kalai and Seny F. Kamara (Eds.). Springer, 101–134. doi:10.1007/978-3-032-01878-6_4

[58] Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. 2022. Round-Optimal Byzantine Agreement. In *Advances in Cryptology - EUROCRYPT 2022 - 41st Annual International Conference on the Theory and Applications of Cryptographic Techniques, Trondheim, Norway, May 30 - June 3, 2022, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 13275)*, Orr Dunkelman and Stefan Dziembowski (Eds.). Springer, 96–119. doi:10.1007/978-3-031-06944-4_4

[59] Diana Ghinea, Ivana Klasovitá, and Chen-Da Liu-Zhang. 2026. Round-Optimal Byzantine Agreement without Trusted Setup. *Cryptology ePrint Archive* (2026).

[60] Oded Goldreich. 2004. *The Foundations of Cryptography - Volume 2: Basic Applications*. Cambridge University Press. doi:10.1017/CBO9780511721656

[61] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. 2015. Constant-Round MPC with Fairness and Guarantee of Output Delivery. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II (Lecture Notes in Computer Science)*, Rosario Gennaro and Matthew Robshaw (Eds.). Springer, 63–82. doi:10.1007/978-3-662-48000-7_4

[62] Vipul Goyal, Yifan Song, and Chenzhi Zhu. 2020. 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 (Lecture Notes in Computer Science)*, Daniele Micciancio and Thomas Ristenpart (Eds.). Springer, 618–646. doi:10.1007/978-3-030-56880-1_22

[63] Martin Hirt and Pavel Raykov. 2014. 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 (Lecture Notes in Computer Science)*, Palash Sarkar and Tetsu Iwata (Eds.). Springer, 448–465. doi:10.1007/978-3-662-45608-8_24

[64] Yuval Ishai, Rafail Ostrovsky, and Vassilis Zikas. 2014. Secure Multi-Party Computation with Identifiable Abort. In *Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 8617)*, Juan A. Garay and Rosario Gennaro (Eds.). Springer, 369–386. doi:10.1007/978-3-662-44381-1_21

[65] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *Advances in Cryptology - ASIACRYPT 2010 - 16th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 5-9, 2010. Proceedings (Lecture Notes in Computer Science, Vol. 6477)*, Masayuki Abe (Ed.). Springer, 177–194. doi:10.1007/978-3-642-17373-8_11

[66] Jonathan Katz and Chiu-Yuen Koo. 2006. On Expected Constant-Round Protocols for Byzantine Agreement. In *Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings (Lecture Notes in Computer Science)*, Cynthia Dwork (Ed.). Springer, 445–462. doi:10.1007/11818175_27

[67] Andrew Lewis-Pye and Tim Roughgarden. 2023. Byzantine Generals in the Permissionless Setting. In *Financial Cryptography and Data Security - 27th International Conference, FC 2023, Bol, Brač, Croatia, May 1-5, 2023, Revised Selected Papers, Part I (Lecture Notes in Computer Science)*, Foteini Baldimtsi and Christian Cachin (Eds.). Springer, 21–37. doi:10.1007/978-3-031-47754-6_2

[68] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. 2002. 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*, Aleta Ricciardi (Ed.). ACM, 203–212. doi:10.1145/571825.571859

[69] Julian Loss and Jesper Buus Nielsen. 2024. Early Stopping for Any Number of Corruptions. In *Advances in Cryptology – EUROCRYPT 2024*, Marc Joye and Gregor Leander (Eds.). Springer Nature Switzerland, Cham, 457–488.

[70] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC '20: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, August 3-7, 2020*, Yuval Emek and Christian Cachin (Eds.). ACM, 129–138. doi:10.1145/3382734.3405707

[71] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *34th International Symposium on Distributed Computing, DISC 2020, Virtual Conference, October 12-16, 2020 (LIPIcs)*, Hagit Attiya (Ed.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:17. doi:10.4230/LIPICS.DISC.2020.28

[72] Lucien K. L. Ng, Pedro Moreno-Sanchez, Mohsen Minaei, Panagiotis Chatzigiannis, Adithya Bhat, and Duc Viet Le. 2025. Lite-PoT: Practical Powers-of-Tau Setup Ceremony. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (Eds.). ACM, 2114–2128. doi:10.1145/3719027.3765182

[73] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *Topics in Cryptology - CT-RSA 2005, The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005, Proceedings (Lecture Notes in Computer Science)*, Alfred Menezes (Ed.). Springer, 275–292. doi:10.1007/978-3-540-30574-3_19

[74] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234. doi:10.1145/322186.322188

[75] Birgit Pfitzmann and Michael Waidner. 1996. *Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$*. IBM Armonk.

[76] Sergio Rajsbaum. 2022. Distributed Decision Problems: Concurrent Specifications Beyond Binary Relations (Invited Talk). In *33rd International Conference on Concurrency Theory, CONCUR 2022, Warsaw, Poland, September 12-16, 2022 (LIPIcs)*, Bartek Klin, Slawomir Lasota, and Anca Muscholl (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 3:1–3:13. doi:10.4230/LIPICS.CONCUR.2022.3

[77] Matthieu Rambaud. 2023. 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 (2023), 1757. https://eprint.iacr.org/2023/1757

[78] Matthieu Rambaud and Christophe Levrat. 2024. Practical Non-interactive Multi-signatures, and a Multi-to-Aggregate Signatures Compiler. *IACR Cryptol. ePrint Arch.* (2024), 1081. https://eprint.iacr.org/2024/1081

[79] Thomas Ristenpart and Scott Yilek. 2007. The Power of Proofs-of-Possession: Securing Multiparty Signatures against Rogue-Key Attacks. In *Advances in Cryptology - EUROCRYPT 2007, 26th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Barcelona, Spain, May 20-24, 2007, Proceedings (Lecture Notes in Computer Science, Vol. 4515)*, Moni Naor (Ed.). Springer, 228–245. doi:10.1007/978-3-540-72540-4_13

[80] Marc Rivinius, Pascal Reisert, Daniel Rausch, and Ralf Küsters. 2022. Publicly Accountable Robust Multi-Party Computation. In *43rd IEEE Symposium on Security and Privacy, SP 2022, San Francisco, CA, USA, May 22-26, 2022*. IEEE, 2430–2449. doi:10.1109/SP46214.2022.9833608

[81] Victor Shoup. 2024. A Theoretical Take on a Practical Consensus Protocol. *IACR Cryptol. ePrint Arch.* 2024 (2024), 696. https://eprint.iacr.org/2024/696

[82] Nibesh Shrestha, Ittai Abraham, and Kartik Nayak. 2025. Communication and Round Efficient Parallel Broadcast Protocols. In *Financial Cryptography and Data Security - 29th International Conference, FC 2025, Miyakojima, Japan, April 14-18, 2025, Revised Selected Papers, Part I (Lecture Notes in Computer Science)*, Christina Garman and Pedro Moreno-Sanchez (Eds.). Springer, 327–344. doi:10.1007/978-3-032-07024-1_19

[83] T. K. Srikanth and Sam Toueg. 1987. Optimal clock synchronization. *J. ACM* 34, 3 (1987), 626–645. doi:10.1145/28869.28876

[84] Georgios Tsimos, Julian Loss, and Charalampos Papamanthou. 2022. Gossiping for Communication-Efficient Broadcast. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part III (Lecture Notes in Computer Science)*, Yevgeniy Dodis and Thomas Shrimpton (Eds.). Springer, 439–469. doi:10.1007/978-3-031-15982-4_15

[85] Jun Wan, Hanshen Xiao, Elaine Shi, and Srinivas Devadas. 2020. 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 (Lecture Notes in Computer Science)*, Rafael Pass and Krzysztof Pietrzak (Eds.). Springer, 381–411. doi:10.1007/978-3-030-64375-1_14


# A Cryptographic Details

This section provides the omitted definitions of underlying cryptographic primitives.

## A.1 Cryptography within Dishonest Majority

***Aggregate / multi-signatures.*** We assume the existence of a dynamic accountable non-interactive compact signing scheme (DANCS) capturing both aggregate-signatures and common-message multi-signatures [15, 16, 50, 78, 79]. Its purpose is to compress certificates attesting that a group $G$ of processes accused some process $p_a$, i.e., that members of $G$ signed accusation statements. Concretely, signers certify either a common message $m^\star = (\text{Accuse}, p_a)$ (multi-signature setting), or signer-distinct messages $m_i^\star = (p_i, \text{Accuse}, p_a)$ (aggregate-signature setting).

Formally, DANCS is a tuple of polynomial-time algorithms

$$\text{DANCS} = (\text{setup}, \text{kg}, \text{sign}, \text{cb}, \text{ver})$$

defined as follows.

- $\text{setup}(1^\kappa) \to \text{pp}$: outputs public parameters.
- $\text{kg}() \to (\text{pk}_i, \text{sk}_i, \pi_i)$: outputs a public/secret key pair for process $p_i$, and optionally a non-interactive proof-of-possession $\pi_i$.
- $\text{sign}(\text{sk}_i, m_i) \to \sigma_i$: outputs an individual signature on message $m_i \in \{0,1\}^{poly(\kappa)}$.
- $\text{cb}(\{(\text{pk}_i, m_i, \sigma_i)\}_{p_i \in G}) \to \sigma/\perp$: combines valid individual signatures into a compact certificate $\sigma$, or returns $\perp$.
- $\text{ver}(\sigma, \{(\text{pk}_i, m_i)\}_{p_i \in G}) \to \{0,1\}$: verifies the compact certificate with respect to signer set $G$ and associated messages.

The scheme satisfies standard correctness, unforgeability, and accountability properties [78, §III]. In particular, no PPT adversary controlling processes outside $G$ can produce a valid certificate claiming accusations by honest members of $G$ that did not sign.

Multi-signatures correspond to the special case where all messages are equal, i.e., $m_i = m^\star$ for every $p_i \in G$. Aggregate-signatures correspond to the case where messages are pairwise distinct (or at least signer-bound), e.g., $m_i = (p_i, \text{Accuse}, p_a)$.

| Scheme | Security / rogue-key defense | Number of pairings per verification |
|---|---|---|
| BGLS fNIA [17] | co-CDH + ROM | $\lvert G\rvert + 1$ |
| PoP-based fNIM [78] | co-CDH + ROM + DL + AGM | 2 |

**Table 3: Comparison of compact certificate mechanisms for accusation certificates. BGLS naturally signs signer-distinct accusation messages $m_i^\star = (p_i, \text{Accuse}, p_a)$, while PoP-based multisignatures support the common accusation message $m^\star = (\text{Accuse}, p_a)$.**

Practical compact schemes are typically pairing-based. In that setting, common-message aggregation may be vulnerable to *rogue-key attacks*, where an adversary registers a malicious public key algebraically correlated with honest keys so as to forge group certificates. This issue can be prevented in several standard ways:

- by signer-binding messages, e.g. using $m_i = (p_i, m^\star)$, which reduces multi-signatures to aggregate-signatures, but typically loses the fastest constant-size verification and may require $\Theta(|G|)$ exponentiations;
- by requiring a proof-of-possession (PoP) during key registration. In the dms scheme [78], each process publishes $(\mathsf{pk}_i, \pi_i)$, where is a non-interactive Schnorr-like PoP for the discrete logarithm of $pk_i$. The verification of one PoP costs essentially two scalar multiplications, one group addition, and one hash-to-scalar. Compared with B(G)LS aggregate signatures, the price is the extra PoP $\pi_i$ attached to each public key and its one-time verification; after registration, common-message multi-signature verification remains constant-size, essentially two pairings, instead of the $\Theta(|G|)$-pairing verification of aggregate BGLS on signer-bound messages;

In our setting, any of the above instantiations is acceptable. The only required abstraction is that accusation certificates by a group $G$ can be compressed into $O(\kappa)$ bits beyond the description of $G$.

## A.2 Cryptography within Honest Majority

***$(t, n)$-threshold signatures.*** We also consider dynamic accountable non-interactive threshold signature schemes (DANTS), where any subset $G \subseteq [n]$ of size at least $t+1$ can jointly certify a message, while the verifier need not know the exact signer set $G$, but only the threshold parameters $(t, n)$. Such schemes are particularly natural when any quorum of $t + 1$ accusations should suffice.

