

# Toward Optimal-Complexity Hash-Based Asynchronous MVBA with Optimal Resilience

Jovan Komatovic<sup>1</sup>, Joachim Neu<sup>2</sup>, and Tim Roughgarden<sup>2,3</sup>

<sup>1</sup> École Polytechnique Fédérale de Lausanne (EPFL)

`jovan.komatovic@epfl.ch`

<sup>2</sup> a16z Crypto Research

`{jne,roughgarden}@a16z.com`

<sup>3</sup> Columbia University

`tim.roughgarden@gmail.com`

**Abstract.** Multi-valued validated Byzantine agreement (MVBA), a fundamental primitive of distributed computing, enables  $n$  processes to agree on a valid  $\ell$ -bit value, despite  $t$  faulty processes behaving arbitrarily. Among hash-based protocols for the asynchronous setting with adaptive faults, the state-of-the-art HMOVBA protocol has optimal  $O(1)$  time complexity and near-optimal  $O(n\ell + n^2\lambda \log n)$  bit complexity, but tolerates only  $t < n/5$  faults. We present **Reducer**, an MVBA protocol that matches HMOVBA’s time and bit complexity and improves resilience to  $t < n/4$ . Like HMOVBA, **Reducer** relies solely on collision-resistant hash functions. Toward optimal one-third resilience, we also propose **Reducer++**, an MVBA protocol with further improved  $t < (1/3 - \epsilon)n$  resilience, for any fixed  $\epsilon > 0$ , assuming hash functions modeled as random oracles. Time and bit complexity of **Reducer++** remain constant and quasi-quadratic, respectively, with constants depending on  $\epsilon$ .

## 1 Introduction

Multi-valued validated Byzantine agreement (MVBA), first introduced in [16], has become a fundamental building block for secure distributed systems, such as fault-tolerant replicated state-machines. MVBA protocols enable  $n$  processes in a message-passing model to agree on an  $\ell$ -bit value that satisfies a fixed external validity predicate, despite  $t$  faulty processes deviating from the protocol in any arbitrary, possibly coordinated, but computationally bounded manner. Of particular interest, due to superior robustness to network conditions, is MVBA under asynchrony, where messages are guaranteed eventual delivery, but no assumptions are made about the timing. The design of *asynchronous MVBA protocols* is the subject of this paper.

The seminal FLP impossibility result [36] implies that no deterministic algorithm can solve asynchronous MVBA. In other words, any asynchronous MVBA protocol must employ randomness. Since then, it has become standard practice [35,32,62,46,9,11] to construct asynchronous MVBA protocols in two parts: an *idealized common-coin abstraction*, and an otherwise (possibly) deterministic

*protocol core.* The common coin encapsulates the randomness, and upon invocation by sufficiently many processes provides the same unpredictable and unbiased random sequence to all processes. The rest of the protocol is the actual deterministic distributed-computing “core mechanism”. We adopt this blueprint as well. The common-coin abstraction can be realized using a trusted dealer [58]. By relying on threshold pseudorandom functions [17] or dedicated common-coin protocols [30,38,7], the need for a trusted dealer can be eliminated.

For everything other than instantiating the common-coin abstraction, we pursue a *hash-based* protocol, that is, in particular, setup-free and signature-free. Hash-based protocols rely on relatively cheap “unstructured” operations like hashes instead of the relatively expensive “highly-structured” algebraic operations that underlie, for instance, public-key or threshold cryptography. As a result, hash-based protocols are plausibly post-quantum secure, and, *ceteris paribus*, tend to be more performant. Furthermore, they avoid the complexity and trust issues that comes with trusted or private setups.

Finally, [16] has hinted and [35, Sec. 1.2] has shown that the design of good hash-based asynchronous MVBA protocols is easy if security is required only against a static adversary, who determines which processes to corrupt at the beginning of the execution before any randomness of the protocol’s common coin is sampled. The gold standard, however, is security against *adaptive* adversaries, who are at liberty to decide which processes to corrupt during the protocol execution as randomness is revealed. We thus pursue *adaptive security*.

*Scope & state-of-the-art.* In summary, the broad subject of this paper is *adaptively-secure, hash-based, asynchronous MVBA*. Within this scope, as is usual, we want to minimize a protocol’s time complexity and bit complexity and to maximize its resilience (number  $t$  of faulty processes it can tolerate relative to number  $n$  of all processes). The best known MVBA protocols in this context are HMOVBA [35] and FIN-MVBA [32] (see Tab. 1). The HMOVBA protocol relies solely on collision-resistant hash functions and achieves near-optimal  $O(n\ell + n^2\lambda \log n)$  expected bit complexity (where  $\lambda$  denotes the size of a hash value) and optimal  $O(1)$  expected time complexity. A key drawback of HMOVBA is its sub-optimal  $t < n/5$  resilience. FIN-MVBA, on the other hand, tolerates up to  $t < n/3$  faults, while also relying exclusively on collision-resistant hash functions and achieving optimal constant expected time complexity, but comes with a whopping  $O(n^2\ell + n^3\lambda)$  expected bit complexity, not improving over what is achieved when threshold signatures are naively instantiated with lists of hash-based signatures in “classic” threshold-signature-based MVBA protocols [4,46,40,16] [35, Table 1].

*Contributions.* It is therefore natural to ask for a protocol that combines the strengths of HMOVBA and FIN-MVBA. To this end, we present **Reducer**, an adaptively-secure hash-based asynchronous MVBA protocol that matches the HMOVBA protocol in expected time and bit complexity and improves upon HMOVBA’s resilience with  $t < n/4$  resilience (see Tab. 1). Like HMOVBA, **Reducer** relies solely on collision-resistant hash functions. To approach the optimal one-third resilience found in FIN-MVBA, we also propose **Reducer++**, a hash-based MVBA protocol with further improved  $t < (1/3 - \epsilon)n$  resilience, for any fixed

**Table 1:** State-of-the-art adaptively-secure asynchronous MVBA protocols in terms of whether they are hash-based, expected bit complexity, expected time complexity, and resilience  $t$  (each as a function of number of processes  $n$ ). [35, Table 1]

Construction	Hash-based	Bits	Time	Resilience
CKPS01-MVBA [16] <sup>TS</sup>	✗	$O(n^2\ell + n^2\lambda + n^3)$	$O(1)$	$t < \frac{1}{3}n$
CKPS01-MVBA/HS [16] <sup>H-CR</sup>	✓	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
VABA [4] <sup>TS</sup>	✗	$O(n^2\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
VABA/HS [4] <sup>H-CR</sup>	✓	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-MVBA [46] <sup>TS</sup>	✗	$O(n\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-MVBA/HS [46] <sup>H-CR</sup>	✓	$O(n\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
sMVBA [40] <sup>TS</sup>	✗	$O(n^2\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
sMVBA/HS [40] <sup>H-CR</sup>	✓	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-sMVBA [46,40] <sup>TS</sup>	✗	$O(n\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-sMVBA/HS [46,40] <sup>H-CR</sup>	✓	$O(n\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
FIN-MVBA [32] <sup>H-CR</sup>	✓	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
HMVBA [35] <sup>H-CR</sup>	✓	$O(n\ell + n^2\lambda \log n)$	$O(1)$	$t < \frac{1}{5}n$
This work: Reducer <sup>H-CR</sup>	✓	$O(n\ell + n^2\lambda \log n)$	$O(1)$	$t < \frac{1}{4}n$
This work: Reducer++ <sup>H-RO</sup> ( $\gamma$ : constant depending on $\epsilon$ )	✓	$O(\gamma(n\ell + n^2\lambda \log n))$	$O(\gamma)$	$t < (\frac{1}{3} - \epsilon)n$

“H-CR”, “H-RO”, and “TS” use collision-resistant hashes, hashes modeled as a random oracle, and threshold signatures, respectively,  $\lambda$ -length each. “/HS” denotes the hash-based variant where threshold signatures in the original protocol are replaced by lists of hash-based signatures from  $n - t$  processes. Dumbo-sMVBA results from applying Dumbo-MVBA’s framework to sMVBA. As is convention in the asynchronous-consensus literature, bit/time complexity are provided for the protocol core, taking common coins for granted and consistently omitting their cost. Bit complexity is marked **green** only with optimal linear  $\ell$ - and near-optimal quasi-quadratic  $\lambda$ -factor.

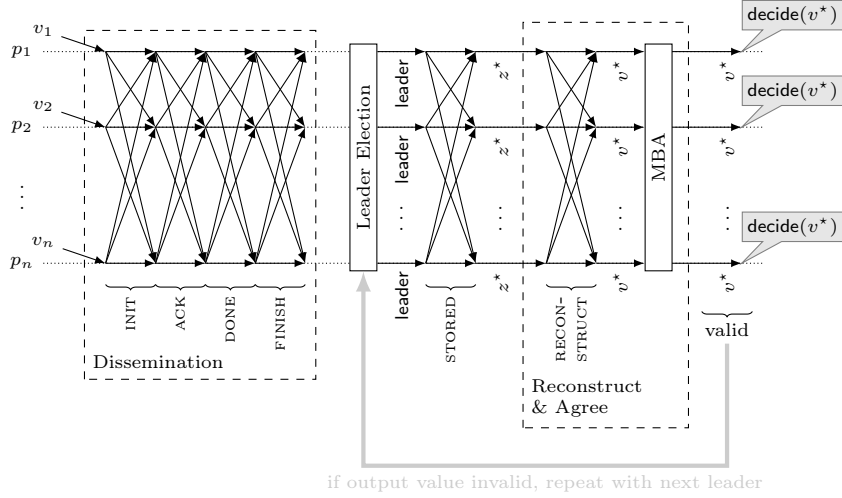
$\epsilon \in (0, 1)$  independent of  $n$ . Expected time and bit complexity of Reducer++ remain constant and quasi-quadratic, respectively, with constants depending on  $\epsilon$ . Reducer++, however, necessitates additional (standard) assumptions regarding the utilized hash function, namely that it can be modeled as a random oracle.

## 2 Technical Overview

Given that the starting point for designing both Reducer and Reducer++ was HMVBA, we begin by revisiting HMVBA (§2.1) as doing so will allow the best insight into the design choices behind our algorithms. Then, we outline the key mechanisms behind Reducer (§2.2) and Reducer++ (§2.3).

### 2.1 Revisiting HMVBA

As previously mentioned, HMVBA solves the MVBA problem with  $O(n\ell + n^2\lambda \log n)$  expected bit complexity and  $O(1)$  expected time complexity among  $n = 5t + 1$  processes. At a high level, HMVBA follows the “Disseminate-Elect-Agree” paradigm, which we describe below. To emphasize key aspects of the HMVBA algorithm and their impact on our designs, we make slight modifications to the original algorithm (without hindering its correctness or asymptotic complexity). Fig. 1 depicts the structure of HMVBA.



**Fig. 1:** Depiction of HMVBA’s structure. The depiction focuses on a good iteration  $k$ , where  $\text{leader}(k)$  has disseminated its valid proposal  $v^*(k)$  and the corresponding digest  $z^*(k)$ . We abridge  $\text{leader} \triangleq \text{leader}(k)$ ,  $z^* \triangleq z^*(k)$ ,  $v^* \triangleq v^*(k)$ .

**Dissemination phase.** Each process first disseminates its proposal. Specifically, when a correct process  $p_i$  proposes a valid value  $v_i$ , process  $p_i$  computes  $n$  Reed-Solomon (RS) symbols  $\{m_1, m_2, \dots, m_n\}$  of value  $v_i$ , where  $v_i$  is treated as a polynomial of degree  $t$ . Process  $p_i$  then utilizes Merkle-tree-based [48] cryptographic accumulators in the following manner:

1. Process  $p_i$  computes the accumulation value (i.e., the Merkle root)  $z_i$  for the  $\{m_1, m_2, \dots, m_n\}$  set. We refer to  $z_i$  as the *digest* of  $p_i$ ’s proposal  $v_i$ . The terms “accumulation value” and “digest” are used interchangeably.
2. For each RS symbol  $m_j$ , process  $p_i$  computes the witness (i.e., the Merkle proof of inclusion)  $w_j$  proving that  $m_j$  belongs to the  $\{m_1, m_2, \dots, m_n\}$  set.

Subsequently, process  $p_i$  sends each RS symbol  $m_j$  along with the digest  $z_i$  and witness  $w_j$  via an INIT message to process  $p_j$ . Once a process  $p_j$  receives a valid RS symbol  $m_j$  from process  $p_i$ , i.e., an RS symbol corresponding to the received digest  $z_i$  and the received witness  $w_j$ , process  $p_j$  replies back via an ACK message confirming that it has received and stored  $[m_j, z_i, w_j]$ . When process  $p_i$  receives  $n - t = 4t + 1$  ACK messages, process  $p_i$  knows that at least  $(n - t) - t = 3t + 1$  valid RS symbols are stored at as many correct processes. Then, process  $p_i$  broadcasts a DONE message. Once process  $p_i$  receives  $n - t = 4t + 1$  DONE messages, which implies that at least  $(n - t) - t = 3t + 1$  correct processes completed their dissemination, process  $p_i$  broadcasts a FINISH message. Finally, upon receiving  $n - t = 4t + 1$  FINISH messages, process  $p_i$  completes the dissemination phase.

*Key takeaways from the dissemination phase.* First, if a “so-far-uncorrupted” process  $p_i$  successfully disseminates its valid proposal  $v_i$ , it is guaranteed that at least  $(n - t) - t = 3t + 1$  correct processes have stored (1) the digest  $z_i$  of

value  $v_i$ , and (2) RS symbols of the value  $v_i$  (one per process). Hence, even if process  $p_i$  later gets corrupted, the original value  $v_i$  can be reconstructed using the material held only by correct processes.

Second, if a correct process completes the dissemination phase, at least  $(n - t) - t = 3t + 1$  “so-far-uncorrupted” processes have already successfully disseminated their proposals. Thus, there are  $3t + 1$  processes whose proposals can be reconstructed, even if the adversary corrupts them after dissemination. The HMOVBA algorithm leverages this insight in the subsequent two phases.

**Election & agreement phases.** After the dissemination phase is concluded, processes start executing HMOVBA through *iterations*. Each iteration utilizes the multi-valued Byzantine agreement (MBA) primitive [15,10,55] ensuring strong unanimity: if all correct processes propose the same value  $v$ , then  $v$  is decided. (We formally define the MBA primitive in §4.)

At the beginning of each iteration  $k$ , processes go through the election phase: by utilizing a common coin, processes elect the leader of iteration  $k$ , denoted by  $\text{leader}(k)$ . Then, the agreement phase starts. The goal of this phase is for processes to agree on the  $\text{leader}(k)$ ’s proposal. Specifically, once the leader is elected, each process  $p_i$  broadcasts via a STORED message the digest received from  $\text{leader}(k)$  during the dissemination phase. If no such digest was received, process  $p_i$  broadcasts a STORED message with  $\perp$ . Once process  $p_i$  receives  $n - t = 4t + 1$  STORED messages, it executes the following logic:

- If there exists a digest  $z$  received in a majority ( $\geq 2t + 1$ ) of STORED messages, process  $p_i$  adopts  $z$ . If such digest  $z$  exists, it is guaranteed that  $(2t + 1) - t = t + 1$  correct processes have valid RS symbols that correspond to  $z$ . (Recall that a correct process accepts an INIT message with the digest  $z$  during the dissemination phase only if the received RS symbol and witness match  $z$ .)
- Otherwise, process  $p_i$  adopts  $\perp$ .

Then, process  $p_i$  disseminates via a RECONSTRUCT message the RS symbol received from the leader during the dissemination phase. If process  $p_i$  adopted a digest  $z$  ( $\neq \perp$ ),  $p_i$  waits until it receives via the aforementioned RECONSTRUCT messages  $t + 1$  RS symbols corresponding to digest  $z$ , uses the received symbols to rebuild some value  $r_i$ , and proposes  $r_i$  to the MBA primitive.<sup>4</sup> Otherwise, process  $p_i$  proposes its proposal  $v_i$  to the MBA primitive. We refer to the combination of the reconstruction step and the MBA primitive as the Reconstruct & Agree (R&A) mechanism. If value  $v$  decided from the R&A mechanism (i.e., from the underlying MBA primitive) is valid, processes decide  $v$  from HMOVBA and terminate. If not, processes continue to the next iteration  $k + 1$ .

**Correctness analysis.** We now briefly explain how HMOVBA’s design guarantees its correctness. Recall that if a correct process completes the dissemination phase, it is guaranteed that at least  $(n - t) - t = 3t + 1$  “so-far-uncorrupted” processes have already successfully disseminated their proposals. The HMOVBA

<sup>4</sup> The original HMOVBA algorithm combines STORED and RECONSTRUCT messages: each STORED message contains both the digest and the RS symbol received from the leader. For pedagogical clarity, we separate them as they serve distinct functions.

algorithm ensures that all correct processes decide (and terminate) in an iteration  $k$  whose leader is one of the aforementioned  $3t + 1$  “so-far-uncorrupted” processes. In the rest of the paper, we refer to such iterations as *good*. Let us analyze how any such good iteration  $k$  of HMOVBA unfolds.

The dissemination phase ensures that at least  $(n - t) - t = 3t + 1$  correct processes have stored (1) the digest  $z^*(k)$  of the leader( $k$ )’s valid proposal  $v^*(k)$ , and (2) RS symbols of value  $v^*(k)$ . Hence, when each correct process  $p_i$  receives  $n - t = 4t + 1$  STORED messages, it is guaranteed (due to quorum intersection) that  $p_i$  receives the “good” digest  $z^*(k)$  from at least  $(n - t) + (n - 2t) - n = 2t + 1$  processes. Therefore, it is ensured that *all* correct processes adopt digest  $z^*(k)$ . We emphasize that  $n = 5t + 1$  is critical in this step. Namely, the HMOVBA algorithm cannot guarantee that all correct processes adopt  $z^*(k)$  if  $n < 5t + 1$ . As we argue in §2.2, this “adoption issue” is the primary challenge one must overcome to achieve better resilience than  $t < n/5$  while preserving the HMOVBA’s complexity.

From this point on, making all correct processes agree on the leader( $k$ )’s valid value  $v^*(k)$  does not represent a significant challenge. Indeed, using the fact that at least  $t + 1$  correct processes have valid RS symbols corresponding to the digest  $z^*(k)$ , all correct processes manage to (1) reconstruct  $v^*(k)$  after receiving  $t + 1$  such RS symbols via RECONSTRUCT messages, and (2) propose  $v^*(k)$  to the MBA primitive. Then, all correct processes decide  $v^*(k)$  from the MBA primitive due to its strong unanimity property. In other words, all correct processes output valid value  $v^*(k)$  from the R&A mechanism, which ensures that all correct processes decide  $v^*(k)$  from HMOVBA and terminate.

In summary, ensuring that all correct processes adopt the digest  $z^*(k)$  of the leader( $k$ )’s successfully disseminated valid proposal  $v^*(k)$  is a critical step in efficiently solving the MVBA problem against an adaptive adversary. Our algorithms adhere to this principle as well.

## 2.2 Overview of Reducer

To improve the resilience of HMOVBA while maintaining its time and bit complexity, we propose Reducer that operates among  $n = 4t + 1$  processes.

**Key concepts behind Reducer.** In Reducer (Fig. 2, Alg. 1), processes first disseminate their proposals using a phase identical to that of HMOVBA. After the dissemination phase, processes start executing Reducer through iterations. Each Reducer’s iteration  $k$  starts in the same way as HMOVBA’s iterations: (1) correct processes elect the leader of the iteration  $k$  using a common coin, (2) correct processes broadcast their STORED messages containing the digest received from the leader during the dissemination phase, and (3) each correct process waits for  $n - t = 3t + 1$  such STORED messages. To motivate our design choices, we explain how Reducer ensures termination in a good iteration. Hence, for the remainder of this subsection, we fix a good iteration  $k$ .

*Resolving the adoption issue.* As mentioned in §2.1, Reducer (and HMOVBA) cannot ensure that any correct process receives the “good” digest  $z^*(k)$  of the



The main underlying idea of our Reducer algorithm is to *reduce* the number of different candidates across all correct processes in a good iteration to a constant regardless of the adversary’s strategy. Thus, the algorithm’s name.

*Reducing the number of different candidates.* We achieve the reduction in a good iteration  $k$  using an additional “all-to-all” communication step. Specifically, when a correct process  $p_i$  has determined its list of candidates, it disseminates them via a SUGGEST message. If process  $p_i$  includes a digest  $z$  in the aforementioned SUGGEST message, we say that  $p_i$  *suggests*  $z$  in iteration  $k$ . Recall that each correct process suggests at most two digests out of which one is  $z^*(k)$ .

Once process  $p_i$  receives  $n - t = 3t + 1$  SUGGEST messages, process  $p_i$  refines its  $candidates_i$  list. Concretely, for every digest  $z$  suggested by  $p_i$  in iteration  $k$ , process  $p_i$  removes  $z$  from the  $candidates_i$  list unless it receives  $z$  in at least  $(n - t) - t = 2t + 1$  SUGGEST messages. First, this design ensures that  $z^*(k)$  “survives” this communication step as (1) every correct process suggests  $z^*(k)$ , and (2)  $p_i$  hears suggestions of at least  $(n - t) - t = 2t + 1$  correct processes. Second, this design ensures that at most  $3 \in O(1)$  digests “survive” this communication step *across all correct processes*. Let us elaborate on this. Given that each correct process suggests at most two digests out of which one is  $z^*(k)$ , there are (at most)  $n - t = 3t + 1$  suggestions coming from correct processes for adversarial non- $z^*(k)$  digests (assuming there are  $t$  faulty processes). Moreover, each adversarial non- $z^*(k)$  digest that “survives” this step receives (at least)  $(2t + 1) - t = t + 1$  suggestions coming from correct processes. Hence, as  $\frac{3t+1}{t+1} < 3$ , at most two adversarial non- $z^*(k)$  candidates get through the suggestion step. Consequently, a maximum of three candidates, including  $z^*(k)$ , remain.

*Establishing order in the chaos of candidates.* At this point, each correct process has up to two candidate digests (one of which is  $z^*(k)$ ), and across all correct processes there are only up to three different candidate digests. As we will see below, we have at our disposal a special agreement primitive—strong multi-valued Byzantine agreement (SMBA) [37], defined in §4—ensuring that if up to two different digests are proposed by correct processes, then the decided digest is among those proposed by a correct process. The high-level idea now is to invoke this primitive multiple times, and in each invocation correct processes pick the digest to propose from their local candidates in a “smart” way so that: (1) For each invocation, correct processes propose at most two different digests. (2) As a result, correct processes learn from the decided digests about the candidate digests of other correct processes, and can adjust their proposals so that (3) in one of the invocations, all correct processes will inevitably propose  $z^*(k)$ .

Specifically, suppose each correct process  $p_i$  proceeds by sorting its  $candidates_i$  list lexicographically. If  $p_i$  has only one candidate, which must be  $z^*(k)$ ,  $p_i$  duplicates  $z^*(k)$ , resulting in  $candidates_i = [z^*(k), z^*(k)]$ . We say that a correct process  $p_i$  *1-commits* (resp., *2-commits*) a digest  $z$  in iteration  $k$  if  $candidates_i[1] = z$  (resp.,  $candidates_i[2] = z$ ) after the sorting and (potentially) duplicating steps.<sup>5</sup>

---

<sup>5</sup> The list index starts from 1.



For any  $c \in \{1, 2\}$ , let us define the  $\text{committed}(k, c)$  set:

$$\text{committed}(k, c) \equiv \{z \mid z \text{ is } c\text{-committed by a correct process in iteration } k\}.$$

We also say that a correct process  $p_i$  *commits* a digest  $z$  in iteration  $k$  if  $p_i$  1-commits or 2-commits  $z$  in iteration  $k$ . Let us define the  $\text{committed}(k)$  set:

$$\text{committed}(k) \equiv \{z \mid z \text{ is committed by a correct process in iteration } k\}.$$

Note that  $\text{committed}(k) = \text{committed}(k, 1) \cup \text{committed}(k, 2)$ .

First, suppose  $|\text{committed}(k)| = 2$ . Let  $\text{committed}(k) = \{z, z^*(k)\}$ , and assume that, without loss of generality,  $z^*(k)$  is lexicographically smaller than  $z$ . Clearly, each correct process  $p_i$  has its *candidates<sub>i</sub>* list as either  $[z^*(k), z^*(k)]$  or  $[z^*(k), z]$ . In this case, if the correct processes invoke the SMBA primitive twice—first proposing their first committed candidate, and then proposing their second committed candidate—they agree on  $z^*(k)$  during the first invocation. Agreement on  $z^*(k)$  would be sufficient for the processes to reconstruct  $v^*(k)$ , agree on it, and thus terminate.

Now, suppose  $|\text{committed}(k)| = 3$ , with  $\text{committed}(k) = \{z_1, z_2, z^*(k)\}$ . Assume that  $z^*(k)$  is lexicographically smaller than  $z_1$  and  $z_2$ . Then, each correct process  $p_i$  has its *candidates<sub>i</sub>* list as either  $[z^*(k), z^*(k)]$ ,  $[z^*(k), z_1]$ , or  $[z^*(k), z_2]$ . The same procedure as in the case above leads to termination: during the first invocation of the SMBA primitive, correct processes agree on  $z^*(k)$ , then agree on  $v^*(k)$ , and terminate. The above argument naturally carries over to the case where  $z^*(k)$  is lexicographically greater than  $z_1$  and  $z_2$ : the processes agree on  $z^*(k)$  during the second invocation, ensuring termination.

Finally, consider the case where  $\text{committed}(k) = \{z_1, z_2, z^*(k)\}$  and  $z^*(k)$  is lexicographically between  $z_1$  and  $z_2$ . This is where the situation becomes more intricate. Each correct process  $p_i$  now has its *candidates<sub>i</sub>* list as either  $[z^*(k), z^*(k)]$ ,  $[z_1, z^*(k)]$ , or  $[z^*(k), z_2]$ . We employ the same procedure as outlined above: correct processes invoke the SMBA primitive twice, initially proposing their first committed candidate, followed by proposing their second committed candidate. However, in this case, correct processes are not guaranteed to agree on  $z^*(k)$  in any of the two invocations. The only assurance for the correct processes is that the decided digest from the first invocation is either  $z_1$  or  $z^*(k)$ , and the decided digest from the second invocation is either  $z^*(k)$  or  $z_2$ . If either of these two invocations decides  $z^*(k)$ , we are in a favorable position, following the same reasoning as previously discussed.

Finally, we need to deal with the case where the digests decided by the two invocations are  $z_1$  and  $z_2$ , respectively. Our approach is to “duplicate” each of the two invocations, effectively retrying them. Let us elaborate on this. After correct processes agree on  $z_1$  (resp.,  $z_2$ ) during the first (resp., second) invocation, they will “retry” the invocation in the following manner: All correct processes that proposed the decided digest ( $z_1$  in the first, and  $z_2$  in the second invocation) now propose their other committed candidate. All correct processes that proposed a digest other than was decided stick with the same committed candidate. As a result, all correct processes propose  $z^*(k)$  in the “repetition” of both the first and

second invocation. Therefore, agreement on  $z^*(k)$  is ensured, which, following the earlier arguments, implies Reducer’s termination.

It is important to emphasize that repeating both invocations is not necessary; repeating only the first invocation would be sufficient. However, for pedagogical purposes, Reducer repeats both invocations, thereby treating them equally and preserving a symmetrical design.

*Complexity analysis.* As Reducer is guaranteed to terminate in the first good iteration and the probability that each iteration is good is  $\frac{2t+1}{4t+1} \geq \frac{1}{2}$ , Reducer terminates in  $O(1)$  expected time. As correct processes send  $O(n\ell + n^2\lambda \log n)$  bits during the dissemination phase and during each iteration, with Reducer terminating in constantly many iterations, the expected bit complexity is  $O(n\ell + n^2\lambda \log n)$ .

### 2.3 Overview of Reducer++

As already noted, the objective of Reducer++ is to improve Reducer’s resilience while maintaining its quasi-quadratic expected bit complexity and constant expected time complexity. Concretely, Reducer++ solves the MVBA problem with  $n = (3 + \epsilon)t + 1$ , for any fixed  $\epsilon \in (0, 1)$  independent of  $n$ . We begin by discussing why the design of Reducer cannot achieve resilience better than  $t < n/4$ . Following that, we describe how we develop Reducer++ by building upon the foundation of Reducer.

**Reducer below  $n = 4t + 1$ .** Recall that one of the crucial ingredients of the Reducer algorithm is the use of the SMBA primitive that guarantees agreement on a digest proposed by a correct process, as long as at most two different digests are proposed by correct processes. To ensure that the “two-different-proposals” precondition is met in a good iteration  $k$ , Reducer enforces two key constraints: (1) any correct process commits at most two digests in iteration  $k$ , and (2) there are at most three different digests committed across all correct processes (i.e.,  $|\text{committed}(k)| \leq 3$ ). The aforementioned two constraints ensure that the following holds:

$$\exists c \in \{1, 2\} : z^*(k) \in \text{committed}(k, c) \wedge |\text{committed}(k, c)| \leq 2. \quad (\odot)$$

In a nutshell, Reducer cannot improve upon the  $t < n/4$  resilience threshold because Eq.  $(\odot)$  breaks down with improved resilience, as we will see now by examining the behavior of Reducer in a good iteration  $k$  assuming  $n = 4t$ .

*Observation 1: Each correct process can have more than two candidates.* First, we note that when  $n = 4t$ , each correct process may have up to three different candidates after the exchange of STORED messages in good iteration  $k$ . Indeed, as  $n = 4t$ , leader( $k$ ) has successfully disseminated the digest  $z^*(k)$  of its valid proposal  $v^*(k)$  to at least  $(n - t) - t = 2t$  correct processes. Hence, each correct process is guaranteed to hear  $z^*(k)$  only from  $(n - t) + (n - 2t) - n = n - 3t = t$  processes once it receives  $n - t = 3t$  STORED messages in iteration  $k$ . (Recall that each correct process hears  $z^*(k)$  from at least  $n - 3t = t + 1$  correct processes when

$n = 4t + 1$ .) Thus, to ensure that every correct process marks the “good” digest  $z^*(k)$  as a candidate, the “candidate-threshold” is  $n - 3t = t$ . As each correct process waits for  $n - t = 3t$  STORED messages before determining its candidates, each correct process may end up with up to  $\frac{3t}{t} = 3$  different candidates.

*Observation 2: The number of different candidates across correct processes may be greater than three.* To ensure that every correct process commits the “good” digest  $z^*(k)$ , each correct process commits its candidate  $z$  if it hears  $(n - t) - t = 2t$  SUGGEST messages for  $z$ . The fact that each correct process can suggest two different adversarial (non- $z^*(k)$ ) digests implies that the number of different candidates “surviving” the suggestion phase across all correct processes might be greater than three:  $|\text{committed}(k)| > 3$ . (Recall that  $|\text{committed}(k)| \leq 3$  with  $n = 4t + 1$ .) Concretely, it can be shown that  $|\text{committed}(k)| \leq 7$  with  $n = 4t$ .

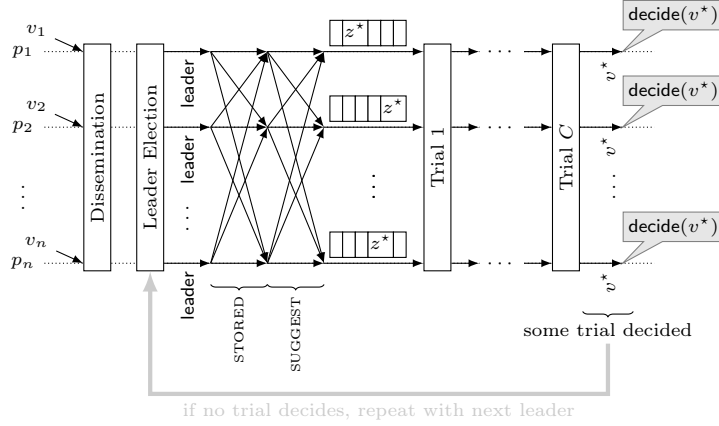
*Observation 3: Reducer among  $n = 4t$  would require an impossible variant of the SMBA primitive.* As  $|\text{committed}(k)| \leq 7$ , Eq. (⊙) does not hold: there might not exist  $c$  such that (1)  $z^*(k) \in \text{committed}(k, c)$ , and (2)  $|\text{committed}(k, c)| \leq 2$ . Let us demonstrate this. Suppose  $\text{committed}(k) = \{z_1, z_2, z^*(k), z_3, z_4\}$  with  $z_1 < z_2 < z^*(k) < z_3 < z_4$  according to the lexicographic order. Let us partition all correct processes into three non-empty and disjoint sets  $\mathcal{S}_1$ ,  $\mathcal{S}_2$ , and  $\mathcal{S}_3$ . The following “spread” of the committed digests is possible:

- Correct processes in the  $\mathcal{S}_1$  set commit  $[z_1, z_2, z^*(k)]$ .
- Correct processes in the  $\mathcal{S}_2$  set commit  $[z_2, z^*(k), z_3]$ .
- Correct processes in the  $\mathcal{S}_3$  set commit  $[z^*(k), z_3, z_4]$ .

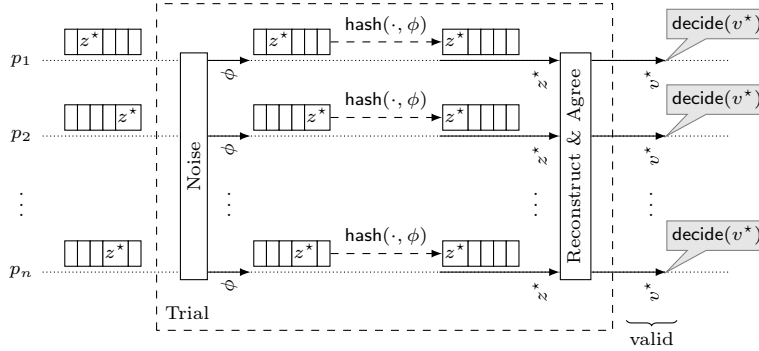
Thus, for any  $c$ , there exist  $3 > 2$  different  $c$ -committed digests, showing that Eq. (⊙) does not hold.

For Reducer to deal with the failure of Eq. (⊙) when  $n = 4t$ , we would need to develop the SMBA primitive with strictly stronger properties than those required for  $n = 4t + 1$ . Specifically, we would need the primitive to satisfy the following guarantee: if up to  $d > 2$  different digests are proposed by correct processes, then the decided digest was proposed by a correct process. Unfortunately, Fitzi and Garay [37] prove that, among  $n = 4t$  processes, such a primitive cannot be implemented even for  $d = 3$ , indicating that Reducer requires substantial design modifications to go below  $n = 4t + 1$ .

**Key concepts behind Reducer++.** Reducer++ (Figs. 3 and 4, Alg. 2) starts in the same way as Reducer. First, processes engage in dissemination identical to that of Reducer (and HMOVBA). The only difference is that, when encoding its proposal  $v_i$  into  $n$  RS symbols, each correct process  $p_i$  treats  $v_i$  as a polynomial of degree  $\epsilon t$  (and not  $t$ , like in Reducer and HMOVBA). Then, Reducer++ proceeds in iterations. Each iteration  $k$  of Reducer++ begins in the same manner as Reducer’s iterations: (1) The leader of iteration  $k$  is elected using a common coin. (2) Each correct process determines its candidates upon receiving  $n - t = (2 + \epsilon)t + 1$  STORED messages. A correct process marks a digest  $z$  as its candidate if it receives  $z$  in (at least)  $n - 3t = \epsilon t + 1$  STORED messages. (3) Each correct process commits some of its candidates upon receiving  $n - t = (2 + \epsilon)t + 1$  SUGGEST messages. Concretely, a correct process commits its candidate digest  $z$  if it receives  $z$  in at



**Fig. 3:** Depiction of Reducer++’s structure. The depiction focuses on a good iteration  $k$  where correct processes decide on the leader( $k$ )’s valid proposal  $v^*(k)$  whose digest is  $z^*(k)$ . See Figs. 1 and 4 for “Dissemination” and “Trial” sub-protocols, respectively. We abridge leader  $\triangleq$  leader( $k$ ),  $z^* \triangleq z^*(k)$ ,  $v^* \triangleq v^*(k)$ .



**Fig. 4:** Depiction of Reducer++’s adoption procedure. The depiction focuses on a case where  $\phi$  happens to be such that the “good” digest  $z^*(k)$  is smallest according to  $hash(\cdot, \phi)$  and is thus adopted by all correct processes. See Fig. 1 for “Reconstruct & Agree” sub-protocol. We abridge leader  $\triangleq$  leader( $k$ ),  $z^* \triangleq z^*(k)$ ,  $v^* \triangleq v^*(k)$ .

least  $(n - t) - t = (1 + \epsilon)t + 1$  SUGGEST messages. This ensures that each correct process commits the “good” digest  $z^*(k)$  in a good iteration  $k$ .

From this point forward, iterations of Reducer++ differ in design from iterations of Reducer. To justify our design choices, we now explain how Reducer++ guarantees termination with *constant probability* in a good iteration. For the rest of the subsection, we focus on a fixed good iteration  $k$ .

*Establishing only constantly many different candidates across correct processes.* Reducer++ guarantees that  $|\text{committed}(k)| \leq C$ , where  $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$ . Let us explain. First, each correct process has at most  $\frac{(2+\epsilon)t+1}{\epsilon t+1} \leq \lceil \frac{3}{\epsilon} \rceil$  candidates after

receiving  $n - t = (2 + \epsilon)t + 1$  STORED messages. Second, as each digest committed by a correct process is suggested by (at least)  $(n - t) - t - t = \epsilon t + 1$  correct processes,  $|\text{committed}(k)| \leq \frac{((3 + \epsilon)t + 1) \lceil \frac{3}{\epsilon} \rceil}{\epsilon t + 1} \leq C$ . (We prove this inequality in §D.)

*Unsuccessfully ensuring termination with constant probability.* The remainder of iteration  $k$  unfolds as follows. Each correct process adopts (in some way) one of its committed digests. If all correct processes adopt the “good” digest  $z^*(k)$ , correct processes decide the leader( $k$ )’s valid proposal  $v^*(k)$  from Reducer++ by relying on the R&A mechanism of Reducer (and HMOVBA) and terminate. Therefore, for Reducer++ to terminate with constant probability in iteration  $k$ , it is crucial to ensure that all correct processes adopt  $z^*(k)$  with constant probability. To accomplish this, each process follows the adoption procedure outlined below.