Formally, DANTS is a tuple of polynomial-time algorithms

$$\mathsf{DANTS} = (\mathsf{setup}, \mathsf{kg}, \mathsf{psign}, \mathsf{pver}, \mathsf{cb}, \mathsf{ver})$$

defined as follows.

- $\mathsf{setup}(1^\kappa) \to \mathsf{pp}$: outputs public parameters.
- $\mathsf{kg}() \to (\mathsf{pk}, \{(\mathsf{pk}_i, \mathsf{sk}_i)\}_{i\in[n]})$: outputs a global verification key pk, together with public verification shares and secret signing shares for the $n$ processes.
- $\mathsf{psign}(\mathsf{sk}_i, m) \to \sigma_i$: process $p_i$ outputs a partial signature on message $m$.
- $\mathsf{pver}(\mathsf{pk}_i, m, \sigma_i) \to \{0, 1\}$: verifies a partial signature share.
- $\mathsf{cb}(m, \{(\mathsf{pk}_i, \sigma_i)\}_{p_i\in G}) \to \sigma/\perp$: on any set $G$ with $|G| \geq t + 1$, combines valid partial signatures into a compact threshold certificate $\sigma$.
- $\mathsf{ver}(\mathsf{pk}, m, \sigma) \to \{0, 1\}$: verifies that at least $t + 1$ authorized processes signed $m$, without revealing which subset $G$ participated.

The scheme satisfies standard correctness, unforgeability, and robustness properties [38, §4.3]. In particular, no PPT adversary controlling at most $t$ processes can produce a valid threshold certificate on behalf of honest parties that did not collectively authorize the message.

In our setting, the certified message is typically

$$m^\star = (\text{Accuse}, p_a),$$

meaning that some quorum of at least $t + 1$ processes accused process $p_a$. Contrary to aggregate- or multi-signatures, the verifier learns only that a qualified subset signed, not the identity of the subset itself.

A standard instantiation is Boldyreva's threshold BLS signature scheme [15, 38], where certificates remain of size $O(\kappa)$ and final verification is identical to ordinary BLS verification (essentially two pairings).

The leader-election construction [19] additionally requires *uniqueness*: except with negligible probability, for every message $m$ and every two subsets $G, G' \subseteq [n]$ with $|G| = |G'| = t + 1$, if all partial signatures are honestly generated, then $\mathsf{cb}\big(m, \{(\mathsf{pk}_i, \sigma_i)\}_{p_i\in G}\big) = \mathsf{cb}\big(m, \{(\mathsf{pk}_j, \sigma_j)\}_{p_j\in G'}\big)$.

A setup for a unique $(t, n)$-threshold signature scheme with $n = 2t + 1$ can be obtained from the GRandLine framework of [12]. Concretely, under a complete setup assumption (e.g., bulletin-board PKI together with a CRS), and standard pairing-based assumptions in the AGM and ROM, their protocol provides an adaptively secure distributed key generation with $O(\kappa n^2 \log n)$ communication, after which threshold signatures can be generated non-interactively with $O(\kappa n^2)$ communication per signing instance and constant expected latency. The resulting threshold signatures are unique, making them directly suitable for common-coin constructions.

***Cryptography for Extension protocols.*** Protocols manipulating long $L$-bits values typically rely on erasure coding proof system to avoid $Ln^2$ communication terms. We recall the corresponding primitives.

**Erasure codes.** Fix $n, t$. An $\big((n - t), t\big)$*-erasure code* is a pair of polynomial-time algorithms (encode, decode) such that:

- $\mathsf{encode}(v)$ outputs an ordered vector of $n$ coded symbols $\boldsymbol{d} = [d_1, \ldots, d_n]$;
- for every set $J \subseteq [n]$ of size $|J| = n - t$, if $\boldsymbol{d} = \mathsf{encode}(v)$, then $\mathsf{decode}\big(\{(j, d_j)\}_{j\in J}\big) = v$.

In other words, any set of $n - t$ correctly indexed symbols suffices to reconstruct the encoded value.

**ECP.** An *erasure coding proof system* (ECP) [7, 8] for an $(n, t)$-erasure code (encode, decode) is a tuple (ecsetup, ecprove, ecverify), such that:

- $\mathsf{ecsetup}(1^\kappa) \to \mathsf{pp}$ is an optional setup algorithm;
- $\mathsf{ecprove}(v) \to (c, \boldsymbol{d}, \boldsymbol{\pi})$ outputs
  - a constant-size commitment $c$,

  - a vector of coded symbols $\boldsymbol{d} = [d_1, \ldots, d_n]$,
  - a vector of witnesses $\boldsymbol{\pi} = [\pi_1, \ldots, \pi_n]$;
- $\mathsf{ecverify}(c, j, d_j, \pi_j) \in \{0, 1\}$ verifies that $d_j$ is the $j$-th coded symbol associated with commitment $c$.

The scheme satisfies the following properties, except with negligible probability.

- *Completeness.* If $(c, \boldsymbol{d}, \boldsymbol{\pi}) \leftarrow \mathsf{ecprove}(v)$, then $\boldsymbol{d} = \mathsf{encode}(v)$, and for every $j \in [n]$, $\mathsf{ecverify}(c, j, d_j, \pi_j) = 1$.
- *Correct reconstruction.* If $(c, \boldsymbol{d}, \boldsymbol{\pi}) \leftarrow \mathsf{ecprove}(v)$, and $J \subseteq [n]$ has size $n-t$, then $\mathsf{decode}\big(\{(j, d_j)\}_{j\in J}\big) = v$. This is just completeness together with correctness of the underlying erasure code.
- *Binding.* No PPT adversary can produce a commitment $c$ and two indexed collections

$$S = \{(j, d_j, \pi_j)\}_{j\in J}, \qquad S' = \{(j, d'_j, \pi'_j)\}_{j\in J'},$$

with $|J| = |J'| = n - t$, such that all fragments in both collections verify with respect to $c$, but

$$\mathsf{decode}\big(\{(j, d_j)\}_{j\in J}\big) \neq \mathsf{decode}\big(\{(j, d'_j)\}_{j\in J'}\big).$$

The last property is the binding property needed by the extension protocols: once the agreement layer fixes $c$, any $n - t$ valid fragments for $c$ reconstruct the same value. It does not say that valid fragments must have been honestly generated. A Byzantine process may produce a new commitment and valid fragments for some value; the point is that, after $c$ is fixed, it cannot make different honest processes reconstruct different values from valid fragments for $c$.

**Setup for ECP.** The ECP requires a succinct binding commitment to a vector $\vec{\phi} = (\phi_1, \ldots, \phi_n)$ supporting succinct proofs that a value $\phi_i$ is the $i$-th coordinate of the committed vector.

In our application, vectors are promised to be Reed–Solomon codewords $(f(\alpha_1), \ldots, f(\alpha_n))$ with $\deg(f) \le n - t - 1$, but this low-degree validity is certified externally (via PoD / threshold attestation) rather than by the commitment primitive itself. Hence the appropriate abstraction is a vector commitment for polynomial-evaluation vectors.

Under the collision-resistance assumption for a hash function hash, such a vector commitment can be instantiated trivially via a Merkle tree with $n$ leaves, where leaf $i$ contains $(i, \phi_i)$. The commitment digest is the Merkle root, and an opening witness for coordinate $i$ is the corresponding Merkle path. This yields $O(\kappa)$-size commitments and $O(\kappa \log n)$-bit witnesses (more precisely, $\Theta(\log n)$ hash values of $\kappa$ bits each), with $O(\log n)$ verification time.

Alternatively, both the commitment and each opening witness can be reduced to $O(\kappa)$ bits by using pairing-based bilinear accumulators [73] / polynomial-commitment techniques [65]. Concretely, one uses a structured reference string of powers-of-tau form $(g, g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^d})$, for some secret $\tau$ and degree bound $d \in \Theta(n)$, which allows committing succinctly to a vector (viewed as evaluations of a degree-$d$ polynomial) and proving openings at any coordinate with constant-size witnesses. Security relies on the hardness of the $q$-Strong Diffie–Hellman assumption ($q$-SDH), with $q = \Theta(d)$, in the underlying bilinear groups.

When $n > 3t$, such a powers-of-tau setup can be realized efficiently using the asynchronous protocol of [39], which achieves $O(\log n)$ expected rounds and $O(\kappa n^2 \log n)$ expected communication.

For a powers-of-tau setup in the regime $n > 2t$, one may use the round-reducing compiler of [30], which gives guaranteed output delivery even against $n - 1$ corruptions and reduces the sequential broadcast complexity of the round-robin PoT ceremony to $O(\sqrt{n})$ rounds generically, or $O(\sqrt{n} \log d)$ rounds for their concrete PoT-specific construction, in the $F_{\mathsf{CRS}}$-hybrid model. Alternatively, LitePoT [72] gives a permissionless blockchain-mediated ceremony tolerating $m - 1$ corrupt contributors; its contribution is orthogonal, reducing on-chain verification cost from $O(md)$ to $O(d)$ for $m$ batched contributions, with optimistic updates and succinct fraud proofs.

In our model, the blockchain/bulletin-board abstraction used in powers-of-tau ceremonies can be instantiated by Byzantine broadcast, equivalently by interactive consistency. Thus, we can run the corresponding PoT setup protocols using our IC abstraction. Initially this costs authentication parameter $\kappa_a = \kappa \log n$. Once the PoT string has been generated, it can be used to instantiate succinct accumulators, which in turn reduce the authentication parameter to $\kappa_a = \kappa$. This makes the PoT setup a one-time bootstrapping step: we first pay the larger IC cost to generate the structured reference string, and then use the resulting accumulators to obtain the desired $O(\kappa)$-size authentication objects thereafter.

## B Reduction to one VBA instance

This section isolates the honest-majority extension step used by our main construction. Its goal is twofold. First, it shows how to reduce Interactive Consistency to a single short-input instance of validated Byzantine agreement. Second, as an intermediate step, it gives a compact reduction from Byzantine Broadcast to validated Byzantine agreement.

The structure follows the standard Disperse–Agree–Retrieve paradigm [25]: first obtain a short digest together with a proof that the corresponding long preimage is safe to agree on; then run one instance of VBA on this digest/proof pair; finally retrieve the corresponding long preimage. Most of this machinery is standard. The retrieve phase is the usual erasure-code-based double-echo retrieval, and the DARE decomposition is by now classical. The point of this appendix is the precise dispersion interface needed here. In the case of source-validity (for BB), the proof of dispersal must certify either an ordinary digest whose preimage was observed by at least one honest process, or an explicit certified fault/default value for a non-responsive or equivocating source. In the case of vector-validity (for IC), the proof of dispersal must certify an ordinary digest whose preimage $\vec{v}$ was observed by at least one honest process (who was additionally able to locally check the vector-validity of $\vec{v}$).

We write the construction abstractly as

$$\mathsf{Agree}_{\mathsf{val}} = \mathsf{Disperse}_{\mathsf{val}} \rightarrow \mathsf{VBA} \rightarrow \mathsf{Retrieve}.$$

Here Disperse produces a transferable digest of a valid output candidate, together with a proof of dispersal. The parties then use one instance of validated Byzantine agreement to agree on a single

**Module 4** Retrieve

**Parameters:**
- A collision-free digest function Digest : $\mathcal{V}_O \to$ DigestValue.

**Events:**
- *request* reconstruct($c \in$ DigestValue, $v \in \mathcal{V}_O \cup \{\perp\}$).
- *indication* output($v' \in \mathcal{V}_O$).

**Assumed behavior:**
- $\exists v^* \in \mathcal{V}_O$, such that at least one honest process invokes reconstruct$(*, v^*)$ with $v^* \neq \perp$, and no honest process invokes reconstruct$(c, v)$ with $c \neq$ Digest$(v^*)$, or $v \neq \perp$ and $v \neq v^*$
- No simultaneity is required w.r.t. inputs

**Properties:**
- *Termination:* every honest process eventually outputs a value.
- *Stability:* No honest process output a value different from $v^*$

digest/proof pair. Finally, Retrieve reconstructs the corresponding value from the parties that actually observed it.