For the adoption procedure, Reducer++ employs a common coin, denoted by Noise, that returns some random  $\lambda$ -bit value  $\phi$ . Intuitively, each correct process  $p_i$  (1) concatenates each committed digest  $z$  with the obtained random value  $\phi$ , and (2) hashes the concatenation using the hash function `hash` modeled as a random oracle. Concretely, once  $\phi$  is obtained, each correct process  $p_i$  constructs its local set  $H_i$  in the following way:

$$H_i = \{h \mid h = \text{hash}(z, \phi) \wedge z \text{ is committed by } p_i\}.$$

Finally, process  $p_i$  adopts the committed digest  $z'$  that produced the lexicographically smallest hash value:

$$\forall h \in H_i : \text{hash}(z', \phi) \leq h.$$

Let us analyze the probability that all correct processes adopt  $z^*(k)$ . Given that only polynomially many (in  $\lambda$ ) random oracle queries can be made and the common coin outputs a  $\lambda$ -bit random value, the procedure described above emulates a random permutation of the committed digests. Concretely, for every  $z \in \text{committed}(k)$ ,  $\text{hash}(z, \phi)$  is uniformly random with all but negligible probability. Hence, the probability that all correct processes adopt  $z^*(k)$  is equal to the probability that, given the obtained random value  $\phi$ ,  $\text{hash}(z^*(k), \phi)$  is lexicographically smallest in the  $\mathcal{H} = \{h \mid h = \text{hash}(z, \phi) \wedge z \in \text{committed}(k)\}$  set. As all members of  $\mathcal{H}$  are uniformly random (except with negligible probability), this probability is  $\frac{1}{|\mathcal{H}|} \geq \frac{1}{|\text{committed}(k)|} \geq \frac{1}{C}$  given that  $|\mathcal{H}| = |\text{committed}(k)| \leq C$ .<sup>6</sup>

*The problem.* Unfortunately, the approach above has a clear problem. Namely, an adaptive adversary can rig the described probabilistic trial, thus ensuring that not *all* correct processes adopt  $z^*(k)$ . To illustrate why this is the case, we now showcase a simple adversarial attack.

Note that, once the random value  $\phi$  gets revealed, the adversary learns the hash value  $\text{hash}(z^*(k), \phi)$  it needs to “beat”. At this point, the adversary can find an adversarial digest  $z_A$  such that  $\text{hash}(z_A, \phi) < \text{hash}(z^*(k), \phi)$ . Then,

<sup>6</sup> One could obtain a random permutation of digests via a common coin object, thus eliminating the need for the random oracle assumption. However, this would necessitate the coin to disseminate  $2^\lambda \cdot \lambda \gg \lambda$  bits.

by corrupting  $\text{leader}(k)$  and making it disseminate  $z_A$ , the adversary introduces digest  $z_A$  to correct processes. Specifically, by delaying some correct processes and carefully controlling the scheduling of the STORED and SUGGEST messages, the adversary can force some slow correct processes to commit  $z_A$ . As  $\text{hash}(z_A, \phi) < \text{hash}(z^*(k), \phi)$ , these correct processes adopt  $z_A$ , thus preventing termination in good iteration  $k$ . In brief, the adversary is capable of rigging the trial as the set of all digests committed by correct processes is not fixed once the randomness is revealed: upon observing  $\phi$ , the adversary gains the ability to manipulate the trial to its advantage.

*The solution.* Luckily, we can prevent the adversary from manipulating trials “too many” times. The key insight is this: for the adversary to rig a trial, it needs to force correct processes to commit an adversarial digest. Hence, whenever the adversary “cheats”, the number of different digests committed across correct processes increases. However, recall that  $|\text{committed}(k)| \leq C$ . Therefore, given that  $z^*(k)$  is committed by each correct process, the adversary can inject only  $C - 1$  adversarial digests. Roughly speaking, by extending iteration  $k$  to contain  $C$  sequential and independent trials, we ensure the existence of (at least) one trial the adversary cannot rig. More specifically, there exists a trial prior to which the adversary has already injected *all* of its  $C - 1$  adversarial digests. Consequently, even though the adversary might be aware of the “winning” adversarial digest for this trial, it cannot inject it. Thus, this one fair trial indeed provides constant  $\frac{1}{C}$  probability that all correct processes adopt  $z^*(k)$ , which further implies constant  $\frac{1}{C}$  probability that correct processes decide and terminate in good iteration  $k$ .

*Complexity analysis.* Reducer++ terminates in a good iteration with constant  $\frac{1}{C}$  probability. Given that each iteration is good with probability  $\frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \approx \frac{1}{3}$ , Reducer++ terminates in  $O(C)$  iterations in expectation. As each iteration takes  $O(C)$  time (since there are  $C$  trials), the expected time complexity is  $O(C^2)$ .

Correct processes send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase. Additionally, each iteration exchanges  $O(C(n\ell + n^2\lambda \log n))$  bits. As Reducer++ terminates in expected  $O(C)$  iterations, Reducer++ yields an expected bit complexity of  $O(C^2(n\ell + n^2\lambda \log n))$ .

**Reducer++ with optimal  $n = 3t + 1$ .** We conclude the section by explaining why Reducer++ cannot achieve optimal resilience of  $t < n/3$ . One reason is that, when  $t < n/3$ , Reducer++ cannot maintain its quasi-quadratic expected bit complexity. To ensure that the “good” digest  $z^*(k)$  is identified as a candidate by each correct process after receiving  $n - t = 2t + 1$  STORED messages in a good iteration  $k$ , the “candidate threshold” must be set at  $(n - t) + (n - 2t) - n = 1$ . Therefore, each correct process could have *linearly* many candidates after collecting the STORED messages. As a result, disseminating these candidates via the SUGGEST messages would require  $O(n^3\lambda)$  exchanged bits (digests are  $O(\lambda)$  bits), thereby violating the desired quasi-quadratic upper bound. Incorporating the  $\epsilon \cdot t$  gap into the resilience of Reducer++ enforces each correct process to have only *constantly* many candidates after receiving the STORED messages, thus ensuring that the exchange of STORED messages incurs  $o(n^3\lambda)$  bits.

### 3 System Model & Problem Definition

*System model.* We consider a system  $\{p_1, p_2, \dots, p_n\}$  of  $n$  processes. On the one hand, Reducer requires  $n = 4t + 1$ , for any  $t \in \mathbb{N}$ . On the other hand, Reducer++ has better resilience as it requires  $n = (3 + \epsilon)t + 1$ , for any fixed  $\epsilon \in (0, 1)$  and any  $t \in \mathbb{N}$ . Processes are connected through pairwise authenticated channels.

This work considers a computationally bounded and adaptive adversary capable of corrupting up to  $t$  processes throughout (and not only at the beginning of) the protocol execution. Processes not corrupted by the adversary at a certain stage are said to be *so-far-uncorrupted*. Once the adversary corrupts a process, the process falls under the adversary's control and may behave maliciously. We underline that the adversary possesses the *after-the-fact-removal* capabilities: if a so-far-uncorrupted process  $p_i$  sent a message and then got corrupted by the adversary before the message was delivered, the adversary is capable of retracting the message, thus preventing its delivery. A process is said to be *correct* if it is never corrupted; a non-correct process is said to be *faulty*.

We focus exclusively on an asynchronous communication network where message delays are unbounded (but finite). Concretely, we assume that the adversary controls the network: the adversary can intentionally delay messages, but each message exchanged between correct processes must eventually be delivered.

*Multi-valued validated Byzantine agreement (MVBA).* In this paper, we aim to design an MVBA [4,46,35,18,20,41] protocol that operates in the model described above. Informally, MVBA requires correct processes to agree on a *valid*  $\ell$ -bit value. Formally, let  $\text{Value}$  denote the set of all  $\ell$ -bit values. There exists a pre-determined logical predicate  $\text{valid} : \text{Value} \rightarrow \{\text{true}, \text{false}\}$ ; we say that a value  $v$  is *valid* if and only if  $\text{valid}(v) = \text{true}$ . MVBA exposes the following interface:

- **input**  $\text{propose}(v \in \text{Value})$ : a process proposes a value  $v$ .
- **output**  $\text{decide}(v' \in \text{Value})$ : a process decides a value  $v'$ .

Each correct process proposes exactly once and it does so with a valid value. An MVBA protocol satisfies these properties, with all but negligible probability:

- *Agreement*: If a correct process decides a value  $v_1 \in \text{Value}$  and another correct process decides a value  $v_2 \in \text{Value}$ , then  $v_1 = v_2$ .
- *Integrity*: If all processes are correct and a correct process decides a value  $v \in \text{Value}$ , then  $v$  was proposed by a correct process.
- *External validity*: No correct process decides an invalid value  $v \in \text{Value}$ .
- *Termination*: All correct processes eventually decide.

Additionally, we consider the quality property [4] that bounds the probability that the decided value was determined by the adversary:

- *Quality*: If a correct process decides a value  $v \in \text{Value}$ , then the probability that  $v$  is a value determined by the adversary is at most  $q < 1$ .

### 4 Preliminaries

This section overviews the building blocks employed in our algorithms.

*Reed-Solomon codes.* Our algorithms rely on Reed-Solomon (RS) codes [60]. `Reducer` and `Reducer++` use RS as erasure codes; no (substitution-)error correction is required. We use `encode(·)` and `decode(·)` to denote RS' encoding and decoding algorithms. In a nutshell, `encode(v)` takes a value  $v$ , chunks it into the coefficients of a polynomial of degree  $t$  (for `Reducer`) or degree  $\epsilon t$  (for `Reducer++`), and outputs evaluations of the polynomial (RS symbols) at  $n$  (the total number of processes) distinct locations. Similarly, `decode(S)` takes a set of  $t + 1$  (for `Reducer`) or  $\epsilon t + 1$  (for `Reducer++`) RS symbols  $S$  and interpolates them into a polynomial of degree  $t$ , whose coefficients are concatenated and output. The bit-size of an RS symbol obtained by the `encode(v)` algorithm is  $O(\frac{|v|}{n} + \log n)$ , where  $|v|$  denotes the bit-size of value  $v$ .

*Hash functions.* For `Reducer`, we assume a collision-resistant hash function `hash(·)` guaranteeing that a computationally bounded adversary cannot find two different inputs resulting in the same hash value (except with negligible probability). In contrast, `Reducer++` requires hash functions modeled as a random oracle with independent and uniformly distributed hash values. Each hash value is of size  $\lambda$  bits; we assume that  $\lambda \in \omega(\log n)$ .<sup>7</sup>

*Cryptographic accumulators.* A cryptographic accumulator scheme constructs an accumulation value for a set of values and produces a witness for each value in the set. Given the accumulation value and a witness, any process can verify if a value is indeed in the set. More formally, given a parameter  $\lambda$  and a set  $\mathcal{D}$  of  $n$  values  $d_1, \dots, d_n$ , an accumulator has the following components:

- `Gen( $1^\lambda, n$ )`: This algorithm takes a parameter  $\lambda$  represented in the unary form  $1^\lambda$  and an accumulation threshold  $n$  (an upper bound on the number of values that can be accumulated securely); returns a public accumulator key  $ak$ .
- `Eval( $ak, \mathcal{D}$ )`: This algorithm takes an accumulator key  $ak$  and a set of values  $\mathcal{D}$  to be accumulated; returns an accumulation value  $z$  for the set  $\mathcal{D}$ .
- `CreateWit( $ak, z, d_i, \mathcal{D}$ )`: This algorithm takes an accumulator key  $ak$ , an accumulation value  $z$  for  $\mathcal{D}$ , a value  $d_i$  and a set of values  $\mathcal{D}$ ; returns  $\perp$  if  $d_i \notin \mathcal{D}$ , and a witness  $w_i$  if  $d_i \in \mathcal{D}$ .
- `Verify( $ak, z, w_i, d_i$ )`: This algorithm takes an accumulator key  $ak$ , an accumulation value  $z$  for  $\mathcal{D}$ , a witness  $w_i$ , and a value  $d_i$ ; returns *true* if  $w_i$  is the witness for  $d_i \in \mathcal{D}$ , and *false* otherwise.

Concretely, we use Merkle trees [48] as our cryptographic accumulators given they are purely hash-based. Elements of  $\mathcal{D}$  form the leaves of a Merkle tree, the accumulator key is a specific hash function, an accumulation value is the Merkle tree root, and a witness is a Merkle tree proof. Importantly, the size of an accumulation value is  $O(\lambda)$  bits, and the size of a witness is  $O(\lambda \log n)$  bits, where  $\lambda$  denotes the size of a hash value. Throughout the remainder of the paper, we refrain from explicitly mentioning the accumulator key  $ak$  as we assume that the associated hash function is fixed. Moreover, the accumulator scheme is assumed to be collision-free, i.e., for any accumulator key  $ak \leftarrow \text{Gen}(1^\lambda, n)$ ,

<sup>7</sup> Otherwise,  $t$  faulty processes would have computational power exponential in  $\lambda$ .



it is computationally impossible to establish  $(\{d_1, \dots, d_n\}, d', w')$  such that (1)  $d' \notin \{d_1, \dots, d_n\}$ , (2)  $z \leftarrow \text{Eval}(ak, \{d_1, \dots, d_n\})$ , and (3)  $\text{Verify}(ak, z, w', d') = \text{true}$ . For Merkle trees, this property is reduced to the collision resistance of the underlying hash function [48].

Our algorithms instruct each process  $p_i$  to construct a Merkle tree over the RS symbols of its proposal  $v_i$ , with the resulting Merkle root serving as a digest of  $v_i$ . Therefore, throughout the remainder of this paper, we use the terms “accumulation value” and “digest” interchangeably.

*Common coin.* We follow the approach of prior works [11,26,50,66,32,35] and assume the existence of an idealized common coin, an object introduced by Rabin [58], that delivers the same sequence of random coins to all processes. To ensure that the adversary cannot anticipate the coin values in advance, we establish the condition that the value is disclosed only after  $t+1$  processes (thus, at least one correct) have queried the coin. Concretely, both of our algorithms use the common coin objects for (1) obtaining a uniformly random  $\log n$ -bit integer, denoted by  $\text{Election}()$ , and (2) obtaining a uniformly random integer in a specified constant range, denoted by  $\text{Index}()$ .  $\text{Reducer++}$  utilizes an additional common coin object, denoted by  $\text{Noise}()$ , that generates a uniformly random  $\lambda$ -bit value. All coins are independent.

*Multi-valued Byzantine agreement (MBA).* Our algorithms internally utilize the well-known MBA primitive [15,10,55]. MBA is similar to the MVBA primitive: processes propose their values and decide on a common value. Formally, let  $\text{Value}_{\text{MBA}}$  denote the set of values processes can propose and decide. The MBA primitive is associated with the special value  $\perp_{\text{MBA}} \notin \text{Value}_{\text{MBA}}$ . We assume that  $\perp_{\text{MBA}}$  is invalid:  $\text{valid}(\perp_{\text{MBA}}) = \text{false}$ . MBA exposes the following interface:

- **input**  $\text{propose}(v \in \text{Value}_{\text{MBA}})$ : a process proposes a value  $v$ .
- **output**  $\text{decide}(v' \in \text{Value}_{\text{MBA}} \cup \{\perp_{\text{MBA}}\})$ : a process decides a value  $v'$  or the special value  $\perp_{\text{MBA}}$ .

Each process proposes exactly once. The following properties are ensured (except with negligible probability):

- *Agreement*: If a correct process decides a value  $v_1 \in \text{Value}_{\text{MBA}} \cup \{\perp_{\text{MBA}}\}$  and another correct process decides a value  $v_2 \in \text{Value}_{\text{MBA}} \cup \{\perp_{\text{MBA}}\}$ , then  $v_1 = v_2$ .
- *Strong unanimity*: If all correct processes propose the same value  $v \in \text{Value}_{\text{MBA}}$  and a correct process decides a value  $v' \in \text{Value}_{\text{MBA}} \cup \{\perp_{\text{MBA}}\}$ , then  $v' = v$ .
- *Termination*: All correct processes eventually decide.
- *Non-intrusion*: If a correct process decides a value  $v \in \text{Value}_{\text{MBA}}$  (thus,  $v \neq \perp_{\text{MBA}}$ ), then  $v$  was proposed by a correct process.

In contrast to the MVBA primitive, MBA ensures non-intrusion, but it does not guarantee external validity. Note that, whenever correct processes propose different values, MBA might decide futile  $\perp_{\text{MBA}}$ . An MVBA algorithm must decide a valid non- $\perp_{\text{MBA}}$  value in this case, which represents a crucial difference between these two primitives.

Our algorithms utilize MBA algorithms to agree on (1)  $\ell$ -bit values (i.e.,  $\text{Value}_{\text{MBA}} \equiv \text{Value}$ ), and (2) on  $O(\lambda)$ -bit digests (i.e.,  $\text{Value}_{\text{MBA}} \equiv \text{Digest}$ , where

Digest denotes the set of all digests). For agreeing on  $\ell$ -bit values, our algorithms rely on our implementation of the MBA primitive (relegated to §A) with  $O(n\ell + n^2\lambda \log n)$  expected bit complexity and  $O(1)$  expected time complexity. For agreeing on digests (utilized in our implementation of the SMBA primitive; see §B), we rely on the cryptography-free MBA implementation proposed in [55,2] with  $O(n^2\lambda)$  expected bit complexity and  $O(1)$  expected time complexity.

*Strong multi-valued Byzantine agreement (SMBA).* Finally, the Reducer algorithm relies on the SMBA [37] primitive. Concretely, Reducer utilizes SMBA to enable correct processes to agree on a digest. The following interface is exposed:

- **input** propose( $z \in \text{Digest}$ ): a process proposes a digest  $z$ .
- **output** decide( $z' \in \text{Digest}$ ): a process decides a digest  $z'$ .

Each correct process proposes exactly once. Let  $\mathcal{Z}_C$  denote the set of all digests proposed by correct processes. We assume that  $|\mathcal{Z}_C| \in O(1)$ : only constantly many different digests are proposed by correct processes. The following properties are satisfied by the SMBA primitive, with all but negligible probability:

- *Agreement:* If a correct process decides a digest  $z_1 \in \text{Digest}$  and another correct process decides a digest  $z_2 \in \text{Digest}$ , then  $z_1 = z_2$ .
- *Strong validity:* If  $|\mathcal{Z}_C| \leq 2$  and a correct process decides a digest  $z \in \text{Digest}$ , then  $z \in \mathcal{Z}_C$ .
- *Termination:* All correct processes eventually decide.

Intuitively, the SMBA primitive ensures that all correct processes eventually agree on the same digest  $z$ . Moreover, if no more than two different digests are proposed by correct processes (i.e.,  $|\mathcal{Z}_C| \leq 2$ ), the primitive ensures that  $z$  was proposed by a correct process. If correct processes propose three (or more) different digests, a non-proposed digest can be decided.

In Reducer, we utilize our implementation of the SMBA primitive (relegated to §B), which has an expected bit complexity of  $O(n^2\lambda)$  and an expected time complexity of  $O(1)$ . Notably, this represents the first implementation of SMBA with quadratic bit complexity and constant time complexity.<sup>8</sup> Thus, we believe that our implementation is of independent interest.

## 5 Reducer

In this section, we present Reducer, an MVBA algorithm achieving  $O(n\ell + n^2\lambda \log n)$  expected bit complexity and  $O(1)$  expected time complexity. Reducer operates among  $n = 4t + 1$  processes, out of which  $t$  can be corrupted by an adaptive adversary, and relies solely on a collision-resistant hash function.

We start by introducing the implementation of the Reducer algorithm (§5.1). Then, we provide an informal analysis of Reducer’s correctness (§5.2). A formal proof of Reducer’s correctness and complexity can be found in §C.

<sup>8</sup> The binary version of the primitive, where processes propose only 0 or 1 (or any two fixed values), is known to be solvable with quadratic bits in constant time [55,11,21].

---

**Algorithm 1** Reducer: Pseudocode for process  $p_i$  [part 1 of 2]

---

```
1 Uses:
2 SMBA (§B), instances  $\mathcal{SMBA}[k][x][y]$ ,  $\forall k \in \mathbb{N}, \forall x, y \in \{1, 2\}$ 
3 MBA for  $\ell$ -bit values (§A), instances  $\mathcal{MBA}[k][x][y]$ ,  $\forall k \in \mathbb{N}, \forall x, y \in \{1, 2\}$ 
4 Rules:
5 - Process  $p_i$  ignores any received RS symbol without a valid witness.
6 - Process  $p_i$  accepts only one INIT message from each process.
7 Constants:
8 Digest default ▷ default digest
9 Local variables:
10 Value  $v_i \leftarrow p_i$ 's proposal
11 Boolean dissemination_completed $_i \leftarrow false$ 
12 Map(Process  $\rightarrow$  [RS, Digest, Witness]) symbols $_i \leftarrow$  empty map
13 List(Digest) candidates $_i \leftarrow$  empty list ▷ will be reset every iteration
14 Digest adopted_digest $_i \leftarrow \perp$ 
15 List(Value) quasi_decisions $_i \leftarrow$  empty list

16 upon propose(Value  $v_i$ ): ▷ start of the algorithm
17 ▷ dissemination phase starts
18 List(RS)  $[m_1, m_2, \dots, m_n] \leftarrow$  encode( $v_i$ ) ▷ encode( $v_i$ ) treats  $v_i$  as a polynomial of degree  $t$ 
19 Digest  $z_i \leftarrow$  Eval( $\{(1, m_1), (2, m_2), \dots, (n, m_n)\}$ ) ▷ compute the digest
20 for each Process  $p_j$ :
21 Witness  $w_j \leftarrow$  CreateWit( $z_i, (j, m_j), \{(1, m_1), (2, m_2), \dots, (n, m_n)\}$ ) ▷ compute the witness
22 Send (INIT,  $m_j, z_i, w_j$ ) to process  $p_j$ 
23 upon receiving (INIT, RS  $m_i$ , Digest  $z_j$ , Witness  $w_i$ ) from a process  $p_j$ :
24 if dissemination_completed $_i = false$ :
25  $symbols_i[p_j] \leftarrow [m_i, z_j, w_i]$  ▷ store the received RS symbol
26 Send (ACK) to process  $p_j$ 
27 upon receiving (ACK) from  $n - t$  processes (for the first time):
28 Broadcast (DONE)
29 upon receiving (DONE) from  $n - t$  processes (for the first time):
30 Broadcast (FINISH) if  $p_i$  has not broadcast (FINISH) before
31 upon receiving (FINISH) from  $t + 1$  processes (for the first time):
32 Broadcast (FINISH) if  $p_i$  has not broadcast (FINISH) before
```

---

## 5.1 Implementation

The pseudocode of Reducer is given in Alg. 1.

*Pseudocode description.* When processes start executing the Reducer algorithm, they first disseminate their proposals in the *dissemination phase* (lines 16-34). The dissemination phase of Reducer is identical to that of HMOVBA and has already been covered in §2. Briefly, processes disseminate their proposals via INIT messages: each INIT message contains an RS symbol, a digest, and a witness proving the validity of the RS symbol against the digest. When a process receives a valid INIT message, the process stores the content of the message and acknowledges the reception by sending an ACK message back. Once a process receives  $n - t$  ACK messages, it informs all other processes that its proposal is disseminated by broadcasting a DONE message. Upon receiving  $n - t$  DONE message, a process broadcasts a FINISH message; a process may also broadcast a FINISH message upon receiving  $t + 1$  FINISH messages. Lastly, when a process receives  $n - t$  FINISH messages, the process completes the dissemination phase.

After completing the dissemination phase, processes start executing Reducer through *iterations*. Each iteration  $k \in \mathbb{N}$  unfolds as follows (lines 35-73):

---

**Algorithm 1** Reducer: Pseudocode for process  $p_i$  [part 2 of 2]

---

```

33 upon receiving ⟨FINISH⟩ from  $n - t$  processes (for the first time):
34    $dissemination\_completed_i \leftarrow true$  ▷ dissemination phase completes
35   for each  $k = 1, 2, \dots$ : ▷ iteration  $k$  starts
36      $candidates_i \leftarrow$  empty list ▷ reset the list of candidates
37     Process  $leader(k) \leftarrow$  Election() ▷ elect a random leader
38     Broadcast ⟨STORED,  $k$ ,  $symbols_i[leader(k)].digest()$ ⟩ ▷ disseminate the leader's digest
39     wait for  $n - t = 3t + 1$  STORED messages for iteration  $k$ 
40     for each Digest  $z$  included in  $n - 3t = t + 1$  received STORED messages:
41        $candidates_i.append(z)$ 
42     Broadcast ⟨SUGGEST,  $k$ ,  $candidates_i$ ⟩ ▷ disseminate  $p_i$ 's candidates
43     wait for  $n - t = 3t + 1$  SUGGEST messages for iteration  $k$ 
44     for each Digest  $z \in candidates_i$ :
45       if  $z$  is not included in  $n - 2t = 2t + 1$  received SUGGEST messages:
46          $candidates_i.remove(z)$ 
47     if  $candidates_i.size = 0$ : ▷ if no candidate “survives” the suggestion step
48        $candidates_i[1] \leftarrow default$ ;  $candidates_i[2] \leftarrow default$  ▷ commit the default digest
49     else if  $candidates_i.size = 1$ : ▷ if exactly one candidate “survives” the suggestion step
50        $candidates_i[2] \leftarrow candidates_i[1]$  ▷ copy the candidate
51     Sort  $candidates_i$  in the lexicographic order ▷ these digests are committed
52     for each  $x = 1, 2$ :
53        $adopted\_digest_i \leftarrow candidates_i[x]$  ▷ adopt the  $x$ -th committed digest
54       for each  $y = 1, 2$ : ▷ sub-iteration  $(k, x, y)$  starts
55         Digest  $z \leftarrow SMBA[k][x][y].propose(adopted\_digest_i)$ 
56         ▷ Reconstruct & Agree
57         Broadcast ⟨RECONSTRUCT,  $k, x, y$ ,  $symbols_i[leader(k)]$ ⟩
58         wait for  $n - t = 3t + 1$  RECONSTRUCT messages for sub-iteration  $(k, x, y)$ 
59         Set(RS)  $S_i \leftarrow$  the set of all received RS symbols with valid witnesses for digest  $z$ 
60         if  $|S_i| \geq t + 1$ : ▷ check if a value can be decoded using  $S_i$ 
61           Value  $r_i \leftarrow decode(S_i)$  ▷ if yes, set  $r_i$  to the decoded value
62         else
63           Value  $r_i \leftarrow v_i$  ▷ if not, set  $r_i$  to  $p_i$ 's proposal
64         Value  $\cup \{\perp_{MBA}\} v \leftarrow MBA[k][x][y].propose(r_i)$  ▷ propose  $r_i$ 
65         if  $valid(v) = true$ :
66            $quasi\_decisions_i.append(v)$  ▷ quasi-decide  $v$ 
67         if  $z = adopted\_digest_i$ : ▷ is  $p_i$ 's adopted digest decided from  $SMBA[k][x][y]$ ?
68           ▷ if yes,  $p_i$  adopts the other committed digest
69           if  $x = 1$ :  $adopted\_digest_i \leftarrow candidates_i[2]$ 
70           else:  $adopted\_digest_i \leftarrow candidates_i[1]$ 
71     if  $quasi\_decisions_i.size > 0$  and  $p_i$  has not previously decided:
72       Integer  $I \leftarrow Index()$  ▷ obtain a random integer  $I$  in the  $[1, 4]$  range
73       trigger decide( $quasi\_decisions_i[(I \bmod quasi\_decisions_i.size) + 1]$ ) ▷ for quality

```

---

1. Processes randomly elect the iteration's leader, denoted by  $leader(k)$  (line 37).
2. Processes establish their candidate digests through STORED and SUGGEST messages (lines 38-51), as previously discussed in §2.2. Concretely, each process  $p_i$  *commits* up to two candidate digests. Formally, we say that a correct process  $p_i$  *c-commits* a digest  $z$  in iteration  $k$ , for any  $c \in \{1, 2\}$ , if and only if  $candidates_i[c] = z$  when process  $p_i$  reaches line 52. Moreover, we define the  $committed(k, c)$  set, for any  $c \in \{1, 2\}$ , as

$$committed(k, c) = \{z \mid z \text{ is } c\text{-committed by a correct process in iteration } k\}.$$

3. Processes aim to agree on a valid value (lines 52-70). To achieve this, Reducer relies on Eq. (⊙): if iteration  $k$  is good— $leader(k)$  has disseminated proposal  $v^*(k)$  with digest  $z^*(k)$ —the following holds:

$$\exists c \in \{1, 2\} : z^*(k) \in committed(k, c) \wedge |committed(k, c)| \leq 2.$$

Recall that, as discussed in §2.2, it may take two repetitions for processes to agree on  $z^*(k)$  when correct processes propose (to the SMBA primitive) two different digests from the `committed( $k, c$ )` set. In the first repetition, they may decide on a non- $z^*(k)$  digest (the other digest from the `committed( $k, c$ )` set), and only in the second repetition do they succeed in agreeing on  $z^*(k)$ . For this reason, each iteration  $k$  is divided into four sub-iterations  $(k, 1, 1)$ ,  $(k, 1, 2)$ ,  $(k, 2, 1)$  and  $(k, 2, 2)$ . Intuitively, a sub-iteration  $(k, x, y)$  represents the  $y$ -th repetition to reach agreement on the “good” digest  $z^*(k)$  assuming the iteration  $k$  is good and Eq. (⊙) holds for  $x$ .

Each sub-iteration  $(k, x, y)$  unfolds as follows (lines 54-70). Each process  $p_i$  first proposes its adopted digest (i.e., `adopted_digest $_i$` ) to the SMBA primitive. If  $y = 1$ , then the adopted digest is  $p_i$ ’s  $x$ -committed digest (i.e., `adopted_digest $_i$  = candidates $_i$ [ $x$ ]`); otherwise, any committed digest can be the adopted one (as determined by the “proposal-switching” logic at lines 67-70). Once processes agree on a digest  $z$  via SMBA, processes start the R&A mechanism. Specifically, processes disseminate the RS symbols received from `leader( $k$ )` during the dissemination phase. Using the received RS symbols, each correct process  $p_i$  decodes some value  $r_i$  if possible; otherwise,  $p_i$  sets  $r_i$  to its proposal  $v_i$ . Then, each correct process  $p_i$  proposes value  $r_i$  to the MBA primitive. If the decided value  $v$  is valid, each process  $p_i$  quasi-decides  $v$  by appending it to its `quasi_decision $_i$`  list. Lastly, before concluding the sub-iteration, each process  $p_i$  executes the “proposal-switching” logic by checking if its adopted digest is decided from the SMBA primitive. If it is (i.e., if  $z = \text{adopted\_digest}_i$ ),  $p_i$  adopts its other committed digest.

4. The final phase (lines 71-73) of the iteration is designed to ensure the quality property. After completing all four sub-iterations, processes check if any value was quasi-decided. If so, processes obtain a random integer  $I \in [1, 4]$ , which is then used to select one of the previously quasi-decided values for the decision.

## 5.2 Informal Analysis

This subsection provides an informal analysis of `Reducer`’s correctness. Recall that a formal proof can be found in §C.

*Agreement.* The agreement property is ensured by (1) the agreement property of the MBA primitive employed in each iteration (line 64), and (2) the fact that any `Index()` request (line 72) returns the same integer to all correct processes.

*Integrity.* Suppose all processes are correct and a correct process  $p_i$  decides some value  $v$  in an iteration  $k$ . Due to the non-intrusion property of the MBA primitive, some correct process  $p_j$  proposed  $v$  to the MBA primitive in iteration  $k$ . If  $p_j$  decoded  $v$  at line 61,  $v$  is the proposal of `leader( $k$ )`. Otherwise,  $v$  is  $p_j$ ’s proposal (line 63). In any case,  $v$  is the proposal of a correct process.

*External validity.* The property is trivially satisfied due to the check at line 65.

*Termination.* As discussed in §2.2, `Reducer` is guaranteed to terminate in a good iteration. We now formally define what constitutes a good iteration. Let  $p_{\text{first}}$  denote the first correct process that broadcasts a `FINISH` message at line 30. Note

that  $p_{\text{first}}$  broadcasts the FINISH message at line 30 upon receiving a DONE message from  $n - t = 3t + 1$  processes. Let  $\mathcal{D}_{\text{first}}$  denote the set of so-far-uncorrupted processes from which  $p_{\text{first}}$  receives a DONE message before broadcasting the aforementioned FINISH message; note that  $|\mathcal{D}_{\text{first}}| \geq (n - t) - t = 2t + 1$ .

**Definition 1 (Good iterations).** *An iteration  $k \in \mathbb{N}$  is said to be good if and only if  $\text{leader}(k) \in \mathcal{D}_{\text{first}}$ .*

We are now ready to show that Reducer terminates in the first good iteration  $k$ . Let  $v^*(k)$  denote the valid proposal of  $\text{leader}(k)$  and let  $z^*(k)$  denote the digest of  $v^*(k)$ . As  $k$  is a good iteration,  $\text{leader}(k)$  has stored valid RS symbols of its proposal  $v^*(k)$  at  $(n - t) - t = 2t + 1$  correct processes. As discussed in §2 (and proven in §C), Reducer ensures that Eq. (⊙) holds:

$$\exists c \in \{1, 2\} : z^*(k) \in \text{committed}(k, c) \wedge |\text{committed}(k, c)| \leq 2.$$

Consider sub-iteration  $(k, c, 1)$ . As  $|\text{committed}(k, c)| \leq 2$ , correct processes propose at most two different digests to the SMBA primitive (line 55). The strong validity property guarantees that the decided digest  $z$  is proposed by a correct process. We investigate two possibilities:

- Let  $z = z^*(k)$ . In this case, all correct processes decode  $v^*(k)$  (line 61) and propose  $v^*(k)$  to the MBA primitive (line 64). The strong unanimity property of the MBA primitive ensures that all correct processes decide  $v^*(k)$  from it and, thus, quasi-decide  $v^*(k)$  (line 66). Hence, termination is ensured.
- Let  $z \neq z^*(k)$ . In this case, the “proposal-switching” logic (lines 67-70) executed at the end of sub-iteration  $(k, c, 1)$  ensures that all correct processes propose  $z^*(k)$  to the SMBA primitive in sub-iteration  $(k, c, 2)$ . Therefore,  $z^*(k)$  is decided from the SMBA primitive (line 55) in sub-iteration  $(k, c, 2)$ . This implies that all correct processes decode  $v^*(k)$  (line 61) and propose  $v^*(k)$  to the MBA primitive (line 64). Hence, all correct processes quasi-decide  $v^*(k)$  (line 66), showing that Reducer terminates even in this case.

*Quality.* Let  $P_1$  denote the probability that the first iteration is good; note that  $P_1 \geq \frac{2t+1}{4t+1} \geq \frac{1}{2}$ . As seen in the analysis of termination, correct processes are guaranteed to quasi-decide a non-adversarial value (i.e., the leader’s valid proposal) in a good iteration  $k$ . Let  $P_2$  denote the probability that the `Index()` request (line 72) invoked in a good iteration  $k$  that quasi-decides a non-adversarial value indeed selects a non-adversarial quasi-decided value;  $P_2 \geq \frac{1}{4}$  as at most four values (some of which might be the same) can be quasi-decided. Therefore, the probability that the decided value is non-adversarial is (at least) the probability that (1) the first iteration is good, and (2) the `Index()` request chooses a non-adversarial value. Therefore, the probability that an adversarial value is decided is at most  $1 - P_1 \cdot P_2 = \frac{7}{8} < 1$ .

## 6 Reducer++

This section introduces Reducer++, a version of the Reducer algorithm that comes arbitrarily close to optimal one-third resilience. Concretely, Reducer++

operates among  $n = (3 + \epsilon)t + 1$  processes, for any fixed  $\epsilon \in (0, 1)$ , and it achieves  $O(C^2(nl + n^2\lambda \log n))$  expected bit complexity and  $O(C^2)$  expected time complexity, where  $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$  is a constant depending solely on  $\epsilon$ . As a downside compared to Reducer, Reducer++ requires hash functions modeled as a random oracle that uniformly distributes its outputs.

We start by presenting Reducer++’s implementation (§6.1). Then, we give an informal analysis of Reducer++’s correctness (§6.2). A formal proof of Reducer++’s correctness and complexity can be found in §D.

## 6.1 Implementation

The pseudocode of Reducer++ is given in Alg. 2.

*Pseudocode description.* First, processes engage in the dissemination phase (line 18) that is identical to the dissemination phase of Reducer (and HMOVBA). Once the dissemination phase is completed, processes execute Reducer++ through *iterations*. Each iteration  $k \in \mathbb{N}$  proceeds as follows (lines 21-60):

1. Processes elect the leader –  $\text{leader}(k)$  – using a common coin (line 23).
2. Processes determine their candidate digests through STORED and SUGGEST messages (lines 24-32), as explained in §2.3. Formally, we say that a correct process  $p_i$  *commits* a digest  $z$  in iteration  $k$  if and only if  $z$  belongs to the  $\text{candidates}_i$  list when  $p_i$  reaches line 34 in iteration  $k$ . As argued in §2.3 (and proven in §D), given a good iteration  $k$ , (1) each correct process commits up to  $\lceil \frac{3}{\epsilon} \rceil$  digests, and (2) there are at most  $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$  different digests committed across all correct processes (i.e.,  $|\text{committed}(k)| \leq C$ ).
3. Processes aim to agree on a valid value through  $C$  sequential probabilistic trials (lines 34-57). Concretely, we divide iteration  $k$  into  $C$  sub-iterations  $(k, 1), (k, 2), \dots, (k, C)$ . Each sub-iteration  $(k, x \in [1, C])$  represents the  $x$ -th probabilistic trial within iteration  $k$ , and unfolds as follows. First, correct processes obtain a random value  $\phi$  by utilizing a common coin. Then, each correct process  $p_i$  adopts a digest by updating its  $\text{adopted\_digest}_i$  variable:
  - If no digest is committed by  $p_i$ , process  $p_i$  adopts the fixed default digest.
  - Otherwise, process  $p_i$  constructs the  $\text{hashed\_candidates}_i$  list in the following way: (1) For each committed digest  $z$ ,  $p_i$  appends  $\text{hash}(z, \phi)$  to the  $\text{hashed\_candidates}_i$  list. (2) Process  $p_i$  sorts  $\text{hashed\_candidates}_i$  in the lexicographic order. (3) Finally, process  $p_i$  adopts the committed digest  $z'$  that, along with  $\phi$ , produced the smallest element of the sorted  $\text{hashed\_candidates}_i$  list.

We underline that correct processes might adopt different digests.

Once processes adopt their digests, they start the R&A mechanism. Specifically, processes disseminate the RS symbols received from  $\text{leader}(k)$  during the dissemination phase. Each correct process  $p_i$ , if possible, decodes some value  $r_i$  using the received RS symbols that correspond to its adopted digest; if it is impossible to decode any value, process  $p_i$  sets  $r_i$  to its proposal  $v_i$ . Finally, each correct process  $p_i$  proposes value  $r_i$  to the MBA primitive. If the decided value  $v$  is valid, process  $p_i$  quasi-decides  $v$ .