A useful feature of the presentation is that the IC-oriented dispersion layer is itself built from the BB-oriented one. Thus the only agreement-type subroutine used by the final extension is the single VBA instance above.

We first describe the digest layer and the two auxiliary protocols, and then prove that their composition yields the desired agreement task.

*Digest layer.* Fix an $(n, t)$-erasure code (encode, decode) and an erasure-coded polynomial-commitment primitive

$$(\text{ecsetup}, \text{ecprove}, \text{ecverify}).$$

We write

$$\text{Digest} : \mathcal{V}_O \to \text{DigestValue}$$

for the digest induced by the commitment returned by ecprove. Concretely, for every ordinary value $v \in \mathcal{V}_O$,

$$(c, \boldsymbol{d}, \boldsymbol{\pi}) \leftarrow \text{ecprove}(v) \qquad \text{and} \qquad \text{Digest}(v) := c.$$

In the BB-oriented dispersion layer, we additionally use the distinguished symbol $\perp^{\oplus}$ for a provably faulty source, with

$$\text{Digest}(\perp^{\oplus}) := \perp^{\oplus}.$$

## B.1 Retrieve

We first specify retrieval (see Module 4) independently of how the digest was obtained. The retrieval interface assumes that all honest processes reconstruct the same target digest $c$, and that at least one honest process knows a matching preimage.

The Retrieve's implementation is given in Algorithm 4.

*High-level description.* When process $p_i$ receives the request reconstruct$(c_{\text{in}}, v_{\text{in}})$, it records the target digest $c_i$ and marks the procedure as started (lines 6–7). If it already knows a preimage $v_{\text{in}} \neq \perp$, it computes an erasure-coded commitment/proof decomposition $(c, \boldsymbol{d}, \boldsymbol{\pi})$ (line 9) and sends fragment $(d_j, \pi_j)$ to each process $p_j$ (line 11). Upon receiving for the first time a valid fragment consistent with its target digest, a process echoes it to all parties (lines 14–15). Thereafter, every valid echoed fragment for the same digest is stored once (line 18); as soon as $n - t$ such fragments have been collected, the process decodes and outputs the value (line 20).

Lemma B.1. *Algorithm 4 implements Module 4. Moreover, Algorithm 4 terminates in two rounds after the reconstruction request, every point-to-point message has size $O(L_o/n + \kappa_a)$.*

**Algorithm 4** Retrieve$(c_{\text{in}}, v_{\text{in}})$

**Local variables at process $p_i$:**
1: $c_i \leftarrow \perp$
2: $started_i \leftarrow$ false
3: $sentEcho_i \leftarrow$ false
4: $\mathcal{S}_i \leftarrow \emptyset$ ▷ set of stored indexed symbols $(j, d_j)$

5: **upon** request reconstruct$(c_{\text{in}}, v_{\text{in}})$
6: $started_i \leftarrow$ true
7: $c_i \leftarrow c_{\text{in}}$
8: **if** $v_{\text{in}} \neq \perp$ **then**
9: $(c, \boldsymbol{d}, \boldsymbol{\pi}) \leftarrow \text{ecprove}(v_{\text{in}})$ ▷ assert $c = c_i$
10: **for** every $j \in [n]$ **do**
11: send (DISPERSE, $c, d_j, \pi_j$) to $p_j$

12: **upon** reception of (DISPERSE, $c, d, \pi$) from $p_j \wedge started_i$
13: **if** $c == c_i \wedge \text{ecverify}(c, i, d, \pi) \wedge \neg sentEcho_i$ **then**
14: $sentEcho_i \leftarrow$ true
15: broadcast (RECONSTRUCT, $c, d, \pi$)

16: **upon** reception of (RECONSTRUCT, $c, d, \pi$) from $p_j \wedge started_i$
17: **if** $c == c_i \wedge \text{ecverify}(c, j, d, \pi) \wedge (j, \cdot) \notin \mathcal{S}_i$ **then**
18: $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{(j, d)\}$
19: **if** $|\mathcal{S}_i| = n - t$ **then**
20: indicate output(decode$(\mathcal{S}_i)$)

Proof. We prove termination and stability.

*Stability.* Let $v^*$ be the distinguished value from the assumed behavior of Module 4. By assumption, at least one honest process invokes reconstruct$(*, v^*)$ with $v^* \neq \perp$, and no honest process invokes reconstruct$(c, v)$ with either $c \neq$ Digest$(v^*)$ or $v \neq \perp, v^*$.

Hence every honest process starts with the same target digest

$$c^* := \text{Digest}(v^*).$$

Any honest process that knows $v^*$ computes fragments of *that same* value at line 9, and sends only fragments that verify against $c^*$ at line 11. Therefore every valid echoed fragment that can be stored by an honest process is a fragment of $v^*$ consistent with digest $c^*$. Since the digest is collision-free and the erasure code is correct, once an honest process accumulates $n - t$ such fragments, decode$(\mathcal{S}_i)$ at line 20 can only return $v^*$. Thus no honest process outputs a value different from $v^*$.

*Termination.* Fix an honest process $p_h$ that invokes reconstruct$(*, v^*)$ with $v^* \neq \perp$. By lines 9–11, $p_h$ sends to every honest process a valid fragment for digest $c^*$. Hence every honest process eventually receives at least one valid message (DISPERSE, $c^*, d, \pi$) matching its local target digest, and therefore broadcasts an echo at lines 14–15. Since there are at least $n - t$ honest processes, every honest process eventually receives $n - t$ valid echoed fragments for the same digest $c^*$ (by ECP completeness), stores them at line 18, and outputs at line 20. This proves termination.

*Complexity.* The protocol has two communication steps: holders of a preimage send one erasure-coded fragment and proof to each process, and each process echoes at most one valid fragment/proof for the target digest. Each such message has size $O(L_o/n + \kappa_a)$. □

*A useful special case: balanced broadcast.* The same pattern yields the two-round balanced-broadcast primitive shown in Algorithm 5: the source first disperses one fragment per party (lines 5–8); every recipient who validates its own fragment rebroadcasts it once and adopts the corresponding digest (lines 11–12); once $n - t$ echoed

**Algorithm 5** BalancedBroadcast($v_{\text{in}}$)

**Local variables at process $p_i$:**
1: $c_i \leftarrow \perp$
2: $sentEcho_i \leftarrow$ false
3: $\mathcal{S}_i \leftarrow \emptyset$ ▷ set of stored indexed symbols $(j, d_j)$

4: **upon** request broadcast($v_{\text{in}}$) by the source $p_s$
5: $(c, \boldsymbol{d}, \boldsymbol{\pi}) \leftarrow$ ecprove($v_{\text{in}}$)
6: $c_i \leftarrow c$
7: **for** every $j \in [n]$ **do**
8: send (DISPERSE, $c, d_j, \pi_j$) to $p_j$

9: **upon** reception of (DISPERSE, $c, d, \pi$) from the source $p_s$ for the first time
10: **if** ecverify($c, i, d, \pi$) **then**
11: broadcast (RECONSTRUCT, $c, d, \pi$)
12: $c_i \leftarrow c$

13: **upon** reception of (RECONSTRUCT, $c, d, \pi$) from $p_j \wedge c_i \neq \perp$
14: **if** $c = c_i \wedge$ ecverify($c, j, d, \pi$) $\wedge (j, \cdot) \notin \mathcal{S}_i$ **then**
15: $\mathcal{S}_i \leftarrow \mathcal{S}_i \cup \{(j, d)\}$
16: **if** $|\mathcal{S}_i| = n - t$ **then**
17: indicate output(decode($\mathcal{S}_i$))

**Module 5** Disperse

**Parameters:**
- A validity property [27] val : $\mathcal{I}(\mathcal{V}_I) \to 2^{\mathcal{V}_O} \setminus \{\emptyset\}$, where each input configuration $c_I \in \mathcal{I}(\mathcal{V}_I)$ is a partial map from honest processes to submitted inputs in $\mathcal{V}_I$.
- A collision-free digest function Digest : $\mathcal{V}_O \to$ DigestValue.
- A proof domain $\Pi_{\text{PoD}}$.
- A predicate PoDValid : DigestValue $\times \Pi_{\text{PoD}} \to \{\text{true}, \text{false}\}$.
- In the BB-oriented case, a distinguished symbol $\perp^{\oplus} \in$ DigestValue $\cap \mathcal{V}_O$ associated with the default faulty-source branch.

**Events:**
- *request* disseminate($v \in \mathcal{V}_I$).
- *notification* observe($v \in \mathcal{V}_O$).
- *indication* acquire($c \in$ DigestValue, $\pi \in \Pi_{\text{PoD}}, u \in \mathcal{V}_O \cup \{\perp\}$).

**Properties:**
- *Integrity:* if an honest process acquires $(c, \pi, *)$, then PoDValid($c, \pi$) = true.
- *Validity:* Whenever an honest process observes $v$, we have $v \in \text{val}(c_I)$, where $c_I$ is the input configuration of the processes.
- *Retrievability:* no PPT adversary can produce a pair $(c, \pi)$, such that PoDValid($c, \pi$) = true, unless some honest process has previously observed a value $v \in \mathcal{V}_O$ such that Digest($v$) = $c$.
- *Termination:*
  - Case val = $\text{val}_{\text{IC}}$: all honest processes eventually acquire within one round of one another.
  - Case val = $\text{val}^*_{\text{BB}}$: all honest processes simultaneously acquire after a constant number of rounds.

fragments are gathered, the value is decoded and delivered (lines 15–17). We will use this balanced-broadcast subroutine inside the BB-oriented implementation of DISPERSE.

## B.2 Disperse

The non-trivial part is therefore not retrieval, but dispersion. A dispersion protocol must create the condition required by retrieval while communicating only a short digest to the agreement layer. For an ordinary digest $c$, its proof of dispersal must imply that at least one honest process observed a matching preimage. In the case of source-validity (for BB), for a faulty source, the protocol may instead produce a certified default value, so that the subsequent VBA instance still has a valid input after a fixed number of rounds. Disperse is specified in Module 5.

**Algorithm 6** Disperse$^{\text{BB}}_{p_\ell}$($v_{\text{in}}$)

**Local variables at process $p_i$:**
1: $v_i \leftarrow \perp$
2: $c_i \leftarrow \perp$
3: $acquired_i \leftarrow$ false

4: **upon** request disseminate($v_{\text{in}}$) by the source $p_\ell$
5: invoke BalancedBroadcast($v_{\text{in}}$) with source $p_\ell$

6: **upon** indication output($v \in \mathcal{V}_O^{\text{BB}}$) from BalancedBroadcast
7: $v_i \leftarrow v$
8: $c_i \leftarrow$ digest($v$)
9: indicate observe($v_i$)

**Round 3:**
10: **if** $v_i \neq \perp$ **then**
11: broadcast (ACK, $c_i$, ShareSign$_i^{t+1}(c_i)$)

**Round 4:**
12: **if** there exists $c \in$ DigestValue and a set $J \subseteq [n]$ of size $t+1$ such that, for every $j \in J$, a message (ACK, $c$, $psig_j$) was received by the end of Round 3 and ShareVerify$_j^{t+1}(c, psig_j)$ = true **then**
13: $\sigma_c \leftarrow \text{Combine}^{t+1}(\{(j, psig_j)\}_{j \in J})$
14: broadcast (CERTIFY, $c, \sigma_c$)
15: **else**
16: indicate observe($\perp^{\oplus}$)
17: broadcast (ACK, $\perp^{\oplus}$, ShareSign$_i^{t+1}(\perp^{\oplus})$)

**End of Round 4:**
18: **if** there exists $c \in$ DigestValue $\cup \{\perp^{\oplus}\}$ and a set $J \subseteq [n]$ of size $t+1$ such that, for every $j \in J$, either **then**
(a) a message (ACK, $c$, $psig_j$) was received by the end of Round 4 and ShareVerify$_j^{t+1}(c, psig_j)$ = true,
**or**
(b) a message (CERTIFY, $c, \sigma_c$) was received by the end of Round 4 and CombinedVerify$^{t+1}(c, \sigma_c)$ = true
19: **if** case (a) holds **then**
20: $\sigma_c \leftarrow \text{Combine}^{t+1}(\{(j, psig_j)\}_{j \in J})$
21: **if** $c = \perp^{\oplus}$ **then**
22: $u_i \leftarrow \perp^{\oplus}$
23: **else if** $v_i \neq \perp \wedge$ digest($v_i$) = $c$ **then**
24: $u_i \leftarrow v_i$
25: **else**
26: $u_i \leftarrow \perp$
27: $acquired_i \leftarrow$ true
28: indicate acquire($c, \sigma_c, u_i$)

*B.2.1 Implementation for BB.* We first instantiate DISPERSE for the BB validity predicate $\text{val}^*_{\text{BB}}$. A Byzantine source may be silent, in which case no ordinary digest is available. Algorithm 6 therefore certifies either an ordinary digest, or the distinguished value $\perp^{\oplus}$.