---

**Algorithm 2** Reducer++: Pseudocode for process  $p_i$ 


---

```

1  Uses:
2    MBA for  $\ell$ -bit values (§A), instances  $\mathcal{MBA}[k][x], \forall k, x \in \mathbb{N}$ 
3  Rules:
4    - Process  $p_i$  ignores any received RS symbol without a valid witness.
5    - Process  $p_i$  accepts only one INIT message from each process.
6  Constants:
7    Digest default ▷ default digest
8    Integer  $C = \lceil \frac{12}{\epsilon} \rceil + \lceil \frac{7}{\epsilon} \rceil$  ▷ constant  $C$  is used by all processes
9  Local variables:
10  Value  $v_i \leftarrow p_i$ 's proposal
11  Boolean  $\overline{dissemination\_completed}_i \leftarrow false$ 
12  Map(Process  $\rightarrow$  [RS, Digest, Witness])  $\overline{symbols}_i \leftarrow$  empty map
13  List(Digest)  $\overline{candidates}_i \leftarrow$  empty list ▷ will be reset every iteration
14  List(Hash_Value)  $\overline{hashed\_candidates}_i \leftarrow$  empty list ▷ will be reset every iteration
15  Digest  $\overline{adopted\_digest}_i \leftarrow \perp$ 
16  List(Value)  $\overline{quasi\_decisions}_i \leftarrow$  empty list

17 upon propose(Value  $v_i$ ): ▷ start of the algorithm
18   Execute the dissemination phase identical to that of Reducer (lines 16-34 of Alg. 1)
19 upon receiving (FINISH) from  $n - t$  processes (for the first time):
20    $\overline{dissemination\_completed}_i \leftarrow true$  ▷ dissemination phase completes
21   for each  $k = 1, 2, \dots$ :
22      $\overline{candidates}_i \leftarrow$  empty list;  $\overline{hashed\_candidates}_i \leftarrow$  empty list ▷ reset the candidates
23     Process leader( $k$ )  $\leftarrow$  Election() ▷ elect a random leader
24     Broadcast (STORED,  $k, \overline{symbols}_i[\text{leader}(k)].\text{digest}()$ ) ▷ disseminate the leader's digest
25     wait for  $n - t = (2 + \epsilon)t + 1$  STORED messages for iteration  $k$ 
26     for each Digest  $z$  included in  $n - 3t = \epsilon t + 1$  received STORED messages:
27        $\overline{candidates}_i.\text{append}(z)$ 
28     Broadcast (SUGGEST,  $k, \overline{candidates}_i$ ) ▷ disseminate  $p_i$ 's candidates
29     wait for  $n - t = (2 + \epsilon)t + 1$  SUGGEST messages for iteration  $k$ 
30     for each Digest  $z \in \overline{candidates}_i$ :
31       if  $z$  is not included in  $n - 2t = (1 + \epsilon)t + 1$  received SUGGEST messages:
32          $\overline{candidates}_i.\text{remove}(z)$ 
33   ▷ from this point forward, Reducer++'s design differs from that of Reducer
34   for each  $x = 1, 2, \dots, C$ :
35     ▷ adoption procedure
36     Integer  $\phi \leftarrow \text{Noise}()$  ▷ obtain a random  $O(\lambda)$ -bit value  $\phi$ 
37     if  $\overline{candidates}_i.\text{size} = 0$ :
38        $\overline{adopted\_digest}_i \leftarrow \text{default}$  ▷ no committed digest  $\rightarrow$  adopt default
39     else
40       for each  $j = 1, 2, \dots, \overline{candidates}_i.\text{size}$ :
41          $\overline{hashed\_candidates}_i[j] = \text{hash}(\overline{candidates}_i[j], \phi)$ 
42       ▷ find the smallest hashed candidate and adopt the associated digest
43       Sort  $\overline{hashed\_candidates}_i$  in the lexicographic order
44       Hash_Value  $\overline{smallest}_i \leftarrow \overline{hashed\_candidates}_i[1]$ 
45       Let  $z' \in \overline{candidates}_i$  be the digest such that  $\overline{smallest}_i = \text{hash}(z', \phi)$ 
46        $\overline{adopted\_digest}_i \leftarrow z'$ 
47     ▷ Reconstruct & Agree
48     Broadcast (RECONSTRUCT,  $k, x, \overline{symbols}[\text{leader}(k)]$ )
49     wait for  $n - t = (2 + \epsilon)t + 1$  RECONSTRUCT messages for sub-iteration ( $k, x$ )
50     Set(RS)  $S_i \leftarrow$  the set of received RS symbols with valid witnesses for  $\overline{adopted\_digest}_i$ 
51     if  $|S_i| \geq \epsilon t + 1$ : ▷ check if a value can be decoded using  $S_i$ 
52       Value  $r_i \leftarrow \text{decode}(S_i)$  ▷ if yes, set  $r_i$  to the decoded value
53     else
54       Value  $r_i \leftarrow v_i$  ▷ if not, set  $r_i$  to  $p_i$ 's proposal
55     Value  $\cup \{\perp_{\text{MBA}}\} v \leftarrow \mathcal{MBA}[k][x].\text{propose}(r_i)$  ▷ propose  $r_i$ 
56     if  $\text{valid}(v) = true$ :
57        $\overline{quasi\_decisions}_i.\text{append}(v)$  ▷ quasi-decide  $v$ 
58   if  $\overline{quasi\_decisions}_i.\text{size} > 0$  and  $p_i$  has not previously decided:
59     Integer  $I \leftarrow \text{Index}()$  ▷ obtain a random integer  $I$  in the  $[1, C]$  range
60     trigger  $\text{decide}(\overline{quasi\_decisions}_i[(I \bmod \overline{quasi\_decisions}_i.\text{size}) + 1])$  ▷ for quality

```

---



4. The final phase (lines 58-60) of the iteration ensures the quality property. After finishing all  $C$  sub-iterations, processes check if any value was quasi-decided. If so, processes obtain a random integer  $I \in [1, C]$ , which is then used to select one of the previously quasi-decided values for the decision.

## 6.2 Informal Analysis

This subsection gives an informal analysis of Reducer++’s correctness. A formal proof is relegated to §D. Since the analysis of Reducer++’s agreement, integrity, external validity, and quality follows the arguments made in the analysis of Reducer (see §5.2), we primarily focus on Reducer++’s termination.

*Termination.* Reducer++ ensures termination with constant  $\frac{1}{C}$  probability in any good iteration (see Def. 1). We now provide an informal justification for this statement. Let  $k$  be any good iteration. Let  $v^*(k)$  denote the valid proposal of leader( $k$ ) and let  $z^*(k)$  denote the digest of  $v^*(k)$ . As argued in §2.3, there exists a randomized trial  $\mathcal{T}$  in the good iteration  $k$  that the adversary cannot rig. Concretely, the following holds for some trial  $\mathcal{T}$ :  $\text{start}(\mathcal{T}) = \text{end}(\mathcal{T})$ , where  $\text{start}(\mathcal{T})$  (resp.,  $\text{end}(\mathcal{T})$ ) denotes the set of all digests committed by any correct process before (in global time) the first correct process starts (resp., ends) trial  $\mathcal{T}$ . Then, the probability that all correct processes adopt the “good” digest  $z^*(k)$  and input it to the R&A mechanism (of trial  $\mathcal{T}$ ) before the first correct process ends  $\mathcal{T}$  is  $= \frac{1}{|\text{start}(\mathcal{T})|} = \frac{1}{|\text{end}(\mathcal{T})|} \geq \frac{1}{C}$  as  $\text{start}(\mathcal{T}) = \text{end}(\mathcal{T}) \subseteq \text{committed}(k)$  and  $|\text{committed}(k)| \leq C$ . If this indeed happens, Reducer++ ensures that all correct processes decide  $v^*(k)$  in trial  $\mathcal{T}$  and terminate.

## 7 Related Work

*Byzantine agreement.* Byzantine agreement (BA) was initially studied in synchronous networks [31,43] where message delay is subject to a bound known to the protocol and where deterministic protocols are possible [3,42,49,59,12,23,34]. Subsequently, it was found [36] that in asynchronous networks, where messages arrive eventually but there is no delay bound, deterministic protocols with non-trivial resilience are impossible. This led to the development of various protocols that side-step the impossibility result through different approaches: Protocols for partial synchrony [33] assume that a known delay bound holds *eventually* and such protocols *can* be deterministic [14,20,22,27,33,44,47,65,24]. The condition-based paradigm [51,52,53,5,54,67] relaxes the BA problem to require termination only under optimistic conditions. Last but not least, *randomized* protocols for the asynchronous setting were developed [8,13,63,64,50], in particular for not just binary but multi-valued BA [55,26,56,45,25,62,57].

*Common coins & their implementation.* State-of-the-art asynchronous BA protocols rely on “common coins” [50,32,35,66]. Introduced in [58], they enable processes to obtain unpredictable and unbiased randomness. To implement a common coin, one can rely on a trusted dealer. A common coin can be implemented

without a trusted dealer by utilizing threshold cryptography [19,17]. Moreover, the literature contains some dedicated common coin protocols [30,38,7,6,61], which however often entail heavy hardness and/or setup assumptions.

*MVBA protocols.* The MVBA problem was introduced in [16] alongside a protocol that achieves  $O(1)$  time complexity,  $O(n^2\ell + n^2\lambda + n^3)$  bit complexity, and optimal one-third resilience (see Tab. 1). VABA [4] improves bit complexity to  $O(n^2\ell + n^2\lambda)$  while maintaining optimal resilience and time complexity, as does sMVBA [40]. Dumbo-MVBA [46] further reduces bit complexity to  $O(n\ell + n^2\lambda)$  while retaining  $O(1)$  time complexity and one-third resilience. All these protocols are secure against an adaptive adversary, but rely on threshold cryptography which requires trusted setup (or expensive key generation protocols), is not post-quantum secure, and tends to be slow. These protocols can be converted into hash-based variants (see Tab. 1) by replacing threshold signatures with lists of hash-based signatures from a quorum of processes. This, however, comes with an additional  $O(n^3\lambda)$  term in the resulting bit complexity.

These limitations have sparked growing interest in designing adaptively-secure MVBA protocols that are hash-based from the ground up, the state-of-the-art of which are HMOVBA [35] and FIN-MVBA [32] (see Tab. 1). HMOVBA achieves  $O(n\ell + n^2\lambda \log n)$  bit complexity and  $O(1)$  time complexity, but only sub-optimal one-fifth resilience. FIN-MVBA tolerates up to  $t < n/3$  faults, while also achieving  $O(1)$  time complexity, but only sub-optimal  $O(n^2\ell + n^3\lambda)$  bit complexity. FIN-MVBA itself is an improvement over the hash-based MVBA protocol implied by the distributed key generation protocol of [30,29], which suffers from  $O(\log n)$  time complexity, and is only secure against static adversaries.

*Other BA variants.* Beyond MVBA, BA primitives such as MBA, asynchronous common subset (ACS), and atomic broadcast (ABC) are well-studied in distributed systems. Mostefaoui and Raynal [55] introduced a cryptography-free asynchronous MBA protocol with optimal resilience,  $O(n^2\ell)$  bits and  $O(1)$  time. In [56], a general way of building MBA protocols for long values by “extending” MBA protocols for short values is proposed (both in synchrony and asynchrony).

PACE [66] solves the ACS problem by relying on  $n$  parallel instances of asynchronous binary agreement, thus obtaining  $O(\log n)$  time complexity. Building on FIN-MVBA, [32] presents a hash-based ACS protocol with  $O(1)$  time complexity and  $O(n^2\ell + n^3\lambda)$  bit complexity. The hash-based ACS protocol of [28] also exhibits  $O(1)$  time complexity and  $O(n^3\lambda)$  bit complexity, but is secure only against static adversaries. Importantly, the ACS protocol of [28] does not assume a common coin. Similarly, [1] achieves hash-based ACS with  $O(1)$  time complexity and without assuming a common coin, but only  $1/4$  resilience and  $O(n^5)$  bit complexity; it is worth noting that [1] is safe against an adaptive adversary. Constructing ABC from MVBA is possible, as shown in [16]. ABC constructed from FIN-MVBA has  $O(n^3)$  message complexity, which [62] reduces to  $O(n^2)$  but without reducing the  $O(n^2\ell + n^3\lambda)$  bit complexity any further. All of these protocols have optimal resilience.

## 8 Concluding Remarks

In this work, we have presented two adaptively-secure hash-based asynchronous MVBA algorithms with quasi-quadratic expected bit complexity and constant expected time complexity. In the future, we plan to address the following limitations of our algorithms:

- *Suboptimal resilience*: Our algorithms do not achieve optimal resilience. Hence, it is natural to investigate whether the MVBA problem can be solved efficiently while maintaining optimal resilience.
- *Suboptimal quality*: As proven in [39], an MVBA algorithm achieves optimal quality if the probability that the decided value was chosen by the adversary is at most  $\frac{t}{n-t}$ . However, neither of our algorithms achieves optimal quality.

## Acknowledgment

We thank Pierre Civit, Daniel Collins, Sourav Das, Jason Milionis, and Manuel Vidigueira for fruitful discussions. The work of Jovan Komatovic was conducted in part while at a16z Crypto Research. The research of Tim Roughgarden at Columbia University was supported in part by NSF awards CCF-2006737 and CNS-2212745, and research awards from the Briger Family Digital Finance Lab and the Center for Digital Finance and Technologies.

## References

1. Abraham, I., Asharov, G., Patra, A., Stern, G.: Asynchronous agreement on a core set in constant expected time and more efficient asynchronous VSS and MPC. Cryptology ePrint Archive, Paper 2023/1130 (2023), <https://eprint.iacr.org/2023/1130>
2. Abraham, I., Ben-David, N., Yandamuri, S.: Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In: PODC. pp. 381–391. ACM (2022)
3. Abraham, I., Devadas, S., Nayak, K., Ren, L.: Brief announcement: Practical synchronous byzantine consensus. In: DISC. LIPIcs, vol. 91, pp. 41:1–41:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2017)
4. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: PODC. pp. 337–346. ACM (2019)
5. Attiya, H., Avidor, Z.: Wait-free n-set consensus when inputs are restricted. In: DISC. LNCS, vol. 2508, pp. 326–338. Springer (2002)
6. Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., Papachristoudis, D.: GRand-Line: Adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. Cryptology ePrint Archive, Paper 2023/1887 (2023). <https://doi.org/10.1145/3658644.3690287>, <https://eprint.iacr.org/2023/1887>
7. Bandarupalli, A., Bhat, A., Bagchi, S., Kate, A., Reiter, M.: Random beacons in monte carlo: Efficient asynchronous random beacon without threshold cryptography. Cryptology ePrint Archive, Paper 2023/1755 (2023). <https://doi.org/10.1145/3658644.3670326>, <https://eprint.iacr.org/2023/1755>

8. Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In: PODC. pp. 27–30. ACM (1983)
9. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC. pp. 52–61. ACM (1993)
10. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. *Distributed Comput.* **16**(4), 249–262 (2003)
11. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC. pp. 183–192. ACM (1994)
12. Biely, M., Schmid, U., Weiss, B.: Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.* **412**(40), 5602–5630 (2011)
13. Bracha, G.: Asynchronous byzantine agreement protocols. *Inf. Comput.* **75**(2), 130–143 (1987)
14. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938v3 [cs.DC] (2018), <http://arxiv.org/abs/1807.04938v3>
15. Cachin, C., Guerraoui, R., Rodrigues, L.E.T.: *Introduction to Reliable and Secure Distributed Programming* (2. ed.). Springer (2011)
16. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO. LNCS, vol. 2139, pp. 524–541. Springer (2001)
17. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptol.* **18**(3), 219–246 (2005)
18. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: SRDS. pp. 191–202. IEEE Computer Society (2005)
19. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: STOC. pp. 42–51. ACM (1993)
20. Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.* **20**(4), 398–461 (2002)
21. Chor, B., Moscovici, L.: Solvability in asynchronous environments (extended abstract). In: FOCS. pp. 422–427. IEEE Computer Society (1989)
22. Civit, P., Dzulfikar, M.A., Gilbert, S., Gramoli, V., Guerraoui, R., Komatovic, J., Vidigueira, M.: Byzantine consensus is  $\Theta(n^2)$ : the Dolev-Reischuk bound is tight even in partial synchrony! *Distributed Comput.* **37**(2), 89–119 (2024)
23. Civit, P., Dzulfikar, M.A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M.: DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In: PODC. pp. 145–156. ACM (2024)
24. Civit, P., Dzulfikar, M.A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M., Zabolotchi, I.: Partial synchrony for free? new upper bounds for byzantine agreement. arXiv:2402.10059v4 [cs.DC] (2024), <http://arxiv.org/abs/2402.10059v4>
25. Cohen, R., Forghani, P., Garay, J.A., Patel, R., Zikas, V.: Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In: TCC (4). LNCS, vol. 14372, pp. 422–451. Springer (2023)
26. Crain, T.: Two more algorithms for randomized signature-free asynchronous binary byzantine consensus with  $t < n/3$  and  $O(n^2)$  messages and  $O(1)$  round expected termination. arXiv:2002.08765v1 [cs.DC] (2020), <http://arxiv.org/abs/2002.08765v1>
27. Crain, T., Gramoli, V., Larrea, M., Raynal, M.: DBFT: efficient leaderless byzantine consensus and its application to blockchains. In: NCA. pp. 1–8. IEEE (2018)
28. Das, S., Duan, S., Liu, S., Momose, A., Ren, L., Shoup, V.: Asynchronous consensus without trusted setup or public-key cryptography. *Cryptology ePrint Archive*, Paper 2024/677 (2024). <https://doi.org/10.1145/3658644.3670327>, <https://eprint.iacr.org/2024/677>

29. Das, S., Xiang, Z., Tomescu, A., Spiegelman, A., Pinkas, B., Ren, L.: Verifiable secret sharing simplified. Cryptology ePrint Archive, Paper 2023/1196 (2023), <https://eprint.iacr.org/2023/1196>
30. Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: SP. pp. 2518–2534. IEEE (2022)
31. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
32. Duan, S., Wang, X., Zhang, H.: FIN: practical signature-free asynchronous common subset in constant time. In: CCS. pp. 815–829. ACM (2023)
33. Dwork, C., Lynch, N.A., Stockmeyer, L.J.: Consensus in the presence of partial synchrony. J. ACM **35**(2), 288–323 (1988)
34. Elsheimy, F., Tsimos, G., Papamanthou, C.: Deterministic byzantine agreement with adaptive  $O(n \cdot f)$  communication. In: SODA. pp. 1120–1146. SIAM (2024)
35. Feng, H., Lu, Z., Mai, T., Tang, Q.: Making hash-based MVBA great again. Cryptology ePrint Archive, Paper 2024/479 (2024), <https://eprint.iacr.org/2024/479>
36. Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)
37. Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: PODC. pp. 211–220. ACM (2003)
38. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Efficient asynchronous byzantine agreement without private setups. In: ICDCS. pp. 246–257. IEEE (2022)
39. Goren, G., Moses, Y., Spiegelman, A.: Probabilistic indistinguishability and the quality of validity in byzantine agreement. In: AFT. pp. 111–125. ACM (2022)
40. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding Dumbo: Pushing asynchronous BFT closer to practice. In: NDSS. The Internet Society (2022)
41. Kotla, R., Alvisi, L., Dahlin, M., Clement, A., Wong, E.L.: Zyzzyva: Speculative byzantine fault tolerance. ACM Trans. Comput. Syst. **27**(4), 7:1–7:39 (2009)
42. Kowalski, D.R., Mostéfaoui, A.: Synchronous byzantine agreement with nearly a cubic number of communication bits: synchronous byzantine agreement with nearly a cubic number of communication bits. In: PODC. pp. 84–91. ACM (2013)
43. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
44. Lewis-Pye, A.: Quadratic worst-case message complexity for state machine replication in the partial synchrony model. arXiv:2201.01107v1 [cs.DC] (2022), <http://arxiv.org/abs/2201.01107v1>
45. Li, F., Chen, J.: Communication-efficient signature-free asynchronous byzantine agreement. In: ISIT. pp. 2864–2869. IEEE (2021)
46. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: PODC. pp. 129–138. ACM (2020)
47. Lynch, N.A.: Distributed Algorithms. Morgan Kaufmann (1996)
48. Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO. LNCS, vol. 293, pp. 369–378. Springer (1987)
49. Momose, A., Ren, L.: Optimal communication complexity of authenticated byzantine agreement. In: DISC. LIPIcs, vol. 209, pp. 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)
50. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with  $t < n/3$ ,  $O(n^2)$  messages, and  $O(1)$  expected time. J. ACM **62**(4), 31:1–31:21 (2015)

51. Mostéfaoui, A., Rajsbaum, S., Raynal, M.: Conditions on input vectors for consensus solvability in asynchronous distributed systems. *J. ACM* **50**(6), 922–954 (2003)
52. Mostéfaoui, A., Rajsbaum, S., Raynal, M.: Using conditions to expedite consensus in synchronous distributed systems. In: *DISC. LNCS*, vol. 2848, pp. 249–263. Springer (2003)
53. Mostéfaoui, A., Rajsbaum, S., Raynal, M., Roy, M.: A hierarchy of conditions for consensus solvability. In: *PODC*. pp. 151–160. ACM (2001)
54. Mostéfaoui, A., Rajsbaum, S., Raynal, M., Roy, M.: Condition-based protocols for set agreement problems. In: *DISC. LNCS*, vol. 2508, pp. 48–62. Springer (2002)
55. Mostéfaoui, A., Raynal, M.: Signature-free asynchronous byzantine systems: from multivalued to binary consensus with  $t < n/3$ ,  $O(n^2)$  messages, and constant time. *Acta Informatica* **54**(5), 501–520 (2017)
56. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: *DISC. LIPIcs*, vol. 179, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
57. Patra, A.: Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In: *OPODIS. LNCS*, vol. 7109, pp. 34–49. Springer (2011)
58. Rabin, M.O.: Randomized byzantine generals. In: *FOCS*. pp. 403–409. IEEE Computer Society (1983)
59. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: *PRDC*. pp. 221–228. IEEE Computer Society (2002)
60. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics* **8**(2), 300–304 (1960). <https://doi.org/10.1137/0108018>
61. de Souza, L.F., Kuznetsov, P., Tonkikh, A.: Distributed randomness from approximate agreement. In: *DISC. LIPIcs*, vol. 246, pp. 24:1–24:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022)
62. Sui, X., Wang, X., Duan, S.: Signature-free atomic broadcast with optimal  $O(n^2)$  messages and  $O(1)$  expected time. *Cryptology ePrint Archive*, Paper 2023/1549 (2023), <https://eprint.iacr.org/2023/1549>
63. Toueg, S.: Randomized byzantine agreements. In: *PODC*. pp. 163–178. ACM (1984)
64. Turpin, R., Coan, B.A.: Extending binary byzantine agreement to multivalued byzantine agreement. *Inf. Process. Lett.* **18**(2), 73–76 (1984)
65. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: HotStuff: BFT consensus with linearity and responsiveness. In: *PODC*. pp. 347–356. ACM (2019)
66. Zhang, H., Duan, S.: PACE: fully parallelizable BFT from reposable byzantine agreement. In: *CCS*. pp. 3151–3164. ACM (2022)
67. Zibin, Y.: Condition-based consensus in synchronous systems. In: *DISC. LNCS*, vol. 2848, pp. 239–248. Springer (2003)

## A MBA for $\ell$ -Bit Values

In this section, we present a simple adaptively-secure MBA algorithm for  $\ell$ -bit values (i.e.,  $\text{Value}_{\text{MBA}} = \text{Value}$ ) that exchanges  $O(n\ell + n^2\lambda \log n)$  bits in expectation and terminates in  $O(1)$  time in expectation.

*Graded consensus.* We start by introducing the graded consensus primitive, which we utilize in our MBA algorithm. The graded consensus primitive exposes the following interface:

- **input** `propose`( $v \in \text{Value}$ ): a process proposes a value  $v$ .
- **output** `decide`( $v' \in \text{Value}, g' \in \{0, 1\}$ ): a process decides a value  $v'$  with a binary grade  $g'$ .

Each correct process proposes exactly once. The following properties are ensured:

- *Strong unanimity:* If all correct processes propose the same value  $v \in \text{Value}$  and a correct process decides a pair  $(v' \in \text{Value}, g' \in \{0, 1\})$ , then  $v' = v$  and  $g' = 1$ .
- *Consistency:* If any correct process decides a pair  $(v \in \text{Value}, 1)$ , then no correct process decides any pair  $(v' \neq v, \cdot)$ .
- *Termination:* Every correct process eventually decides.

In our implementation of the MBA primitive, we rely on a *deterministic* implementation [24] of the graded consensus primitive that exchanges  $O(n\ell + n^2\lambda \log n)$  bits and terminates in  $O(1)$  time. This graded consensus implementation operates among  $n \geq 3t+1$  processes and relies solely on a collision-resistant hash function. Observe that, as the implementation is deterministic, it is secure against an adaptive adversary.

*Implementation.* Our implementation of the MBA primitive is given in Alg. 3. We emphasize that our implementation internally relies on two instances  $\mathcal{BA}_1$  and  $\mathcal{BA}_2$  of the binary Byzantine agreement primitive. The binary Byzantine agreement primitive satisfies the same properties as the MBA primitive (see §4); the only difference is that processes can propose only 0 or 1 to the binary Byzantine agreement primitive. The concrete implementation of the binary Byzantine agreement primitive we utilize is the one proposed by Mostefaoui and Raynal [50]; this implementation is safe against an adaptive adversary, exchanges  $O(n^2)$  bits in expectation, terminates in  $O(1)$  time in expectation, and relies on no cryptography.

*Proof of correctness.* We start by proving that Alg. 3 satisfies termination.

**Theorem 1 (Termination).** *Alg. 3 satisfies termination. Concretely, Alg. 3 terminates in  $O(1)$  time in expectation.*

*Proof.* The termination property follows trivially from the termination properties of  $\mathcal{GC}$ ,  $\mathcal{BA}_1$ , and  $\mathcal{BA}_2$ . Moreover, Alg. 3 terminates in  $O(1)$  time in expectation as  $\mathcal{GC}$ ,  $\mathcal{BA}_1$  and  $\mathcal{BA}_2$  terminate in  $O(1)$  time in expectation.  $\square$

A correct process  $p_i$  *adopts* a value  $v$  if and only if `adopted_value` =  $v$  when process  $p_i$  reaches line 9. As Alg. 3 terminates, every correct process adopts some value. The following lemma proves that if a correct process proposes 1 to  $\mathcal{BA}_1$ , then no two correct processes adopt different values.

**Lemma 1.** *If any correct process proposes 1 to  $\mathcal{BA}_1$ , then no two correct processes adopt different values.*

---

**Algorithm 3** MBA for  $\ell$ -bit values: Pseudocode for process  $p_i$ 

---

```
1 Uses:  
2 Graded consensus [24], instance  $\mathcal{GC}$   
3 Binary Byzantine agreement [50], instances  $\mathcal{BA}_1, \mathcal{BA}_2$   
4 Local variables:  
5 Value  $v_i \leftarrow p_i$ 's proposal  
6  $\{0, 1\}$   $ba\_2\_proposal_i \leftarrow \perp$   
  
7 upon propose(Value  $v_i$ ): ▷ start of the algorithm  
8 (Value,  $\{0, 1\}$ ) ( $adopted\_value, g$ )  $\leftarrow \mathcal{GC}.propose(v_i)$   
9  $\{0, 1\}$   $g' \leftarrow \mathcal{BA}_1.propose(g)$   
10 if  $g' = 1$ :  
11   if  $adopted\_value = v_i$ :  
12      $ba\_2\_proposal_i \leftarrow 1$   
13   else  
14      $ba\_2\_proposal_i \leftarrow 0$   
15 else  
16    $ba\_2\_proposal_i \leftarrow 0$   
17  $\{0, 1\}$   $g'' \leftarrow \mathcal{BA}_2.propose(ba\_2\_proposal_i)$   
18 if  $g'' = 1$ :  
19   trigger decide( $adopted\_value$ )  
20 else  
21   trigger decide( $\perp_{MBA}$ )
```

---

*Proof.* By contradiction, suppose that a correct process  $p_i$  adopts a value  $v$  and another correct process  $p_j$  adopts a value  $v' \neq v$ . Therefore, process  $p_i$  decides  $v$  from  $\mathcal{GC}$  (line 8). Similarly, process  $p_j$  decides  $v' \neq v$  from  $\mathcal{GC}$  (line 8).

Let  $p_k$  be a correct process that proposes 1 to  $\mathcal{BA}_1$  (line 9). This means that process  $p_k$  decides  $(v'', 1)$  from  $\mathcal{GC}$  (line 8). The consistency property of  $\mathcal{GC}$  then ensures that  $v'' = v$  and  $v'' = v'$ , which implies  $v = v'$ . We reach a contradiction with  $v \neq v'$ , concluding the proof.  $\square$

Next, we prove that if a correct process proposes 1 to  $\mathcal{BA}_2$  and if a correct process adopts a value  $v$ , then  $v$  is the proposal of a correct process to Alg. 3.

**Lemma 2.** *Suppose a correct process proposes 1 to  $\mathcal{BA}_2$ . If any correct process adopts a value  $v$ , then  $v$  is the proposal of a correct process to Alg. 3.*

*Proof.* Let  $p_i$  be a correct process that proposes 1 to  $\mathcal{BA}_2$  (line 17). We denote by  $adopted(p_i)$  the value  $p_i$  adopts. Due to the check at line 11,  $adopted(p_i)$  is the proposal of  $p_i$  to Alg. 3. Moreover, process  $p_i$  decides 1 from  $\mathcal{BA}_1$  (line 9).

The strong unanimity property of  $\mathcal{BA}_1$  ensures that there exists a correct process  $p_j$  that proposes 1 to  $\mathcal{BA}_1$ . (Otherwise,  $\mathcal{BA}_1$  would decide 0.) Now, consider any correct process  $p_k$  that adopts some value  $v$ . By Lem. 1,  $v = adopted(p_i)$ . Given that  $adopted(p_i)$  is the proposal of process  $p_i$  to Alg. 3, the lemma holds.  $\square$

The following lemma proves that if a correct process proposes 1 to  $\mathcal{BA}_2$ , no two correct processes adopt different values.

**Lemma 3.** *If any correct process proposes 1 to  $\mathcal{BA}_2$ , then no two correct processes adopt different values.*



*Proof.* Let  $p_i$  be a correct process that proposes 1 to  $\mathcal{BA}_2$  (line 17). Hence, process  $p_i$  decides 1 from  $\mathcal{BA}_1$  (line 9). The strong unanimity property of  $\mathcal{BA}_1$  proves that at least one correct process proposes 1 to  $\mathcal{BA}_1$ . Thus, the lemma follows from Lem. 1.  $\square$

We are now ready to prove that Alg. 3 satisfies agreement.

**Theorem 2 (Agreement).** *Alg. 3 satisfies agreement.*

*Proof.* Let  $g^*$  denote the binary value decided from  $\mathcal{BA}_2$  (line 17). We distinguish two cases:

- Let  $g^* = 0$ . In this case, all correct processes that decide do so with  $\perp_{\text{MBA}}$  (line 21). The agreement property is satisfied in this case.
- Let  $g^* = 1$ . In this case, every correct process decides its adopted value (line 19). Moreover, the strong unanimity property of  $\mathcal{BA}_2$  ensures that a correct process proposes 1 to  $\mathcal{BA}_2$ . Therefore, the agreement is satisfied due to Lem. 3.

As agreement is ensured in any possible case, the proof is concluded.  $\square$

Next, we prove strong unanimity.

**Theorem 3 (Strong unanimity).** *Alg. 3 satisfies strong unanimity.*

*Proof.* Suppose all correct processes propose the same value  $v$ . Hence, each correct process  $p_i$  decides  $(v, 1)$  from  $\mathcal{GC}$  (due to its strong unanimity property) and adopts  $v$  (line 8). Therefore, each correct process proposes 1 to  $\mathcal{BA}_1$  (line 9), which ensures that all correct processes decide 1 from  $\mathcal{BA}_1$  (due to its strong unanimity property). Then, each correct process proposes 1 to  $\mathcal{BA}_2$  (as the check at line 11 passes). The strong unanimity property then ensures that all correct processes decide 1 from  $\mathcal{BA}_2$  (line 17). Finally, each correct process then decides its adopted value (line 19), which is  $v$ .  $\square$

Lastly, we prove the non-intrusion property.

**Theorem 4 (Non-intrusion).** *Alg. 3 satisfies non-intrusion.*

*Proof.* Suppose a correct process  $p_i$  decides  $v \neq \perp_{\text{MBA}}$  (line 19). Hence, process  $p_i$  adopts  $v$  and process  $p_i$  decides 1 from  $\mathcal{BA}_2$  (line 17). The strong unanimity property of  $\mathcal{BA}_2$  ensures that a correct process proposes 1 to  $\mathcal{BA}_2$ . Therefore, Lem. 2 implies that  $v$  is the proposal of a correct process to Alg. 3, thus concluding the proof.  $\square$

*Bit complexity.* Finally, we prove the expected bit complexity of Alg. 3.

**Theorem 5 (Bit complexity).** *Alg. 3 exchanges  $O(nl + n^2\lambda \log n)$  bits in expectation.*

*Proof.* Alg. 3 does exchange  $O(nl + n^2\lambda \log n)$  bits in expectation as  $\mathcal{GC}$  exchanges  $O(nl + n^2\lambda \log n)$  bits in expectation and  $\mathcal{BA}_1$  and  $\mathcal{BA}_2$  instances exchange  $O(n^2)$  bits in expectation.  $\square$

## B SMBA: Implementation & Proof

In this section, we present an implementation of the SMBA primitive that attains an expected bit complexity of  $O(n^2\lambda)$  and an expected time complexity of  $O(1)$ . We begin by defining and implementing the collective reliable broadcast (CRB) primitive (§B.1), which serves as a foundation for our SMBA algorithm. Next, we introduce our implementation of the SMBA primitive itself (§B.2). Finally, we provide proof of correctness and complexity for our implementation (§B.3). Recall that SMBA is defined in §4.

### B.1 Collective Reliable Broadcast (CRB)

*Primitive definition.* The definition of the CRB primitive is associated with the special value  $\perp_{\text{CRB}} \notin \text{Digest}$ . We emphasize that the special  $\perp_{\text{CRB}}$  value is different from the special  $\perp_{\text{MBA}}$  value associated with MBA (see §4). The CRB primitive exposes the following interface:

- **input broadcast**( $z \in \text{Digest}$ ): a process broadcasts a digest  $z$ .
- **output deliver**( $z' \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$ ): a process delivers a digest  $z'$  or the special value  $\perp_{\text{CRB}}$ .

Each correct process broadcasts exactly once. Let  $\mathcal{B}_C$  denote the set of digests broadcast by correct processes. We assume that  $|\mathcal{B}_C| \in O(1)$ : only constantly many different digests are broadcast by correct processes. The following properties are satisfied by the CRB primitive:

- *Validity*: If  $|\mathcal{B}_C| \leq 2$ , then no correct process delivers the special value  $\perp_{\text{CRB}}$ .
- *Termination*: All correct processes deliver at least once.
- *Justification*: If a correct process delivers a digest  $z \in \text{Digest}$  (thus,  $z \neq \perp_{\text{CRB}}$ ), then  $z \in \mathcal{B}_C$ .
- *Totality*: If a correct process delivers  $z \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$ , then all correct processes eventually deliver  $z$ .

Observe that a correct process may deliver multiple times. Moreover, the termination property ensures that each correct process delivers at least once.

*Implementation.* Our implementation of the CRB primitive can be found in Alg. 4. We underline that our implementation is heavily inspired by Bracha’s reliable broadcast algorithm [13]. Recall that our CRB primitive tolerates up to  $t < n/4$  Byzantine processes (as it is utilized internally in the Reducer algorithm).

*Proof of correctness.* We start by proving that Alg. 4 satisfies the validity property. To this end, we first show that if there exists a correct process that broadcasts a BROKEN message at line 32, then  $|\mathcal{B}_C| \geq 3$ . (Recall that  $\mathcal{B}_C$  denotes the set of all digests broadcast by correct processes via the CRB primitive.)

**Lemma 4.** *If there exists a correct process that broadcasts a BROKEN message at line 32, then  $|\mathcal{B}_C| \geq 3$ .*

*Proof.* Let  $p_i$  be any correct process that broadcasts a BROKEN message at line 32. By contradiction, suppose  $|\mathcal{B}_C| \leq 2$ . Let *init* denote the set of all INIT

---

**Algorithm 4** CRB: Pseudocode for process  $p_i$ 


---

```

1 Rules:
2 - Process  $p_i$  accepts only one INIT message per process.
3 - Process  $p_i$  broadcasts at most one ECHO message per digest.
4 - Process  $p_i$  broadcasts at most one READY message per digest.
5 - Process  $p_i$  broadcasts at most one BROKEN message.
6 Local variables:
7 Digest  $z_i \leftarrow p_i$ 's broadcast digest
8 Map(Digest  $\rightarrow$  Integer)  $num_i \leftarrow \{0, \text{ for every } z \in \text{Digest}\}$ 
9 Local functions:
10  $\text{distinct}() \equiv |\{z \in \text{Digest} \mid num_i[z] > 0\}|$ .
11  $\text{eliminated}()$ :
12 (1) Let  $Z = \{z \in \text{Digest} \mid num_i[z] > 0\}$ .
13 (2) Sort  $Z$  in the ascending order according to the  $num_i$  map.
14 (3) Return the greatest integer  $x \in \mathbb{N}_{\geq 0}$  such that  $num_i[Z[1]] + \dots + num_i[Z[x]] \leq t$ .

15 upon broadcast(Digest  $z_i$ ): ▷ start of the algorithm
16 Broadcast  $\langle \text{INIT}, z_i \rangle$ 
17 upon exists Digest  $z$  such that  $\langle \text{INIT}, z \rangle$  is received from  $t + 1$  processes:
18 Broadcast  $\langle \text{ECHO}, z \rangle$ 
19 upon exists Digest  $z$  such that  $\langle \text{ECHO}, z \rangle$  is received from  $2t + 1$  processes:
20 Broadcast  $\langle \text{READY}, z \rangle$ 
21 upon exists Digest  $z$  such that  $\langle \text{READY}, z \rangle$  is received from  $t + 1$  processes:
22 Broadcast  $\langle \text{READY}, z \rangle$ 
23 upon exists Digest  $z$  such that  $\langle \text{READY}, z \rangle$  is received from  $2t + 1$  processes:
24 trigger deliver( $z$ )
25 upon  $\langle \text{BROKEN} \rangle$  is received from  $t + 1$  processes:
26 Broadcast  $\langle \text{BROKEN} \rangle$ 
27 upon  $\langle \text{BROKEN} \rangle$  is received from  $2t + 1$  processes:
28 trigger deliver( $\perp_{\text{CRB}}$ )
29 upon receiving  $\langle \text{INIT}, \text{Digest } z \rangle$ :
30  $num_i[z] \leftarrow num_i[z] + 1$ 
31 upon ( $\text{distinct}() - \text{eliminated}() \geq 3$ ) and ( $\geq n - t = 3t + 1$  INIT messages are received):
32 Broadcast  $\langle \text{BROKEN} \rangle$ 

```

---

messages received by  $p_i$  prior to broadcasting the BROKEN message at line 32. Let us define a set  $\mathcal{I}$ :

$$\mathcal{I} = \{(p_j, z) \mid \exists m = \langle \text{INIT}, z \rangle : m \in \text{init} \wedge p_j \text{ is the sender of } m\}.$$

Note that  $|\mathcal{I}| \geq n - t = 3t + 1$  (due to the second condition of the rule at line 31). We now define a set of digests  $\text{correct\_digs}$  in the following way:

$$\text{correct\_digs} \equiv \{z \mid \exists (p_j, z) \in \mathcal{I} : p_j \text{ is a correct process}\}.$$

Let  $X = |\text{correct\_digs}|$ . Note that  $X \in \{1, 2\}$  (as  $|\mathcal{B}_C| \leq 2$ ,  $|\mathcal{I}| \geq 3t + 1$  and there are at most  $t$  faulty processes). We also define a set of digests  $\text{faulty\_digs}$ :

$$\text{faulty\_digs} \equiv \{z \mid \exists (p_j, z) \in \mathcal{I} : p_j \text{ is a faulty process}\}.$$

Note that  $\text{distinct}() = |\text{correct\_digs} \cup \text{faulty\_digs}|$ . Let  $F$  be defined as

$$F = |\{(p_j, z) \mid (p_j, z) \in \mathcal{I} \wedge p_j \text{ is a faulty process}\}|.$$

Note that  $F \leq t$  as process  $p_i$  accepts at most one INIT message per process (line 2) and there are up to  $t$  faulty processes.