*High-level description.* The protocol has four rounds. In the first two rounds, the source $p_\ell$ invokes balanced broadcast on its value (line 5); processes that successfully decode store the resulting BB-output value $v_i$ and compute its digest $c_i$ (lines 7–8). In Round 3, every process that obtained a value broadcasts a threshold-signature share acknowledging its digest (line 11). In Round 4, if a process has seen $t+1$ valid shares for some common digest $c$, it combines them into a threshold certificate $\sigma_c$ and rebroadcasts it (lines 13–14); otherwise, it signs the accusation symbol $\perp^{\oplus}$ (line 17). At the end of Round 4, as soon as a process has enough evidence for some common $c$—either directly as $t+1$ shares or through a certified threshold signature—it locally reconstructs $\sigma_c$, sets its local observed value $u_i$ to the matching decoded value, to $\perp$, or to $\perp^{\oplus}$ according to the case (lines 22–26), and triggers acquire($c, \sigma_c, u_i$) (lines 27–28).

The explicit default branch in Algorithm 6 is included to make the dispersion robust to both non-responsive and equivocating sources. This corrects a minor but important flaw in the ECP-based broadcast protocol of [7], where the sender is supposed to first collect signature shares on an ECP commitment and then forward a combined threshold signature, which is used as the input to the subsequent VBA instance. However, if the sender stops before forwarding such a certificate, honest processes may have no valid value to propose to VBA. Our dispersion interface makes this choice explicit. A pair $(c, \pi)$ can satisfy $\mathsf{PoDValid}(c, \pi)$ not only if $c$ is an ordinary digest, in which case the proof certifies that at least one honest process observed a matching preimage, but also if $c = \perp^{\oplus}$, in which case the proof is a threshold certificate for the designated faulty-source value. Thus a silent source still leads, after a fixed number of rounds, to a valid value for the subsequent validated agreement instance.

Lemma B.2. *Algorithm 6 implements Module 5 for* $\mathsf{val} = \mathsf{val}^{*}_{\mathrm{BB}}$. *Moreover, Algorithm 6 has fixed latency four rounds, every point-to-point message has size* $O(L_o/n + \kappa_a)$.

Proof. We verify integrity, validity, retrievability, and termination.

*Integrity.* Every acquisition at line 28 is associated with a threshold certificate $\sigma_c$, obtained either by combining $t + 1$ valid shares at line 20, or by receiving an already valid threshold certificate. Hence $\mathsf{PoDValid}(c, \sigma_c) = \mathsf{true}$.

*Validity.* An honest process observes an ordinary value only after it is delivered by BalancedBroadcast, at line 9. If the source is honest, correctness of BalancedBroadcast implies that this value is the source input. If the source is faulty, any delivered value is admissible for $\mathsf{val}^{*}_{\mathrm{BB}}$. Thus every ordinary observed value is valid. An honest process observes $\perp^{\oplus}$ only in the accusation branch at line 16; this branch is admissible only as the certified faulty-source default.

*Retrievability.* Let $c \neq \perp^{\oplus}$, and suppose $\mathsf{PoDValid}(c, \sigma_c) = \mathsf{true}$. Then $\sigma_c$ is a valid $(t + 1)$-threshold signature on $c$. By threshold-signature unforgeability, except with negligible probability, this certificate contains at least one share produced by an honest process. An honest process signs an ordinary digest $c$ only after delivering a value $v$ from BalancedBroadcast, setting $c = \mathsf{Digest}(v)$, and observing $v$ at line 9. Thus every valid ordinary proof of dispersal is backed by an observed preimage at some honest process.

A valid proof for $\perp^{\oplus}$ is a valid $(t + 1)$-threshold signature on $\perp^{\oplus}$. Again, by threshold-signature unforgeability, at least one honest process must have signed $\perp^{\oplus}$. Honest processes do so only in the accusation branch, after observing $\perp^{\oplus}$ at line 16. If the source is honest, the balanced-broadcast phase delivers the source value to every honest process, so honest processes do not enter this branch. Hence a valid certificate for $\perp^{\oplus}$ can only arise in executions where the default/faulty-source branch is admissible.

*Termination.* The protocol has fixed constant duration. If the balanced-broadcast phase delivers some value to one honest process, then all honest processes deliver the same value within the same two-round window. Hence at least $n - t \geq t + 1$ honest processes broadcast acknowledgment shares for the same digest in Round 3, and every honest process can certify and acquire that digest by the end of Round 4.

Otherwise, no honest process delivers a value by the end of the balanced-broadcast phase. Then every honest process follows the accusation branch at line 17. Since $n - t \geq t + 1$, every honest process receives enough shares for $\perp^{\oplus}$ by the end of Round 4 and acquires the certified default branch. Thus all honest processes simultaneously acquire after a constant number of 4 rounds.

*Complexity.* The balanced-broadcast phase takes two rounds and uses messages of size $O(L_o/n + \kappa_a)$. The following two rounds contain only threshold signature shares or threshold certificates on digests, of size $O(\kappa)$. Thus every point-to-point message in Algorithm 6 has size $O(L_o/n + \kappa_a)$. □

*B.2.2 Implementation for IC.* We now lift the Byzantine Broadcast dispersion layer to Interactive Consistency. The leader first constructs a verifiable vector using LearnOrExpose. Once such a vector is available, it is just a long value whose validity is publicly checkable. Therefore the BB-oriented dispersion layer can be used as a black box to disperse its digest. The only additional step is an echo of acquisitions: if one honest process obtains a valid IC proof-of-dispersal, all honest processes must learn some valid digest/proof pair within one further round, so that the IC version of Disperse satisfies the required acquisition-simultaneity.

*High-level description.* Each process first broadcasts its signed input disclosure upon $\mathsf{disseminate}(m_i)$ (line 16) and starts the common timer (line 17). Every valid first disclosure from process $P_j$ is stored in the local candidate vector and signature table (lines 19–20). Then the protocol enters the same pipelined view structure as the generic compiler: whenever the timer reaches the start of a new view $v$, parties invoke leader election (line 22); once the elected leader $\ell$ is learned, it is stored and LearnOrExpose is started on the current candidate state (lines 24–25). If process $p_i$ is the elected leader of view $v$, it feeds the verifiable vector returned by LearnOrExpose to the BB-oriented dispersal instance $\mathrm{Disp}^{\mathrm{BB}}_{v}$ (line 28); non-leaders simply invoke that instance with $\perp$ (line 30). Whenever some BB-dispersal instance acquires a non-faulty digest $c \neq \perp^{\oplus}$, the process triggers a local helper $\mathsf{acquireAndEcho}(c, \pi, x)$ (line 33); and whenever it receives an echoed acquisition message with a valid IC-proof-of-dispersal, it triggers the same helper with local value $\perp$ (the second acquisition handler). This helper marks the process as acquired, rebroadcasts the acquisition, and triggers the external event $\mathbf{acquire}(c, \pi, x)$.

The role of the final echo is precisely to enforce the one-round acquisition-simultaneity required by the IC version of Disperse: once one honest process acquires a valid digest/proof pair, all honest processes receive the echoed acquisition within one further round and acquire as well.

Lemma B.3. *Algorithm 7 implements Module 5 for* $\mathsf{val} = \mathsf{val}_{\mathrm{IC}}$. *Moreover, Algorithm 7 has* $O(1)$ *expected acquisition latency. Until a halt time* $\Delta_{\mathrm{halt}}$ *the per-process communication is* $O((L_o + n\kappa_a)\Delta_{\mathrm{halt}}))$.

Proof. We prove integrity, validity, retrievability, and termination.

*Integrity.* An honest process triggers $\mathsf{acquire}(c, \pi, *)$ only through $\mathsf{acquireAndEcho}(c, \pi, *)$. This helper is invoked in two cases. First,

**Algorithm 7** Disperse$^{\mathrm{IC}}$

**Black-box subprotocols:**
1: VerifyVec$^{in}$, VerifyVec$^{out}$ : $(\mathcal{V}_I^{\mathrm{IC}} \cup \{\perp\})^n \times \{0,1\}^* \to \{\text{true}, \text{false}\}$. defined as in Section 5.2
2: $\mathcal{V}_I^{\mathrm{BB}} = \mathcal{V}_O^{\mathrm{BB}} = \{(\vec{v}, \pi) \mid \mathrm{VerifyVec}^{\mathrm{out}}(\mathsf{v}, \pi) = \mathit{true}\}$
3: ValidPoD$^{\mathrm{BB}}$, ValidPoD$^{\mathrm{IC}}$ : DigestValue $\times \{0,1\}^* \to \{\text{true}, \text{false}\}$.
4: ValidPoD$^{\mathrm{IC}}(\perp^{\oplus}, *) = \mathit{false}$, ValidPoD$^{\mathrm{IC}}(c \neq \perp^{\oplus}, \pi) = \mathrm{ValidPoD}^{\mathrm{BB}}(c, \pi)$.
5: LeaderElection,
6: LearnOrExpose, , parametrized with (VerifyVec$^{in}$, VerifyVec$^{out}$)
7: Disp$^{\mathrm{BB}}$ , parametrized with ($\mathcal{V}_I^{\mathrm{BB}}$, $\mathcal{V}_O^{\mathrm{BB}}$, ValidPoD$^{\mathrm{BB}}$)

**Parameters:**
8: $c \leftarrow \mathsf{Lat}_{\mathsf{LE}} + \mathsf{Lat}_{\mathsf{LoE}} + \mathsf{Lat}_{\mathsf{Disp(BB)}}$ ▷ a new view starts every $c = O(1)$ rounds

**Local state at process $P_i$:**
9: *acquired* ← false
10: **for** $j \in [n]$ **do**
11: *candVec*$[j] \leftarrow \perp$
12: *candSign*$[j] \leftarrow \perp$
13: **for** $v \in \mathbb{N}_{\geq 1}$ **do**
14: *leader*$[v] \leftarrow \perp$
15: **upon** Disp$^{\mathrm{IC}}$.disseminate$(m_i)$
16: send $(\langle \text{disclose}, m_i \rangle, \mathrm{Sign}_{P_i}(\langle \text{disclose}, m_i \rangle))$ to all parties
17: start timer
18: **upon** receiving for the 1st time a valid message $(\langle \text{disclose}, x_j \rangle, \sigma_j)$ from $P_j$
19: *candVec*$[j] \leftarrow x_j$
20: *candSign*$[j] \leftarrow \sigma_j$
21: **upon** timer reaches time $1 + (v-1) \cdot c$ for some view $v \geq 1$
22: request LeaderElection$_v$.inquire$(v)$
23: **upon** LeaderElection$_v$ informs $(\ell)$
24: *leader*$[v] \leftarrow \ell$
25: request LearnOrExpose$_v$.start(*candVec*, *candSign*)
26: **upon** LearnOrExpose$_v$ returns $(\tilde{vec}, \pi)$
27: **if** $i =$ *leader*$[v]$ **then**
28: Disp$_v^{\mathrm{BB}}$.disseminate$((\tilde{vec}, \pi))$
29: **else**
30: Disp$_v^{\mathrm{BB}}$.disseminate$(\perp)$
31: **upon** Disp$_v^{\mathrm{BB}}$.observe$(x)$ with $x \neq \perp^{\oplus}$
32: indicate observe$(x)$
33: **upon** Disp$_v^{\mathrm{BB}}$.acquire$(c, \pi, x) \wedge c \neq \perp^{\oplus}$, while $\neg$*acquired*
34: trigger acquireAndEcho$(c, \pi, x)$
35: **upon** receiving $\langle$Acquisition$, c, \pi\rangle$ with ValidPoD$^{\mathrm{IC}}(c, \pi)$, while $\neg$*acquired*
36: trigger acquireAndEcho$(c, \pi, \perp)$
37: **upon** triggering acquireAndEcho$(c, \pi, x)$
38: *acquired* ← *true*
39: broadcast $\langle$Acquisition$, c, \pi\rangle$
40: trigger **acquire**$(c, \pi, x)$

it is invoked after some BB-oriented dispersion instance acquires a non-faulty digest/proof pair (line 33). By Lemma B.2, the acquired pair satisfies the BB proof-of-dispersal predicate. Second, it is invoked after receiving an echoed acquisition message whose proof passes the explicit PoDValid$^{\mathrm{IC}}$ test. Hence every IC acquisition by an honest process satisfies

$$\mathrm{PoDValid}^{\mathrm{IC}}(c, \pi) = \text{true}.$$

*Validity.* An honest process emits an IC observation only by forwarding an observation of some BB-oriented dispersion instance:

$$\mathrm{Disp}_v^{\mathrm{BB}}.\mathrm{observe}(x),$$

with $x \neq \perp^{\oplus}$, at line 31. By validity of the BB-oriented dispersion layer, $x$ is valid for the BB-layer validity predicate used in that instance.