Let  $Z$  denote the sorted list of digests constructed by the `eliminated()` function (line 13). We say that a digest  $z$  is *eliminated* if and only if (1)  $z = Z[i]$ , and (2)  $i \in [1, \text{eliminated}()]$ . (If `eliminated() = 0`, no digest is eliminated.)

As process  $p_i$  sends the BROKEN message at line 32, `distinct() - eliminated() ≥ 3` at process  $p_i$  (line 31). Therefore, there are at least three non-eliminated digests. We distinguish two possible scenarios:

- Let  $X = 1$ . Let  $\text{correct\_digs} = \{z\}$ , for some digest  $z$ . Note that  $\text{num}_i[z] \geq 2t + 1$  (as there are at least  $2t + 1$  messages in *init* that are received from correct processes). Therefore,  $z$  cannot be eliminated. Moreover, note that there does not exist a digest  $z' \in \text{correct\_digs} \cup \text{faulty\_digs}$  such that  $z$  precedes  $z'$  in  $Z$ . If such  $z'$  existed,  $\text{num}_i[z'] \geq 2t + 1$ , which further implies that  $z' \in \text{correct\_digs}$ . As  $\text{correct\_digs} = \{z\}$ , this is impossible. Therefore,  $z$  must be the last digest in the list  $Z$ .

For every digest  $z' \in \text{faulty\_digs} \setminus \{z\}$ , all INIT messages for  $z'$  received by  $p_i$  are sent by faulty processes. Hence, every digest  $z' \in \text{faulty\_digs} \setminus \{z\}$  is eliminated. Thus, `distinct() - eliminated() = |\{z\} ∪ faulty_digs| - |faulty_digs \setminus \{z\}| = 1`, which represents a contradiction with the fact that `distinct() - eliminated() ≥ 3`. This means that this case is impossible.

- Let  $X = 2$ . Let  $\text{correct\_digs} = \{z_1, z_2\}$ , for some digests  $z_1$  and  $z_2$ . Note that  $\text{num}_i[z_1] \geq t + 1$  or  $\text{num}_i[z_2] \geq t + 1$  (as there are at least  $2t + 1$  INIT messages received by  $p_i$  from correct processes). Without loss of generality, let  $\text{num}_i[z_1] \geq t + 1$ . Therefore, the digest  $z_1$  cannot be eliminated. Observe also that there cannot exist a digest  $z_3 \in \text{faulty\_digs} \setminus \text{correct\_digs}$  such that  $z_1$  precedes  $z_3$  in  $Z$ . Indeed, for such digest  $z_3$  to exist,  $\text{num}_i[z_3] \geq t + 1$ , which then implies that  $z_3$  must belong to  $\text{correct\_digs}$ . As this is not the case, the only digest that can succeed  $z_1$  in the list  $Z$  is  $z_2$ .

We distinguish two cases:

- Let  $z_2$  not be eliminated. We further study two cases:
  - \* Let  $z_1$  precede  $z_2$  in the sorted list of digests  $Z$ . In this case,  $z_2$  is the last digest in the list  $Z$  and  $z_1$  is the penultimate digest in the list  $Z$ . (This holds as only  $z_2$  can succeed  $z_1$  in  $Z$ .) Crucially, all other digests in the list are eliminated as (1) INIT messages for them are sent only by faulty processes, and (2) there are at most  $t$  faulty processes. Therefore, `distinct() - eliminated() = 2`, which contradicts the fact that `distinct() - eliminated() ≥ 3`. Thus, this case is impossible.

- \* Let  $z_2$  precede  $z_1$  in the sorted list of digests  $Z$ . In this case,  $z_1$  is the last digest in the list. (This holds as only  $z_2$  can succeed  $z_1$  in  $Z$ .) Note that all values that precede  $z_2$  in the list  $Z$  are eliminated (as these are values held by faulty processes only). Let *eliminated* denote the set of eliminated digests. Let  $B_{\text{eliminated}} = \sum_{z \in \text{eliminated}} \text{num}_i[z]$ .

Note that  $B_{\text{eliminated}} \leq t$  due to the definition of the eliminated digests. Importantly, as  $z_2$  is not eliminated, we have  $B_{\text{eliminated}} + \text{num}_i[z_2] \geq t + 1$ , which further implies  $\text{num}_i[z_2] \geq t + 1 - B_{\text{eliminated}}$ .

By contradiction, suppose there exists a digest  $z_3$  such that (1)  $z_2$  precedes  $z_3$  in  $Z$ , and (2)  $z_3$  precedes  $z_1$  in  $Z$ . As  $z_3 \in \text{faulty\_digs} \setminus \text{correct\_digs}$ ,  $B_{\text{eliminated}}$  messages are used on the values preceding  $z_2$  and there are  $F$  messages issued by faulty processes,  $\text{num}_i[z_3] \leq F - B_{\text{eliminated}}$ . Moreover,  $\text{num}_i[z_3] \geq \text{num}_i[z_2]$  (as  $z_2$  precedes  $z_3$  in  $Z$ ). This implies  $\text{num}_i[z_2] \leq F - B_{\text{eliminated}}$ . Hence,  $t + 1 - B_{\text{eliminated}} \leq \text{num}_i[z_2] \leq F - B_{\text{eliminated}}$ , which further implies  $t + 1 \leq F$ . This is impossible as  $F \leq t$ .

- Let  $z_2$  be eliminated. Note that, as  $z_2$  is eliminated and  $z_1$  is not eliminated,  $z_2$  precedes  $z_1$  in list  $Z$ . Given that only  $z_2$  can succeed  $z_1$  in list  $Z$ ,  $z_1$  is the last digest in  $Z$ . As  $\text{distinct}() - \text{eliminated}() \geq 3$  and  $z_1$  is not eliminated, there are at least two digests in  $Z$  that are (1) not eliminated, and (2) not broadcast by correct processes. Let  $z' = Z[\text{eliminated}() + 1]$  and  $z'' = Z[\text{eliminated}() + 2]$ . Note that  $z_2$  precedes both  $z'$  and  $z''$  as  $z_2$  is eliminated and  $z'$  and  $z''$  are not. Moreover, note that all INIT messages sent for  $z'$  or  $z''$  are sent by faulty processes (as  $z', z'' \in \text{faulty\_digs} \setminus \text{correct\_digs}$ ).

Let  $\text{eliminated}$  denote the set of eliminated digests. For each digest  $z \in \text{eliminated}$ , let  $C_z$  (resp.,  $B_z$ ) denote the number of  $\langle \text{INIT}, z \rangle$  messages received by  $p_i$  from correct (resp., faulty) processes. (Hence, for each digest  $z \in \text{eliminated}$ ,  $\text{num}_i[z] = C_z + B_z$ .) Note that  $C_{z_2} > 0$  (as  $z_2 \in \text{correct\_digs}$ ); for every digest  $z \in \text{eliminated} \setminus \{z_2\}$ ,  $C_z = 0$ . Let  $B_{\text{eliminated}} = \sum_{z \in \text{eliminated}} B_z$ . Observe that  $C_{z_2} + B_{\text{eliminated}} + \text{num}_i[z'] \geq t + 1$  as, otherwise,  $z'$  would also be eliminated. Therefore,  $\text{num}_i[z'] \geq t + 1 - C_{z_2} - B_{\text{eliminated}}$ . As  $\text{num}_i[z''] \geq \text{num}_i[z']$ ,  $\text{num}_i[z''] \geq t + 1 - C_{z_2} - B_{\text{eliminated}}$ . Thus,  $\text{num}_i[z'] + \text{num}_i[z''] \geq 2t + 2 - 2C_{z_2} - 2B_{\text{eliminated}}$ . Moreover,  $\text{num}_i[z'] + \text{num}_i[z''] \leq F - B_{\text{eliminated}}$  (as all INIT messages for  $z'$  or  $z''$  are sent by faulty processes and there are already  $B_{\text{eliminated}}$  faulty processes that sent INIT messages for digests different from  $z'$  and  $z''$ ). Hence, we have:

$$2t + 2 - 2C_{z_2} - 2B_{\text{eliminated}} \leq \text{num}_i[z'] + \text{num}_i[z''] \leq F - B_{\text{eliminated}}.$$

This implies  $2t + 2 - 2C_{z_2} - 2B_{\text{eliminated}} \leq F - B_{\text{eliminated}}$ , which implies  $2C_{z_2} \geq 2t + 2 - B_{\text{eliminated}} - F$ .

Observe that  $\text{num}_i[z''] \geq \text{num}_i[z'] \geq C_{z_2}$  (as neither  $z'$  nor  $z''$  are eliminated,  $\text{num}_i[z_2] \geq C_{z_2}$  and  $z_2$  is eliminated). Therefore,  $\text{num}_i[z'] + \text{num}_i[z''] \geq 2C_{z_2}$ . Therefore,  $F - B_{\text{eliminated}} \geq 2C_{z_2}$ .

Thus, we have  $2t + 2 - B_{\text{eliminated}} - F \leq 2C_{z_2} \leq F - B_{\text{eliminated}}$ . Therefore,  $2t + 2 - B_{\text{eliminated}} - F \leq F - B_{\text{eliminated}}$ , which implies  $2t + 2 \leq 2F$ .

As  $F \leq t$ , this is impossible.

The lemma holds as its statement holds in all possible cases.  $\square$

We are now ready to prove that Alg. 4 satisfies the validity property.

**Theorem 6 (Validity).** *Alg. 4 satisfies validity.*

*Proof.* Let  $|\mathcal{B}_C| \leq 2$ . By contradiction, suppose there exists a correct process  $p_i$  that delivers  $\perp_{\text{CRB}}$  (line 28). Therefore, there exists a correct process that broadcasts a BROKEN message. Note that the first correct process that broadcasts a BROKEN message does so at line 32. Hence, Lem. 4 proves that  $|\mathcal{B}_C| \geq 3$ , which implies contradiction with  $|\mathcal{B}_C| \leq 2$ . Thus, the theorem holds.  $\square$

Next, we prove the termination property of Alg. 4. We start by showing that if there exists a digest  $z$  broadcast via Alg. 4 by at least  $t + 1$  correct processes, then every correct process delivers  $z$  in  $O(1)$  message delays.

**Lemma 5.** *Let there exist a digest  $z$  broadcast via Alg. 4 by at least  $t + 1$  correct processes. Then, every correct process delivers  $z$  from Alg. 4 in  $O(1)$  message delays.*

*Proof.* As  $t + 1$  correct processes broadcast  $z$  via Alg. 4, every correct process eventually receives  $t + 1$  INIT messages for  $z$  (line 17) and broadcasts an ECHO message for  $z$  (line 18). (This occurs within a single message delay.) Therefore, every correct process eventually broadcasts a READY message for  $z$  (line 20) upon receiving an ECHO message for  $z$  from (at least)  $2t + 1$  processes (line 19). (This incurs another message delay.) Finally, every correct process eventually receives a READY message for  $z$  from (at least)  $2t + 1$  processes (line 23) and delivers  $z$  (line 24) in  $3 \in O(1)$  message delays.  $\square$

The following lemma proves that all correct processes deliver the special value  $\perp_{\text{CRB}}$  in  $O(1)$  message delays given that (1)  $|\mathcal{B}_C| > 3$ , and (2) no digest is broadcast via Alg. 4 by  $t + 1$  (or more) correct processes.

**Lemma 6.** *Suppose (1)  $|\mathcal{B}_C| > 3$ , and (2) no digest  $z$  is broadcast via Alg. 4 by  $t + 1$  correct processes. Then, every correct process delivers  $\perp_{\text{CRB}}$  in  $O(1)$  message delays.*

*Proof.* To prove the lemma, we prove that all correct processes broadcast a BROKEN message in  $O(1)$  message delays, which then implies that all correct processes receive  $2t + 1$  BROKEN messages (line 27) and deliver  $\perp_{\text{CRB}}$  (line 28) in  $O(1)$  message delays. Let  $|\mathcal{B}_C| = x > 3$ . Without loss of generality, let  $\mathcal{B}_C = \{z_1, z_2, \dots, z_{x-1}, z_x\}$ , for some digests  $z_1, z_2, \dots, z_x$ . Consider any correct process  $p_i$  and time  $\tau$  at which process  $p_i$  receives INIT messages from all correct processes. Note that this occurs within a single message delay. Let  $\mathcal{I}$  denote the set of INIT messages received by  $p_i$  at time  $\tau$ ; note that  $|\mathcal{I}| \geq 3t + 1$  as there are at least  $3t + 1$  correct processes. Next, we define a set of digests *correct\_digs* in the following way:

$$\text{correct\_digs} \equiv \{z \mid z \text{ is received in a message } m \in \mathcal{I} \text{ whose sender is correct}\}.$$

Note that  $\text{correct\_digs} = \{z_1, z_2, \dots, z_{x-1}, z_x\}$ . For each digest  $z \in \text{correct\_digs}$ , let  $C_z$  (resp.,  $B_z$ ) denote the number of  $\langle \text{INIT}, z \rangle$  messages received by  $p_i$  from correct (resp., faulty) processes at time  $\tau$ . Observe that, for every  $z \in \text{correct\_digs}$ ,

$num_i[z] = C_z + B_z$ . Moreover,  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$  as there are at least  $3t + 1$  correct processes.

Let  $Z$  denote the sorted list of digests constructed by the `eliminated()` function (line 13). We say that a digest  $z$  is *eliminated* if and only if (1)  $z = Z[i]$ , and (2)  $i \in [1, \text{eliminated}()]$ . (If `eliminated() = 0`, no digest is eliminated.) To prove the lemma, it suffices to show that at most  $x - 3$  digests from the *correct\_digs* set are eliminated at time  $\tau$  as this (along with the fact that  $|T| \geq 3t + 1$ ) guarantees that the rule at line 31 activates at process  $p_i$ . By contradiction, suppose that  $x - 2$  (or more) digests from the *correct\_digs* set are eliminated. We distinguish three cases:

- Suppose exactly  $x - 2$  digests from the *correct\_digs* set are eliminated. Without loss of generality, let  $z_1, z_2, \dots, z_{x-3}, z_{x-2}$  be eliminated at time  $\tau$ . This implies that  $num_i[z_1] + num_i[z_2] + \dots + num_i[z_{x-2}] = (C_{z_1} + B_{z_1}) + (C_{z_2} + B_{z_2}) + \dots + (C_{z_{x-2}} + B_{z_{x-2}}) \leq t$ , which further implies that  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-2}} \leq t$ . As  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$  and  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-2}} \leq t$ ,  $C_{z_{x-1}} + C_{z_x} \geq 2t + 1$ . Therefore,  $C_{z_{x-1}} \geq t + 1$  or  $C_{z_x} \geq t + 1$ , which contradicts the fact that no digest is broadcast via Alg. 4 by  $t + 1$  correct processes.
- Suppose exactly  $x - 1$  digests from the *correct\_digs* set are eliminated. Without loss of generality, let  $z_1, z_2, \dots, z_{x-3}, z_{x-2}, z_{x-1}$  be eliminated at time  $\tau$ . This implies that  $num_i[z_1] + num_i[z_2] + \dots + num_i[z_{x-2}] + num_i[z_{x-1}] = (C_{z_1} + B_{z_1}) + (C_{z_2} + B_{z_2}) + \dots + (C_{z_{x-2}} + B_{z_{x-2}}) + (C_{z_{x-1}} + B_{z_{x-1}}) \leq t$ , which further implies that  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-2}} + C_{z_{x-1}} \leq t$ . As  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$  and  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-2}} + C_{z_{x-1}} \leq t$ ,  $C_{z_x} \geq 2t + 1$ . This contradicts the fact that no digest is broadcast via Alg. 4 by  $t + 1$  correct processes.
- Suppose exactly  $x$  digests from the *correct\_digs* set are eliminated. This is impossible as  $num[z_1] + num[z_2] + \dots + num[z_x] \geq 3t + 1$  due to the fact that  $C_{z_1} + C_{z_2} + \dots + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$ .

As neither of the aforementioned cases can occur, the lemma holds.  $\square$

We are finally ready to prove the termination property of Alg. 4.

**Theorem 7 (Termination).** *Alg. 4 satisfies termination: all correct processes deliver a digest or the special value  $\perp_{\text{CRB}}$  within  $O(1)$  message delays.*

*Proof.* To prove the theorem, we study two possible scenarios:

- Let  $|\mathcal{B}_C| \leq 3$ . In this case, there exists a digest  $z$  broadcast via Alg. 4 by at least  $t + 1$  correct processes. Therefore, every correct process delivers  $z$  in  $O(1)$  message delays (by Lem. 5). Hence, the termination property is satisfied in this case.
- Let  $|\mathcal{B}_C| > 3$ . We further distinguish two cases:
  - Let there exist a digest  $z$  broadcast via Alg. 4 by  $t + 1$  correct processes. In this case, the termination property is ensured due to Lem. 5.
  - Let there be no digest broadcast via Alg. 4 by  $t + 1$  correct processes. The termination property holds in this case due to Lem. 6.

As termination is satisfied in every possible case, the theorem holds.  $\square$

Next, we prove the justification property.