In the IC construction, the ordinary BB-layer values are precisely verifiable vectors, i.e., values of the form $x = (\vec{v}, \rho)$ such that

$$\mathrm{VerifyVec}^{out}(\vec{v}, \rho) = \text{true}.$$

By output soundness of LearnOrExpose, every honest coordinate of $\vec{v}$ is equal to the corresponding honest input. Thus $\vec{v} \in \mathrm{val}_{\mathrm{IC}}(c_I)$. Since the IC observation is exactly such an ordinary BB observation, every value observed by an honest process in the IC dispersion layer is valid for $\mathrm{val}_{\mathrm{IC}}$.

*Retrievability.* Let a PPT adversary produce a pair $(c, \pi)$ such that

$$\mathrm{PoDValid}^{\mathrm{IC}}(c, \pi) = \text{true}.$$

By definition of the IC proof-of-dispersal predicate, the BB default symbol is not admissible at the IC layer, so $c \neq \perp^{\oplus}$. Moreover, $(c, \pi)$ must be a valid ordinary proof of dispersal for one of the underlying BB-oriented dispersion instances. Hence

$$\mathrm{PoDValid}^{\mathrm{BB}}(c, \pi) = \text{true}$$

for that instance.

By retrievability of the BB-oriented dispersion layer, except with negligible probability, some honest process previously observed a value $x \in \mathcal{V}_O^{\mathrm{BB}}$ such that

$$\mathrm{Digest}(x) = c.$$

Therefore the IC dispersion layer forwards this observation at line 31, and some honest process observes the same $x$ at the IC level. Thus every valid IC proof-of-dispersal is backed by an IC-level honest observation of a preimage with digest $c$.

*Termination.* Consider the first view whose elected leader is honest. Since all honest processes disclose their signed inputs at line 16, the honest leader eventually has enough candidate information. By progress of LearnOrExpose, it returns a verifiable vector in that view. As leader, it invokes the corresponding BB-oriented dispersion instance on that verifiable vector at line 28.

By Lemma B.2, that BB-oriented dispersion instance terminates after a constant number of rounds. Since its source is honest in the view considered here, it acquires an ordinary digest $c \neq \perp^{\oplus}$ at some honest process. That process triggers acquireAndEcho$(c, \pi, x)$, and broadcasts $\langle$Acquisition$, c, \pi\rangle$. Every honest process receives this echoed acquisition within one more round, verifies PoDValid$^{\mathrm{IC}}(c, \pi) =$ true, and triggers acquireAndEcho$(c, \pi, \perp)$. Hence all honest processes eventually acquire, and their acquisitions occur within one round of one another.

*Complexity.* Each view invokes LearnOrExpose, leader election, and one BB-oriented dispersion instance on a value of size $L_o$. By the complexity of Disperse$^{\mathrm{BB}}$, and by the balanced implementation of the auxiliary honest-majority subroutines, the per-process per-round communication complexity is $O(L_o + n\kappa_a)$. The number of views until the first honest leader is geometric with constant success probability, so the expected acquisition latency is $O(1)$.

The Disperse interface itself does not prescribe when a process halts after acquisition. If the surrounding protocol halts the dispersion layer after $\Delta_{\mathrm{halt}}$ rounds, then at most $O(\Delta_{\mathrm{halt}})$ views can be active or started. Hence the per-process communication is $O((L_o + n\kappa_a)\Delta_{\mathrm{halt}})$.

□

**Algorithm 8** $\mathsf{DARE}_{\mathsf{val}}(m_i)$ at process $p_i$

```
   Black-box subprotocols:
1:    Disperse_val, VBA, Retrieve
   Local variables:
2: submitted_i ← false
3: delivered_i ← false
4: obs_i[·] ← ⊥

5: upon request submit(m_i)
6:     request Disperse_val.disseminate(m_i)

7: upon notification Disperse_val.observe(v)
8:     obs_i[Digest(v)] ← v

9: upon indication Disperse_val.acquire(c, π, u)
10:    if submitted_i = false and delivered_i = false then
11:        request VBA.submit(c, π)
12:        submitted_i ← true

13: upon indication VBA.deliver(c*, π*)
14:    delivered_i ← true
15:    if c* = ⊥^⊕ then
16:        indicate deliver(⊥^⊕)
17:        wait Δ^VBA_sim
18:        request halt()
19:    else
20:        w_i ← obs_i[c*]
21:        request Retrieve.reconstruct(c*, w_i)

22: upon indication Retrieve.output(v*)
23:    indicate deliver(v*)
24:    wait Δ^VBA_sim
25:    request halt()
```

## B.3 DARE's instantiation

At this point, all long-message issues have been isolated in Disperse and Retrieve. The remaining agreement step is short: parties agree only on a pair $(c, \pi)$, where $c$ is a digest and $\pi$ is a proof of dispersal. The external-validity predicate of VBA is therefore exactly PoDValid. This is the reason one VBA instance (e.g., [3]) suffices.

We can now compose Disperse, one instance of validated Byzantine agreement, and Retrieve.

*Construction.* The composed protocol is given in Algorithm 8. Each process first invokes the appropriate dispersion layer on its input (line 6). The dispersion layer may then emit two kinds of events. Whenever it observes a value $v$, the process stores $v$ under its digest (line 8). Whenever it acquires a proof of dispersal $(c, \pi)$, the process submits the short pair $(c, \pi)$ to one instance of validated Byzantine agreement, unless it has already submitted such a pair (lines 10–12). The external-validity predicate of this VBA instance is

$$Q(c, \pi) \equiv \mathsf{PoDValid}(c, \pi).$$

Once VBA outputs a common pair $(c^*, \pi^*)$ (line 13), the process ignores subsequent acquisitions by setting $delivered_i = \text{true}$ (line 14). If $c^* = \perp^{\oplus}$, which only exists in the BB-oriented case, the process directly delivers $\perp^{\oplus}$ (line 16). Otherwise, it invokes Retrieve on the agreed digest $c^*$, using the locally stored preimage if it has one and $\perp$ otherwise (lines 20–21). The reconstructed value is then delivered (line 23).

Theorem B.4. *Assume that Disperse implements Module 5 for validity relation* val*, that VBA satisfies consistency, termination, and external validity for* $Q(c, \pi) \equiv \mathsf{PoDValid}(c, \pi)$*, and that Retrieve implements Module 4. Then Algorithm 8 solves the agreement task associated with* val*, except with negligible probability. Moreover, the latency* $Lat_{DARE}$ *of Algorithm 8 is* $O(Lat_{Disperse} + Lat_{\mathsf{VBA}} + Lat_{Retrieve})$*, while the per-process communication complexity* $\mathsf{ppcc}_{DARE}$ *of Algorithm 8 is* $O(\mathsf{ppcc}_{Disperse}(\Delta_{halt} := Lat_{DARE}) + \mathsf{ppcc}_{\mathsf{VBA}} + \mathsf{ppcc}_{Retrieve})$.

Proof. We condition on the good executions in which collision-freeness and the cryptographic unforgeability guarantees used by Disperse hold. The complementary event has negligible probability.

*Preconditions of VBA.* By termination of Disperse, every honest process eventually acquires some tuple $(c, \pi, u)$. Moreover, the acquisitions are near-simultaneous: simultaneous in the BB-oriented case, and within one round of one another in the IC-oriented case. Hence the VBA submissions at line 11 are near-simultaneous.

By integrity of Disperse, every tuple acquired by an honest process satisfies

$$\mathsf{PoDValid}(c, \pi) = \text{true}.$$

Therefore every honest process submits a valid input to VBA. Finally, no honest process halts before receiving the VBA output: before line 13, the only actions of an honest process are to store observations and possibly submit once to VBA. Thus the execution satisfies the preconditions under which the VBA properties are claimed.

*Consequences of VBA.* Let $(c^*, \pi^*)$ be the value delivered by VBA at line 13. By VBA consistency, all honest processes deliver the same pair $(c^*, \pi^*)$. By VBA external validity,

$$\mathsf{PoDValid}(c^*, \pi^*) = \text{true}.$$

By VBA termination, every honest process eventually reaches line 13. Since $\mathsf{PoDValid}(c^*, \pi^*) = \text{true}$, the retrievability property of Disperse implies that some honest process previously observed a value $v^*$ such that

$$\mathsf{Digest}(v^*) = c^*$$

. By validity of Disperse,

$$v^* \in \mathsf{val}(c_I).$$

Also, all honest processes take the same subsequent branch: either the default branch $c^* = \perp^{\oplus}$, or the retrieval branch $c^* \neq \perp^{\oplus}$.

Assume $c^* = \perp^{\oplus}$. All honest processes therefore deliver the same valid value $\perp^{\oplus}$ at line 16 and halt at line 18.

*Preconditions of Retrieve.* Assume $c^* \neq \perp^{\oplus}$. Since $\mathsf{PoDValid}(c^*, \pi^*) = \text{true}$, retrievability of Disperse implies that some honest process previously observed $v^*$. When this observation occurred, that process stored $v^*$ under digest $c^*$ at line 8. Hence, after VBA delivers $(c^*, \pi^*)$, at least one honest process invokes

$$\mathsf{Retrieve.reconstruct}(c^*, v^*)$$

at line 21.

Every other honest process invokes Retrieve with the same digest $c^*$ and with input either equal to a locally stored preimage for $c^*$, or equal to $\perp$, by line 20. Collision-freeness implies that no honest process can have stored a distinct value $v' \neq v^*$ under the same digest $c^*$. Therefore every honest process invokes Retrieve with either $v^*$ or $\perp$, and at least one honest process invokes it with $v^*$. The invocations are near-simultaneous because VBA deliveries are near-simultaneous. Moreover, no honest process halts before

Retrieve outputs in the retrieval branch. Hence the preconditions of Module 4 are satisfied with distinguished value $v^*$.

*Consequences of Retrieve.* By stability of Retrieve, no honest process outputs a value different from (valid) $v^*$. By termination of Retrieve, every honest process eventually outputs $v^*$, and Algorithm 8 then delivers (valid) $v^*$ at line 23 and halts at line 25.

*Complexity.* The protocol only sequences three components. First, it runs $\text{Disperse}_{\text{val}}$ until an acquisition provides a valid VBA input. Second, it runs one VBA instance on the short pair $(c, \pi)$. Third, unless the agreed digest is $\perp^{\oplus}$, it runs Retrieve on the agreed digest. Therefore the latency $\text{Lat}_{\text{DARE}}$ is at most $\text{Lat}_{\text{Disperse}} + O(\text{Lat}_{\text{VBA}}) + \text{Lat}_{\text{Retrieve}}$., while the per-process communication complexity is at most $\text{ppcc}_{\text{Disperse}}(\Delta_{halt}) + \text{ppcc}_{\text{VBA}} + \text{ppcc}_{\text{Retrieve}}$..

Finally, once VBA outputs, all honest processes either halt immediately in the default branch or halt after Retrieve outputs. Thus, when the subroutines provide the stated near-simultaneous termination guarantees, decision and halting have the same asymptotic latency, i.e. $\text{ppcc}_{\text{Disperse}}(\Delta_{halt}) = O(\text{ppcc}_{\text{Disperse}}(\Delta'_{halt} := \text{Lat}_{\text{DARE}}))$. □

*Specialization to BB and IC.* Applying Theorem B.4 with the BB-oriented dispersion layer of Lemma B.2 yields a reduction from one instance of validated Byzantine agreement to Byzantine Broadcast. Applying it with the IC-oriented dispersion layer of Lemma B.3 yields a reduction from one instance of validated Byzantine agreement to Interactive Consistency. In the honest-majority setting, this is the refined path that lets us use a single short-input VBA invocation, rather than first compiling VBA into BB and then applying the generic BB-based compiler.

Corollary B.5 (Balanced BB from one VBA instance). *In the honest-majority regime $t < n/2$, Byzantine Broadcast for output size $L_o$ can be solved with constant expected latency and balanced per-process communication $O(L_o + \kappa_a n)$, hence total communication $O(nL_o + \kappa_a n^2)$.*

Proof. The construction uses $\text{DARE}_{\text{val}^*_{\text{BB}}}$ with the BB-oriented dispersion layer of Lemma B.2, the retrieval protocol of Lemma B.1, and one short-input VBA instance instantiated by the balanced constant-expected-latency protocol of [3]. □

Corollary B.6 (Balanced IC from one VBA instance). *In the honest-majority regime $t < n/2$, Interactive Consistency for output size $L_o = nL_{in}$ can be solved with constant expected latency and balanced per-process communication $O(L_o + \kappa n)$, hence total communication $O(nL_o + \kappa n^2)$. For IC with n inputs of size $L_{\text{in}}$, this gives $O\big(n(L_{\text{in}} + \kappa)\big)$ per process, and $O\big(n^2(L_{\text{in}} + \kappa)\big)$ total communication.*

Proof. The construction uses $\text{DARE}_{\text{val}_{\text{IC}}}$ with the IC-oriented dispersion layer of Lemma B.3, the retrieval protocol of Lemma B.1, and one short-input VBA instance instantiated by [3]. □

# C Extended Related Work

This section discusses the extent to which our assumptions are inherent. Our protocols rely on synchrony, a bulletin-board public-key infrastructure (bb-PKI), and standard signatures secure against polynomial-time adversaries. We explain below how each assumption relates to prior work.

## C.1 Information-Theoretic Agreement

The classical impossibility threshold[8] $n \le 3t$, stems from the adversary's *ability to simulate* inconsistent views.

Protocols tolerating more than $n/3$ Byzantine faults therefore usually assume, as we do, both: (i) a computationally bounded adversary, and (ii) a bulletin-board setup allowing parties to publish the public keys used by a signature scheme that provides transferable authentication. That said, only one of these two assumptions is fundamentally necessary.

A first route keeps the setup assumption (ii), while dropping computational hardness. Using pseudo-signatures and private channels, one can emulate finite but sufficient transferability through purely information-theoretic means. This enables [75] statistically secure protocols beyond the $n/3$ barrier, both in honest majority [46] and in dishonest majority [23, 63]. These approaches, however, remain mostly of theoretical interest. Their cost is far from practical, with at least $\Omega(n^7)$ communication in the honest-majority construction of [46], and at least $\Omega(n^3)$ latency in the dishonest-majority broadcast-based constructions of [23, 63].

Our use of standard signatures should therefore be viewed primarily as an efficiency choice: it provides a simple and practical realization of transferable authentication, yielding negligible failure probability together with substantially better concrete complexity.

## C.2 Bootstrapping the bb-PKI from Resource Assumptions

The bb-PKI assumption can itself be weakened. Observe first that a bb-PKI is a special case of IC: each process inputs its public key, and the parties must agree on a vector mapping every honest process to its key. Thus any mechanism that bootstraps IC on public keys can be viewed as realizing the setup we assume.

For example, Andrychowicz and Dziembowski have shown [9] that bb-PKI/IC can be realized (without PKI assumption) if the honest parties have computing power that is some non-negligible fraction of computing power of the adversary. More broadly, a long line of work studies tradeoffs between efficiency, resilience, cryptographic assumptions, and weaker setup assumptions in permissionless and resource-restricted models [6, 22, 29, 52–56, 67].

## C.3 On the Need for Synchrony

Synchrony is a necessary for IC. In asynchronous networks, the closest analogue of IC is *agreement on a core set*, also known as asynchronous common subset (ACS) or vector consensus, where parties agree on a common set of up to $n - t$ contributed values. Constant-round ACS is achievable with private channels only [1]. Cubic-communication ACS is also known under hash-based assumptions alone [37, 81]. With transferable authentication, ACS can be reduced to validated Byzantine agreement with only additive quadratic overhead [70]. Such quadratic constructions typically rely on threshold signatures and common-coin mechanisms. The setup cost of this route has itself been reduced recently: Feng and Tang give asynchronous distributed key generation for unique threshold signatures from silent public setup [44].

[8] In fact, the impossibility of agreement with strong unanimity validity property [74] also holds for the weak validity property [45].

Finally, for any non-trivial validity property, even much weaker than IC or ACS, a network-agnostic protocol tolerating $t_s$ Byzantine failures in synchrony and $t_a$ Byzantine failures in asynchrony must satisfy $2t_s + t_a < n$, even with cryptographic setup [35].

### C.4 Implications for MPC

Robust MPC, i.e., MPC with guaranteed output delivery (GOD), relies on broadcast, and in its parallel form on IC/PBC. Recent protocols achieve GOD for a robustness threshold $t_r$ while preserving security, including privacy, up to $n - 1 - t_r$ corruptions, using IC as the communication primitive [62, 80]. Under suitable cryptographic assumptions, "round-optimal MPC" with GOD can be obtained from two consecutive IC calls [61], which motivated the study of "broadcast-optimal MPC" [31, 36]. This line of work also clarifies the necessity of parallel broadcast: in honest majority, IC is needed to obtain GOD in two "rounds" [36]. In dishonest majority, the analogous hierarchy gives identifiable/unanimous abort, depending on the available IC rounds [31].

## D On potential extensions for long inputs

The state of the art for extension protocols in dishonest majority incurs a linear $\Omega(t)$ latency overhead [71]. Recent works suggest constant-overhead extensions, either for Byzantine Broadcast [7, §IV] or for Interactive Consistency [34, §3]. Both follow the same Disperse-Agree-Retrieve template: first disperse the long value, second agree on a short ECP commitment $c$ to a long value, and then retrieve the long pre-image of $c$. Here, we explain why the construction is not correct.

We focus on the Byzantine-Broadcast version of [7, §IV]; the PBC extension of [34, §3] uses the same locally validated commitment idea. In the proposed extension, the source sends a long value $M$, computes an ECP commitment $c$, and invokes a $\kappa$-bit seed broadcast on $c$. The seed broadcast is equipped with a predicate $Q$ saying that $Q(c)$ is valid for a process only if that process has previously received a preimage $M$ from the source whose ECP commitment is $c$.

This is not an externally verifiable predicate. It depends on the local history of the verifier, and different honest processes may evaluate it differently at the same time. Consequently, one cannot simply take an ordinary Byzantine Broadcast protocol and "add" this predicate while preserving its agreement proof. For example, in a Dolev–Strong style seed protocol, an honest process that has observed the preimage of $c$ may accept and relay evidence for $c$, while another honest process that has not observed the preimage rejects the same evidence and may eventually output $\perp$. The seed protocol is no longer being used under the hypotheses of its correctness proof.

A natural repair is to enforce observation-synchronicity: whenever one honest process observes a preimage of $c$, all honest processes observe a preimage of the same $c$ within a known delay $\Delta_{\text{obs}}$. Then one could simulate the seed protocol with an additional delay, in the spirit of the synchronizing transformation of Lindell et al. [68], so that no honest process rejects a message merely because the corresponding preimage has not yet arrived. However, this repair has its own cost. If the source equivocates and sends many long values with distinct commitments, then making observations synchronous for all such commitments forces honest processes to disseminate all conflicting long preimages, losing the claimed communication bound.

## E Learn-or-Expose in Dishonest Majority

We now describe the Learn-or-Expose procedure. The main tool is the notion of a *polarizer*, introduced in [69] and further refined in [40], which in turn is inspired by the trust-graph technique of [85]. At a very high level, the idea is to work with the graph induced by mutual accusations. A related graph-based approach already appeared in the dispute-control paradigm [13], which was later used for Byzantine Broadcast in dishonest-majority settings [23, 63]. However, those constructions incur a large additive overhead and high latency. By contrast, the works [40, 69, 85] exploit structural properties of the resulting graph to obtain more efficient protocols.

*Polarizers.* As mentioned earlier, a natural difficulty in our setting is that a missing message is only *local* evidence of misbehavior: if a party $P_i$ does not receive a message from some party $P^*$, then $P_i$ knows that $P^*$ is faulty, but this fact is not immediately transferable to the other parties.

A solution, introduced in [69], is a weaker and context-dependent analogue of a certificate, called a *polarizer*. Informally, let $A \subseteq \{P_1, \ldots, P_n\}$ and let $C = \{P_1, \ldots, P_n\} \setminus A$. Suppose that every party in $A$ signs accusations against every party in $C$. Then this set of accusations can serve as evidence of corruption in any protocol that guarantees that honest parties never accuse one another. In such a case, a party $P_i$ accepts the polarizer as valid evidence of the corruption of $P^*$ whenever $P_i \in A$ and $P^* \in C$.

The key property is transferability within the protocol context. Indeed, if some honest party $P_i$ accepts such a polarizer, then any other honest party $P_j$ must also accept it. Otherwise, if $P_j \in C$, this would imply that $P_i$ accused $P_j$ at some point, which cannot happen since honest parties never accuse honest parties.

A major difference from the honest-majority setting is that, under dishonest majority, corrupted parties may collude and produce a polarizer in which all honest parties lie in $C$ and the corrupted ones lie in $A$. From the point of view of an external observer, there is no way to distinguish which side contains the honest parties. However, this does not create a problem inside the protocol: honest parties do not accept such a polarizer, and therefore soundness is preserved in the protocol context.

*The original construction from [69] in our setting.* We first recall how polarizers are constructed in [69]. The construction relies on digital signatures.

After the input collection attempt, the honest leader has received signed inputs from at least all honest parties. For every party $P_i$ from which no signed input was received, the leader treats this as a missing message and initiates the Learn-or-Expose procedure for $P_i$.

In the first step, the leader $P_\ell$ signs and sends an accusation against every such party $P_i$ to all parties.

Now fix some accused party $P_i$. After seeing the leader's accusation against $P_i$, every party first asks $P_i$ to provide its signed message. Then each party behaves as follows:

- if it receives a signed message from $P_i$, it forwards that message to all parties;
- otherwise, it signs and sends an accusation against $P_i$ to all parties.

Thus, once the leader accuses $P_i$, it expects every other party either to forward a signed message from $P_i$ or to accuse $P_i$ as well.

If $P_i$ is indeed faulty and all parties respond consistently, then the leader eventually gathers accusations from all parties against $P_i$, which yields a valid polarizer against $P_i$.

However, it may happen that some party $P_j$ neither forwards a message from $P_i$ nor sends an accusation against $P_i$. In that case, the leader still does not have a valid polarizer against $P_i$, since the response of $P_j$ is missing. On the other hand, the leader now learns that $P_j$ is corrupted. Therefore, in the next round, the leader signs and sends an accusation against $P_j$ to all parties.

The protocol then continues recursively in the same manner. Whenever a party fails to provide the expected response, it is itself accused. At the same time, all signed messages and all accusations seen so far are forwarded further. Eventually, the leader obtains either:

- a valid signed message from the originally accused party $P_i$, or
- a valid polarizer against $P_i$.

The drawback of this direct recursive approach is that it may require a linear number of rounds in the worst case.

*From linear to a constant number of rounds.* We now explain the constant-round idea from [40], which in turn is borrowed from [85]. The main observation is that one does not need the full family of accusations from all parties in $A$ against all parties in $C$ in order to preserve transferability. A smaller set of accusations already suffices.

To reason about this, it is convenient to build a graph from the accusations observed so far. The vertex set is the set of parties. We place an undirected edge $\{P_k, P_\ell\}$ whenever there is *no* accusation from $P_k$ against $P_\ell$ among the collected accusations. Since honest parties never accuse one another, all honest parties always remain pairwise adjacent, and thus they form a clique of size at least $n - t$.

This suggests the following pruning rule: repeatedly remove every edge that does not belong to any clique of size $n - t$. Such an edge cannot connect two honest parties, because honest parties must lie inside the honest clique of size at least $n - t$. We note, however, that checking whether an edge belongs to a clique of size $k$ is NP-hard in general. We explain later how this issue is handled in [40, 85].

After this pruning step, if an honest party $P_i$ becomes disconnected from the some party $P_j$, then $P_i$ accepts the collected accusations as a valid polarizer against the $P_j$. Moreover, this notion remains transferable: if one honest party is disconnected from $P_j$, then every honest party is disconnected from $P_j$, because all honest parties remain connected to one another inside the honest clique.

*Distance growth argument.* The reason this leads to a constant-round protocol is the following distance-expansion phenomenon.

For the simplicity of narrative, suppose that after the input collection attempt, the leader is missing only one value, namely the signed message of $P^*$. Then the leader initiates the Learn-or-Expose procedure for $P^*$ by accusing $P^*$. If, after the first round of this procedure, the leader still does not receive a valid signed message from $P^*$, then in the graph constructed from the observed accusations, the distance between $P_\ell$ and $P^*$ is at least 2.

Next, suppose that by round 2 the leader still has neither received a valid message from $P^*$ nor accumulated enough accusations to disconnect from $P^*$. Then the leader accuses all parties that failed to accuse $P^*$ in the previous round. This again pushes any possible path from $P_\ell$ to $P^*$ one step further away, and therefore after round 2 the distance between $P_\ell$ and $P^*$ is at least 3.

Continuing inductively, if by round $r$ the leader has neither received a valid signed message from $P^*$ nor obtained enough accusations to disconnect from $P^*$, then the distance between $P_\ell$ and $P^*$ in the pruned graph is at least $r$.

On the other hand, after pruning, every remaining edge belongs to a clique of size at least $n - t$. We now use Lemma 2 from [40] (or, equivalently, Claim 3.1 from [85]): if $G$ is a graph on $n$ nodes in which every edge is contained in a clique of size $h$, then every connected component of $G$ has diameter at most $2n/h$. Applying the lemma with $h = n-t$, we obtain that the diameter of the graph is at most $\frac{2n}{n-t}$, and therefore $O\big(\frac{n}{n-t}\big)$. In particular, when $t \le (1-\varepsilon)n$ for a constant $\varepsilon > 0$, this diameter is $O(1)$.

Therefore, after only a constant number of rounds, the leader either:

- receives a valid signed message from $P^*$, or
- becomes disconnected from $P^*$ and thus obtains a valid polarizer against $P^*$.

*MakeGraph.* As mentioned above, searching for a clique in a graph is computationally hard. We now explain how this issue is addressed in [40], adapting the approach of [85].

The idea is to use an efficient pruning procedure, denoted by MakeGraph. Instead of removing all edges that do not belong to a clique of size $h$, MakeGraph removes every edge $(u, v)$ such that $|N_G(u) \cap N_G(v)| < h$. Lemma 13 of [40] (Lemma 3.3 of [85]) shows that the diameter of any graph obtained after this pruning step is at most $2n/h$. Algorithm 3 implements this procedure. For the sake of completeness, we provide the corresponding lemma in Appendix G.

*Aggregation of proofs of misbehavior.* In the worst case, the accusation set Acc may contain $\Omega(n^2)$ accusations. If stored individually, each accusation consists of a signed message of size $O(\kappa)$ bits, and thus the total size of Acc is $\Omega(n^2\kappa)$ bits. This bound can be improved using the multi-signature primitive introduced above. For each party $p_j$, let $\mathrm{Acc}_j \subseteq \mathrm{Acc}$ denote the set of accusations against $p_j$. We aggregate all signatures in $\mathrm{Acc}_j$ into a single multi-signature on the message $\langle \mathrm{accuse}, p_j \rangle$, together with the corresponding signer set. We denote this aggregation procedure by Aggregate. The resulting proof of misbehavior consists of at most $n$ aggregated accusations, one for each party. Each aggregated accusation contains a signer set of size $O(n)$ bits and one multi-signature of size $O(\kappa)$ bits, and therefore has total size $O(n + \kappa)$. Hence, the total size of the aggregated proof of misbehavior is $O(n^2 + n\kappa)$ bits.

# F Honest Majority Instantiation

*F.0.1 Learn-or-Expose.* For the honest-majority setting, that is, $t < n/2$, we use the LearnOrExpose procedure from [25], instantiated with a threshold-signature scheme. The protocol starts from the current partial candidate vector $(\vec{v}, \vec{\sigma})$. For each index $j \in [n]$, every party proceeds as follows: if $\vec{v}[j] \neq \perp$, it forwards the corresponding disclose message $(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \vec{\sigma}[j])$ to the leader; otherwise, it sends to the leader a partial threshold signature on the accusation message $\langle \mathsf{accuse}, p_j \rangle$.

Thus, for every missing party $p_j$, the leader eventually obtains either a valid disclose message of $p_j$, or a collection of partial accusation signatures against $p_j$. If the leader receives at least $t+1$ valid partial signatures on $\langle \mathsf{accuse}, p_j \rangle$, it combines them into a threshold signature using $\mathsf{Combine}^{t+1}$. This combined signature serves as a transferable, and in fact publicly verifiable, proof of misbehavior for $p_j$.

As a result, after one round, the leader completes the candidate vector together with auxiliary evidence certifying both the disclosed entries and the missing ones. Concretely, the disclosed entries are certified by the forwarded signatures collected in $\vec{\sigma}$, while the missing entries are certified by the combined threshold accusations.

**Algorithm 9** $\textsc{LearnOrExpose}^{\mathrm{hm}}_{p_\ell}(\vec{v}, \vec{\sigma})$ at process $p_i$

```
1: Π ← ∅
2: for each j ∈ [n] do
3:     if v⃗[j] ≠ ⊥ then
4:         send (⟨disclose, v⃗[j]⟩, σ⃗[j]) to p_ℓ
5:     else
6:         send ⟨accuse, p_j, ShareSign_i^{t+1}(⟨accuse, p_j⟩)⟩ to p_ℓ
7: wait until all messages of the current round are delivered
8: if p_i = p_ℓ then
9:     Missing ← {p_j ∈ [n] : v⃗[j] = ⊥}
10:    for each p_j ∈ Missing do
11:        if received a valid signed message (⟨disclose, m_j⟩, σ_j) for p_j then
12:            v⃗[j] ← m_j
13:            σ⃗[j] ← σ_j
14:        else
15:            S_j ← {τ_k | received ⟨accuse, p_j, τ_k⟩ and ShareVerify_k^{t+1}(⟨accuse, p_j⟩, τ_k) =
   1}
16:            if |S_j| ≥ t + 1 then
17:                Π[j] ← Combine^{t+1}(S_j)
18:    return (v⃗, σ⃗, Π)
19: else
20:    return
```

Lemma F.1. *Define* $\mathsf{VerifyVec}^{in}_i(\vec{v}, \vec{\sigma})$ *by* $\mathsf{VerifyVec}^{in}_i(\vec{v}, \vec{\sigma}) = \mathsf{true}$ *iff for every* $j \in [n]$ *with* $\vec{v}[j] \neq \perp$, $\mathsf{ver}_{\mathsf{pk}_j}(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \vec{\sigma}[j]) = \mathsf{true}$.

*Define* $\mathsf{VerifyVec}^{out}_i(\vec{v}, (\vec{\sigma}, \Pi))$ *by* $\mathsf{VerifyVec}^{out}_i(\vec{v}, (\vec{\sigma}, \Pi)) = \mathsf{true}$ *iff the following hold:*

*(1) for every* $j \in [n]$ *with* $\vec{v}[j] \neq \perp$, $\mathsf{ver}_{\mathsf{pk}_j}(\langle \mathsf{disclose}, \vec{v}[j] \rangle, \vec{\sigma}[j]) = \mathsf{true}$;

*(2) for every* $j \in [n]$ *with* $\vec{v}[j] = \perp$, $\mathsf{CombinedVerify}^{t+1}(\langle \mathsf{accuse}, p_j \rangle, \Pi[j]) = \mathsf{true}$.

*Then Algorithm 9 realizes Module 3 in the honest-majority setting. Its latency is one round, its communication complexity is* $\Theta\big(n^2(L_{\mathsf{in}} + \kappa)\big)$, *and its proof size satisfies* $\lambda_{\mathsf{LoE}} = O(n\kappa)$.

Proof. We verify the four properties of Module 3.

*Termination.* Algorithm 9 consists of one communication round, after which the leader locally processes the received messages and returns. Hence all honest processes terminate after fixed known latency one round.

*Progress.* Assume that the leader $p_\ell$ is honest. Fix any index $j \in [n]$. Either at least one honest process knows the message $m_j$ and, by the definition of $\mathsf{VerifyVec}^{in}$, this message is accompanied by a valid disclose signature, so it can be forwarded to the leader; or all honest processes accuse $p_j$ and send partial signatures on the accusation message. In the latter case, the leader receives at least $n - t$ valid partial signatures, and since $n - t \geq t + 1$, this is enough to combine them into a valid threshold proof of misbehavior.

Thus, for every coordinate $j$, the honest leader obtains either a valid disclose message or a valid threshold accusation proof. Therefore it returns some $(\vec{v}, \pi)$ such that $\mathsf{VerifyVec}^{out}_\ell(\vec{v}, \pi) = \mathsf{true}$. This proves Progress.

*Transferability.* Suppose that for some honest process $p_i$ we have $\mathsf{VerifyVec}^{out}_i(\vec{v}, \pi) = \mathsf{true}$. We show that every honest process $p_{i'}$ also accepts. For every coordinate $j$ with $\vec{v}[j] \neq \perp$, verification checks only a standard signature by $p_j$, which is publicly verifiable and therefore independent of the verifier. For every coordinate $j$ with $\vec{v}[j] = \perp$, verification checks only that $\Pi[j]$ is a valid combined threshold signature on $\langle \mathsf{accuse}, p_j \rangle$. Again, this is publicly verifiable and independent of the verifier. Hence all honest processes evaluate exactly the same conditions on the same proof $\pi$, and therefore $\mathsf{VerifyVec}^{out}_i(\vec{v}, \pi) = \mathsf{VerifyVec}^{out}_{i'}(\vec{v}, \pi)$ for all honest $p_i, p_{i'}$. This proves Transferability.

*Output Soundness.* Assume that an honest process $p_i$ accepts $(\vec{v}, \pi)$, that is, $\mathsf{VerifyVec}^{out}_i(\vec{v}, \pi) = \mathsf{true}$.

First consider a coordinate $j$ with $\vec{v}[j] \neq \perp$. Then, by Honest Input Agreement, Input Soundness, and the unforgeability of signatures, the entry $\vec{v}[j]$ must be exactly the input submitted by $p_j$.

Now consider a coordinate $j$ with $\vec{v}[j] = \perp$. Then $\Pi[j]$ is a valid $(t+1)$-threshold signature on $\langle \mathsf{accuse}, p_j \rangle$. Such a threshold signature can only be produced from at least $t+1$ valid partial signatures on that message. Since at most $t$ parties are corrupted, at least one of these partial signatures must come from an honest process. However, an honest process sends a partial accusation signature against $p_j$ only if its local entry for $j$ is $\perp$, which by the Honest Input Agreement precondition can happen only if $p_j$ is dishonest. Therefore, no valid combined accusation proof can exist against an honest $p_j$.

Hence, for every honest process $p_j$, the entry $\vec{v}[j]$ cannot be $\perp$. Combining this with the first case, we conclude that for every honest $p_j$, the entry $\vec{v}[j]$ equals the input of $p_j$. This proves Soundness.

Finally, the proof consists of the vector $\vec{\sigma}$ of disclose signatures and the vector $\Pi$ of threshold accusations. Each signature and each combined threshold signature has size $O(\kappa)$, and there are at most $n$ entries of each type. Therefore, the total proof size is $\lambda_{\mathsf{LoE}} = O(n\kappa)$. Moreover, each process sends at most $\Theta(n(L_{\mathsf{in}} + \kappa))$ bits, since for each coordinate it sends either one disclose message of size $O(L_{\mathsf{in}} + \kappa)$ or one partial accusation signature of size $O(\kappa)$. Hence the total communication complexity is $O\big(n^2(L_{\mathsf{in}} + \kappa)\big)$. □

*F.0.2 Leader Election.* For leader election in the honest-majority setting, we use the approach underlying [19]. In the Random Oracle Model, it yields a leader-election protocol with one-round latency on top of a threshold-signature scheme with signature uniqueness. Such a threshold-signature scheme can in turn be instantiated in constant rounds and quadratic communication during a setup phase, for example, using [12].

*F.0.3 Byzantine Broadcast.* For Byzantine Broadcast in the honest-majority setting, we use the protocol of [3]. For $\kappa$-bit inputs, it provides Byzantine Broadcast with expected $O(1)$ rounds and expected $O(n^2)$ communication.

In our setting, however, the input to Byzantine Broadcast is not a short message, but a verifiable vector of size $L_{\mathsf{vv}} = \Theta\big(n(L_{\mathsf{in}} + \kappa)\big)$. Therefore, to handle such long inputs efficiently, one should combine [3] with a BB extension protocol, for example [71].

This yields a Byzantine Broadcast primitive for verifiable vectors with the same expected $O(1)$ round complexity and communication $O\big(nL_{\mathsf{vv}} + n^2\kappa_a\big)$, where $\kappa_a$ denotes the size of the accumulator used in the extension protocol. Substituting $L_{\mathsf{vv}} = \Theta\big(n(L_{\mathsf{in}} + \kappa)\big)$, we obtain total communication $O\big(n^2(L_{\mathsf{in}} + \kappa + \kappa_a)\big)$. Concretely, one gets $\kappa_a = \kappa \log n$ with a Merkle-tree-based accumulator, or $\kappa_a = \kappa$ with a bilinear accumulator, at the price of a powers-of-$\tau$ setup.

# G Missed Proofs

**Claim 1 from Lemma 5.1.** Honest parties never accuse honest parties.

Proof. For every honest process $p_i$ and every round $r \geq 1$, let $\mathsf{Acc}_i^{(r)}$ be the set of valid accusations known to $p_i$ immediately before line 16 in round $r$, and let $G_i^{(r)} := \mathsf{MakeGraph}(\mathsf{Acc}_i^{(r)})$. We show that no honest process ever sends an accusation against an honest process.

First note that this cannot happen in round 1. Indeed, in round 1 a process accuses only parties whose values are still unresolved after the initial dissemination step. But every honest party broadcasts its valid disclose message to all parties before the first round of LearnOrExpose, and therefore no honest party can still be unresolved at an honest process in round 1.

Assume now toward contradiction that some honest process $p_i$ sends the first accusation against an honest process $p_j$ in round $r \geq 2$. By construction of Algorithm 2, this means that there exists some unresolved process $p^*$ such that

$$p_j \in N_{G_i^{(r)}}(p_i) \cap B_{G_i^{(r)}}(p^*, r-1). \tag{1}$$

We claim that, by the end of round $r-1$, process $p_j$ has not yet received a valid disclose message for $p^*$. Indeed, if $p_j$ had received such a disclose message in round $r-1$, then, being honest, it would broadcast it in the same round, and $p_i$ would receive it before computing the frontier in round $r$. Consequently, $p_i$ would resolve $p^*$ before accusing anyone for this coordinate in round $r$, contradicting the fact that $p^*$ is still unresolved when $p_i$ accuses $p_j$.

Since $p_j$ is honest and has not resolved $p^*$ by the end of round $r-1$, in round $r-1$ it accuses all parties in

$$N_{G_j^{(r-1)}}(p_j) \cap B_{G_j^{(r-1)}}(p^*, r-2). \tag{2}$$

Let $\widehat{\mathsf{Acc}}_j^{(r-1)}$ denote this set of accusations, and define

$$\widehat{G}_j^{(r-1)} := \mathsf{MakeGraph}\big(\mathsf{Acc}_j^{(r-1)} \cup \widehat{\mathsf{Acc}}_j^{(r-1)}\big). \tag{3}$$

By construction, every remaining neighbor of $p_j$ in $\widehat{G}_j^{(r-1)}$ is at distance at least $r-1$ from $p^*$. Hence

$$\mathsf{dist}_{\widehat{G}_j^{(r-1)}}(p^*, p_j) \geq r. \tag{4}$$

On the other hand, since $p_j$ is honest, all valid accusations known to $p_j$ by the end of round $r-1$, together with all accusations sent by $p_j$ in round $r-1$, are known to every honest process before line 16 in round $r$. Hence

$$\mathsf{Acc}_i^{(r)} \supseteq \mathsf{Acc}_j^{(r-1)} \cup \widehat{\mathsf{Acc}}_j^{(r-1)}. \tag{5}$$

It follows that

$$G_i^{(r)} \subseteq \widehat{G}_j^{(r-1)}. \tag{6}$$

Therefore,

$$\mathsf{dist}_{G_i^{(r)}}(p^*, p_j) \geq \mathsf{dist}_{\widehat{G}_j^{(r-1)}}(p^*, p_j) \geq r. \tag{7}$$

But (1) implies that $\mathsf{dist}_{G_i^{(r)}}(p^*, p_j) \leq r-1$, contradicting (7). This contradiction shows that no honest process ever accuses an honest process. □

**Lemma 2 from [40].** Let $G$ be a graph with $n$ nodes in which every edge is contained in a clique of size $h$. Then any connected component of $G$ has diameter at most $d = \frac{2n}{h}$.

Proof. It is enough to prove the claim for a connected component of $G$, so assume without loss of generality that $G$ is connected. Let $d'$ be the diameter of $G$. We show that $d' \leq d$.

Let $v_0$ and $v_{d'}$ be nodes of $G$ such that $\mathsf{dist}_G(v_0, v_{d'}) = d'$, and let

$$e_1 = \{v_0, v_1\}, \quad e_2 = \{v_1, v_2\}, \quad \ldots, \quad e_{d'} = \{v_{d'-1}, v_{d'}\}$$

be a path of length $d'$ between $v_0$ and $v_{d'}$. By assumption, for every $i \in [d']$, the edge $e_i$ is contained in a clique $C_i$ of size $h$.

We claim that any two cliques $C_i$ and $C_j$ with $j - i \geq 2$ are disjoint. Indeed, suppose for contradiction that there exists a node $v'$ belonging to both $C_i$ and $C_j$. Then there are edges $\{v_i, v'\}$ and $\{v', v_{j+1}\}$ in $G$. Hence

$$e_1 = \{v_0, v_1\},\ \ldots,\ e_i = \{v_{i-1}, v_i\},\ \{v_i, v'\},\ \{v', v_{j+1}\},$$
$$e_{j+1} = \{v_{j+1}, v_{j+2}\},\ \ldots,\ e_{d'} = \{v_{d'-1}, v_{d'}\}$$

is a path of length $d' - 1$ from $v_0$ to $v_{d'}$, contradicting the definition of $d'$. Thus $C_i \cap C_j = \emptyset$ whenever $j - i \geq 2$.

It follows that the cliques with even indices are pairwise disjoint, and the cliques with odd indices are pairwise disjoint. Therefore,

$$\begin{aligned} 2n &\geq |C_2 \cup C_4 \cup C_6 \cup \cdots| + |C_1 \cup C_3 \cup C_5 \cup \cdots| \\ &= |C_2| + |C_4| + |C_6| + \cdots + |C_1| + |C_3| + |C_5| + \cdots \\ &= |C_1| + |C_2| + \cdots + |C_{d'}| \\ &= d' \cdot h. \end{aligned}$$

Hence $d' \leq \frac{2n}{h} = d$, as claimed. □

**Claim 2 from Lemma 5.1.** Every connected component of every graph obtained by Algorithm 3 has diameter at most $\left\lceil \frac{2n}{n-t} \right\rceil$.

Proof. Let $G = \mathsf{MakeGraph}(\mathrm{Acc})$, and let $C$ be any connected component of $G$. By inspection of Algorithm 3, for every edge $\{u, v\}$ in $G$ we have $|N_G(u) \cap N_G(v)| \geq n - t$. In particular, this holds for every edge of $C$.

Assume toward contradiction that the diameter of $C$ is $d' > \frac{2n}{n-t}$. Let $v_0$ and $v_{d'}$ be nodes of $C$ such that $\mathrm{dist}_C(v_0, v_{d'}) = d'$, and let $e_1 = \{v_0, v_1\}, e_2 = \{v_1, v_2\}, \ldots, e_{d'} = \{v_{d'-1}, v_{d'}\}$ be a path of length $d'$ connecting $v_0$ and $v_{d'}$.

For each $i \in \{0, \ldots, d'\}$, define $D_i = \{w \in C \mid \mathrm{dist}_C(v_0, w) = i\}$. By definition of graph distance, for all $i \neq j$ we have $D_i \cap D_j = \emptyset$. Moreover, for every $i \in \{0, \ldots, d'\}$ we have $v_i \in D_i$, and for every $i \in \{0, \ldots, d' - 1\}$, $N_C(v_i) \subseteq D_{i-1} \cup D_i \cup D_{i+1}$, where we use the convention $D_{-1} = \emptyset$ and $D_{d'+1} = \emptyset$.

Now define a graph $G' = (V', E')$ by $V' = \bigcup_{i=0}^{d'} D_i$ and $E' = \left(\bigcup_{i=0}^{d'} (D_i \times D_i)\right) \cup \left(\bigcup_{i=0}^{d'-1} (D_i \times D_{i+1})\right)$. Since the sets $D_i$ are pairwise disjoint, we still have $\mathrm{dist}_{G'}(v_0, v_{d'}) = d'$, and therefore the diameter of $G'$ is at least $d'$.

Next, consider any edge $e = \{a, b\} \in E'$. Then there exists $i \in \{0, \ldots, d' - 1\}$ such that $a \in D_i$ and $b \in D_i \cup D_{i+1}$. Hence $e$ is contained in the clique induced by $D_i \cup D_{i+1}$. It remains to lower-bound the size of this clique. For every $i \in \{0, \ldots, d' - 1\}$,

$$\begin{aligned} n - t &\leq |N_C(v_i) \cap N_C(v_{i+1})| \\ &\leq |(D_{i-1} \cup D_i \cup D_{i+1}) \cap (D_i \cup D_{i+1} \cup D_{i+2})| \\ &= |D_i \cup D_{i+1}| \\ &= |D_i| + |D_{i+1}|. \end{aligned}$$

Thus every edge of $G'$ is contained in a clique of size at least $n - t$.

By Lemma 2 from [40], proved above, every connected component of $G'$ has diameter at most $\frac{2n}{n-t}$, contradicting the fact that $G'$ has diameter at least $d' > \frac{2n}{n-t}$. Therefore, $\mathrm{diam}(C) \leq \frac{2n}{n-t} \leq \left\lceil \frac{2n}{n-t} \right\rceil$. Since $C$ was an arbitrary connected component of $G$, the claim follows. □