**Theorem 8 (Justification).** *Alg. 4 satisfies justification.*

*Proof.* Suppose a correct process  $p_i$  delivers a digest  $z \neq \perp_{\text{CRB}}$  from Alg. 4 (line 24). Therefore, there exists a READY message for  $z$  broadcast by a correct process (line 23). Note that the first correct process to broadcast a READY message for  $z$  does so at line 20. Hence, there exists a correct process that broadcasts an ECHO message for  $z$  (line 18), which implies that  $z \in \mathcal{B}_C$  (as at least  $t + 1$  INIT messages for  $z$  are received).  $\square$

Finally, we prove that Alg. 4 satisfies totality.

**Theorem 9 (Totality).** *Alg. 4 satisfies totality: if a correct process  $p_i$  delivers  $z \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$  from Alg. 4, then all correct processes deliver  $z$  within  $O(1)$  message delays from  $p_i$ 's delivery.*

*Proof.* Suppose a correct process  $p_i$  delivers  $z \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$  from Alg. 4. We distinguish two possibilities:

- Let  $z \in \text{Digest}$ . (This implies that  $z \neq \perp_{\text{CRB}}$ .) Hence,  $p_i$  received a  $\langle \text{READY}, z \rangle$  message from (at least)  $2t + 1$  processes, which implies that all correct processes eventually receive (at least)  $t + 1$  READY message for  $z$  (line 21) and broadcast a READY message for  $z$  (line 22). Thus, every correct process receives  $2t + 1$  READY messages for  $z$  (line 23) within  $O(1)$  message delays and delivers  $z$  (line 24).
- Let  $z = \perp_{\text{CRB}}$ . Following the same argument as above (just applied to BROKEN messages), we conclude that every correct process delivers  $\perp_{\text{CRB}}$  (line 28) within  $O(1)$  message delays.

As the statement of the theorem holds in both cases, the proof is concluded.  $\square$

*Bit complexity.* We prove that Alg. 4 exchanges  $O(n^2\lambda)$  bits in expectation, where  $O(\lambda)$  is the bit-size of digests.

**Theorem 10 (Bit complexity).** *Alg. 4 exchanges  $O(n^2\lambda)$  bits in expectation.*

*Proof.* Consider any correct process  $p_i$ . If  $p_i$  broadcasts an ECHO or a READY message for a digest  $z$ , then  $z \in \mathcal{B}_C$ . Moreover,  $p_i$  broadcasts at most one INIT and at most one BROKEN message. Therefore,  $p_i$  sends

$$\underbrace{O(n\lambda)}_{\text{INIT}} + \underbrace{O(n)}_{\text{BROKEN}} + \underbrace{|\mathcal{B}_C| \cdot O(n\lambda)}_{\text{ECHO \& READY}} \text{ bits.}$$

Given that  $|\mathcal{B}_C| \in O(1)$ , process  $p_i$  sends  $O(n\lambda)$  bits, which proves that all correct processes send  $O(n^2\lambda)$  bits in total.  $\square$



---

**Algorithm 5** SMBA: Pseudocode for process  $p_i$ 

---

```
1 Uses:  
2   MBA for digests [55], instances  $\mathcal{MBA}_1, \mathcal{MBA}_2$  ▷ bits:  $O(n^2\lambda)$ ; time:  $O(1)$   
3   CRB (§B.1), instance  $\mathcal{CRB}$  ▷ bits:  $O(n^2\lambda)$ ; time:  $O(1)$   
4 Constants:  
5   Digest default ▷ default digest  
6 Local variables:  
7   Digest  $z_i \leftarrow p_i$ 's proposal  
8   Set(Digest)  $delivered_i \leftarrow \emptyset$   
  
9 upon propose(Digest  $z_i$ ): ▷ start of the algorithm  
10  invoke  $\mathcal{CRB}$ .broadcast( $z_i$ )  
11 upon  $\mathcal{CRB}$ .deliver(Digest  $\cup \{\perp_{\mathcal{CRB}}\}$   $z$ ):  
12    $delivered_i \leftarrow delivered_i \cup \{z\}$   
13   if  $|delivered_i| = 1$ :  
14     invoke  $\mathcal{MBA}_1$ .propose( $z$ )  
15 upon  $\mathcal{MBA}_1$ .decide(Digest  $\cup \{\perp_{\mathcal{CRB}}, \perp_{\mathcal{MBA}}\}$   $z'$ ): ▷ recall that  $\perp_{\mathcal{CRB}} \neq \perp_{\mathcal{MBA}}$   
16   if  $z' \neq \perp_{\mathcal{MBA}}$ :  
17     Digest  $\cup \{\perp_{\mathcal{CRB}}\}$   $z^* \leftarrow z'$   
18     if  $z^* = \perp_{\mathcal{CRB}}$ :  
19        $z^* \leftarrow default$   
20     invoke  $\mathcal{MBA}_2$ .propose( $z^*$ )  
21   else  
22     wait for  $|delivered_i| = 2$   
23     Let  $z^*$  be the lexicographically smallest digest ( $\neq \perp_{\mathcal{CRB}}$ ) in  $delivered_i$   
24     invoke  $\mathcal{MBA}_2$ .propose( $z^*$ )  
25 upon  $\mathcal{MBA}_2$ .decide(Digest  $\cup \{\perp_{\mathcal{MBA}}\}$   $z''$ ):  
26   if  $z'' = \perp_{\mathcal{MBA}}$ :  
27      $z'' \leftarrow default$   
28   trigger decide( $z''$ )
```

---

## B.2 Implementation

Our implementation of the SMBA primitive is given in Alg. 5. Internally, Alg. 5 relies on (1) instances  $\mathcal{MBA}_1$  and  $\mathcal{MBA}_2$  of the MBA primitive (line 2), and (2) an instance  $\mathcal{CRB}$  of the CRB primitive (line 3).

*Pseudocode description.* We explain the pseudocode of our SMBA protocol from the perspective of a correct process  $p_i$ . Once  $p_i$  proposes its digest  $z_i$  (line 9), process  $p_i$  broadcasts  $z_i$  via  $\mathcal{CRB}$  (line 10). When  $p_i$  delivers the first digest (or the special value  $\perp_{\mathcal{CRB}}$ ) from  $\mathcal{CRB}$  (line 11),  $p_i$  proposes it to  $\mathcal{MBA}_1$  (line 14). Let  $p_i$  decide  $z' \in \text{Digest} \cup \{\perp_{\mathcal{CRB}}, \perp_{\mathcal{MBA}}\}$  from  $\mathcal{MBA}_1$  (line 15). We distinguish two cases:

- Let  $z' \neq \perp_{\mathcal{MBA}}$ . Note that the non-intrusion property of  $\mathcal{MBA}_1$  ensures that  $z'$  was proposed by a correct process. We further investigate two scenarios:
  - Let  $z' = \perp_{\mathcal{CRB}}$ . In this case, process  $p_i$  proposes the default digest *default* to  $\mathcal{MBA}_2$  (lines 19 and 20). Note that, as  $\perp_{\mathcal{CRB}}$  is delivered from  $\mathcal{CRB}$ ,  $|\mathcal{Z}_C| > 2$  (due to  $\mathcal{CRB}$ 's validity property). In other words, more than two different digests are proposed by correct processes. Therefore, this case does *not* require the decision to be proposed by a correct process.
  - Let  $z' \neq \perp_{\mathcal{CRB}}$ . Then, process  $p_i$  proposes  $z'$  to  $\mathcal{MBA}_2$  (lines 17 and 20). Observe that the justification property of  $\mathcal{CRB}$  guarantees  $z' \in \mathcal{Z}_C$ , i.e., digest  $z'$  is proposed by a correct process.

- Let  $z' = \perp_{\text{MBA}}$ . The strong unanimity property of  $\text{MBA}_1$  guarantees that not all correct processes proposed the same value  $z \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$  to  $\text{MBA}_1$ . (Otherwise,  $\perp_{\text{MBA}}$  could not have been decided.) Thus, there are at least two different values  $z_1, z_2 \in \text{Digest} \cup \{\perp_{\text{CRB}}\}$  delivered from  $\text{CRB}$  by correct processes. In this case, process  $p_i$  waits to deliver the second digest (or  $\perp_{\text{CRB}}$ ) from  $\text{CRB}$  (line 22). Once that happens (and it will due to  $\text{CRB}$ 's totality property), process  $p_i$  proposes to  $\text{MBA}_2$  the lexicographically smallest digest (thus, not  $\perp_{\text{CRB}}$ !) it has delivered from  $\text{CRB}$  (lines 23 and 24). Finally, once process  $p_i$  decides  $z'' \in \text{Digest} \cup \{\perp_{\text{MBA}}\}$  from  $\text{MBA}_2$  (line 25),  $p_i$  decides from the SMBA protocol (1) the default digest *default* if  $z'' = \perp_{\text{MBA}}$  (lines 27 and 28), or (2)  $z''$  otherwise (line 28).

### B.3 Proof

This subsection formally proves the correctness and complexity of our SMBA implementation (see Alg. 5).

*Agreement.* We start by proving the agreement property of Alg. 5.

**Theorem 11 (Agreement).** *Alg. 5 satisfies agreement.*

*Proof.* The agreement property of Alg. 5 follows directly from the agreement property of the  $\text{MBA}_2$  instance of the MBA primitive (line 25).  $\square$

*Strong validity.* Next, we prove the strong validity property of Alg. 5. Recall that the strong validity property states the following: if  $|\mathcal{Z}_C| \leq 2$  (i.e., at most two different digests are proposed by correct processes) and a correct process decides a digest  $z$ , then  $z \in \mathcal{Z}_C$ . First, we prove that only constantly many different digests are broadcast by correct processes via the  $\text{CRB}$  instance of the CRB primitive. Recall that the CRB primitive requires this in order to satisfy its specification (see §B.1).

**Lemma 7.** *Only  $O(1)$  different digests are broadcast by correct processes via  $\text{CRB}$ .*

*Proof.* The lemma follows directly from the assumption that only  $O(1)$  different digests are proposed to the SMBA primitive by correct processes (see §4).  $\square$

Lem. 7 proves that  $\text{CRB}$  behaves according to its specification. (To not pollute the presentation of the proof, we might not explicitly rely on Lem. 7 in the rest of the proof.) Next, we prove that if  $|\mathcal{Z}_C| \leq 2$ , then all correct processes propose the same digest  $z$  to  $\text{MBA}_2$  such that  $z \in \mathcal{Z}_C$ .

**Lemma 8.** *Let  $|\mathcal{Z}_C| \leq 2$ . Then, there exists a digest  $z$  such that (1)  $z \in \mathcal{Z}_C$ , and (2) every correct process proposes  $z$  to  $\text{MBA}_2$ .*

*Proof.* As  $|\mathcal{Z}_C| \leq 2$ , at most two different values are broadcast via  $\mathcal{CRB}$  by correct processes. Therefore, the validity property of  $\mathcal{CRB}$  guarantees that no correct process delivers the special value  $\perp_{\mathcal{CRB}}$  from  $\mathcal{CRB}$ . Moreover, the termination property of  $\mathcal{CRB}$  ensures that all correct processes eventually deliver from  $\mathcal{CRB}$  and, thus, propose to  $\mathcal{MBA}_1$  (line 14). Hence, the termination and agreement properties of  $\mathcal{MBA}_1$  ensure that all correct processes eventually decide  $z' \in \text{Digest} \cup \{\perp_{\mathcal{CRB}}, \perp_{\mathcal{MBA}}\}$  from  $\mathcal{MBA}_1$ . We distinguish two cases:

- Let  $z' \neq \perp_{\mathcal{MBA}}$ . The non-intrusion property of  $\mathcal{MBA}_1$  guarantees that  $z'$  was proposed to  $\mathcal{MBA}_1$  by a correct process. Therefore,  $z'$  was delivered by a correct process from  $\mathcal{CRB}$ , which further implies that (1)  $z' \neq \perp_{\mathcal{CRB}}$ , and (2)  $z' \in \mathcal{Z}_C$  due to  $\mathcal{CRB}$ 's justification property. This means that every correct process proposes  $z'$  to  $\mathcal{MBA}_2$  (line 20). Hence, the statement of the lemma holds in this case.
- Let  $z' = \perp_{\mathcal{MBA}}$ . The strong unanimity property of  $\mathcal{MBA}_1$  proves that at least two different values from the  $\text{Digest} \cup \{\perp_{\mathcal{CRB}}\}$  set are proposed to  $\mathcal{MBA}_1$  by correct processes. Let  $z_1$  denote one such value and let  $z_2 \neq z_1$  denote another. Both  $z_1$  and  $z_2$  are delivered from  $\mathcal{CRB}$  by correct processes, which implies that  $z_1 \neq \perp_{\mathcal{CRB}}$  and  $z_2 \neq \perp_{\mathcal{CRB}}$ . Moreover, the justification property of  $\mathcal{CRB}$  guarantees that  $z_1 \in \mathcal{Z}_C$  and  $z_2 \in \mathcal{Z}_C$ . As  $z_1 \neq z_2$  and  $|\mathcal{Z}_C| \leq 2$ ,  $\mathcal{Z}_C = \{z_1, z_2\}$ .

Now, consider any correct process  $p_i$ . As both  $z_1$  and  $z_2$  are delivered by correct processes from  $\mathcal{CRB}$ , process  $p_i$  eventually delivers these two digests (due to  $\mathcal{CRB}$ 's totality property). Moreover, process  $p_i$  never delivers (1) any other digest  $z_3 \neq \perp_{\mathcal{CRB}}$  from  $\mathcal{CRB}$  as  $\mathcal{CRB}$ 's justification would imply  $z_3 \in \mathcal{Z}_C$  (recall that  $\mathcal{Z}_C = \{z_1, z_2\}$ ), and (2)  $\perp_{\mathcal{CRB}}$  as no correct process ever delivers  $\perp_{\mathcal{CRB}}$  due to  $\mathcal{CRB}$ 's validity property (recall that  $|\mathcal{Z}_C| \leq 2$ ). Hence,  $\text{delivered}_i = \{z_1, z_2\}$  when  $|\text{delivered}_i| = 2$  at process  $p_i$  (line 22). Thus,  $p_i$  (and every other correct process) proposes the lexicographically smaller digest between  $z_1$  and  $z_2$ , which implies that the statement of the lemma holds even in this case.

The lemma holds as its statement is satisfied in both possible scenarios.  $\square$

Finally, we are ready to prove the strong validity property of Alg. 5.

**Theorem 12 (Strong validity).** *Alg. 5 satisfies strong validity.*

*Proof.* Let  $|\mathcal{Z}_C| \leq 2$ . Let  $p_i$  be any correct process that decides some digest  $z'$  from Alg. 5 (line 28). Lem. 8 shows that there exists a digest  $z$  such that (1)  $z \in \mathcal{Z}_C$ , and (2) all correct processes propose  $z$  to  $\mathcal{MBA}_2$ . Therefore, the strong unanimity property of  $\mathcal{MBA}_2$  ensures that all correct processes decide  $z \neq \perp_{\mathcal{MBA}}$  from  $\mathcal{MBA}_2$ . Thus,  $z' = z$ , which implies  $z' \in \mathcal{Z}_C$ .  $\square$

*Termination.* Finally, we prove the termination property of Alg. 5.

**Theorem 13 (Termination).** *Alg. 5 satisfies termination. Concretely, Alg. 5 terminates in  $O(1)$  expected time.*

*Proof.* The  $\mathcal{CRB}$  primitive guarantees that all correct processes propose to  $\mathcal{MBA}_1$  within  $O(1)$  time (by Thm. 7). Therefore, the termination and agreement properties of  $\mathcal{MBA}_1$  guarantee that all correct processes eventually decide some  $z' \in \text{Digest} \cup \{\perp_{\text{CRB}}, \perp_{\text{MBA}}\}$ . Given that  $\mathcal{MBA}_1$  terminates in  $O(1)$  expected time, all correct processes decide from  $\mathcal{MBA}_1$  in  $O(1)$  expected time in Alg. 5. We now differentiate two cases:

- Let  $z' \neq \perp_{\text{MBA}}$ . In this case, all correct processes propose to  $\mathcal{MBA}_2$  within  $O(1)$  time in expectation.
- Let  $z' = \perp_{\text{MBA}}$ . The strong unanimity property of  $\mathcal{MBA}_1$  proves that at least two different values from the  $\text{Digest} \cup \{\perp_{\text{CRB}}\}$  set have been proposed to  $\mathcal{MBA}_1$  by correct processes. Therefore, there are at least two different values delivered from  $\mathcal{CRB}$  by correct processes. The totality property guarantees that all correct processes eventually deliver (at least) two values from  $\mathcal{CRB}$ , and they do so in additional  $O(1)$  time (by Thm. 9). Thus, all correct processes eventually propose to  $\mathcal{MBA}_2$  even in this case. Concretely, all correct processes propose to  $\mathcal{MBA}_2$  within  $O(1)$  time in expectation.

As all correct processes propose to  $\mathcal{MBA}_2$  within  $O(1)$  time in expectation (in any of the two cases), the termination property of  $\mathcal{MBA}_2$  and its  $O(1)$  expected time complexity ensure that all correct processes decide from  $\mathcal{MBA}_2$  in additional  $O(1)$  time in expectation. Thus, Alg. 5 indeed terminates in  $O(1)$  expected time, which concludes the proof.  $\square$

*Bit complexity.* Lastly, we prove that Alg. 5 exchanges  $O(n^2\lambda)$  bits in expectation. Recall that  $O(\lambda)$  is the bit-size of a digest.

**Theorem 14 (Bit complexity).** *Alg. 5 achieves  $O(n^2\lambda)$  expected bit complexity.*

*Proof.* The theorem follows directly from the fact that  $\mathcal{MBA}_1$ ,  $\mathcal{MBA}_2$  and  $\mathcal{CRB}$  instances exchange  $O(n^2\lambda)$  bits in expectation.  $\square$

## C Reducer: Proof

In this section, we formally prove the correctness and complexity of Reducer.

*External validity.* We start by proving the external validity property.

**Theorem 15 (External validity).** *Reducer (Alg. 1) satisfies external validity.*

*Proof.* The property holds as every value quasi-decided by a correct process is valid due to the check at line 65.  $\square$

*Agreement.* Next, we prove agreement. We say that a correct process  $p_i$  *quasi-decides* a vector  $vec$  in an iteration  $k \in \mathbb{N}$  if and only if  $quasi\_decisions_i = vec$  when process  $p_i$  reaches line 71 in iteration  $k$ . We now prove that, for every iteration  $k \in \mathbb{N}$ , different vectors cannot be quasi-decided by correct processes.

**Lemma 9.** *Let  $k \in \mathbb{N}$  be any iteration. Suppose a correct process  $p_i$  quasi-decides a vector  $vec_i$  in iteration  $k$  and another correct process  $p_j$  quasi-decides a vector  $vec_j$  in iteration  $k$ . Then,  $vec_i = vec_j$ .*

*Proof.* The lemma follows from the agreement property of the  $\mathcal{MBA}[k][x][y]$  instance (of the MBA primitive), for every  $x \in \{1, 2\}$  and every  $y \in \{1, 2\}$ .  $\square$

We are now ready to prove that Reducer ensures agreement.

**Theorem 16 (Agreement).** *Reducer (Alg. 1) satisfies agreement.*

*Proof.* By contradiction, suppose (1) there exists a correct process  $p_i$  that decides a value  $v_i$ , and (2) there exists a correct process  $p_j$  that decides a value  $v_j \neq v_i$ . Let  $p_i$  (resp.,  $p_j$ ) decide  $v_i$  (resp.,  $v_j$ ) in some iteration  $k_i \in \mathbb{N}$  (resp.,  $k_j \in \mathbb{N}$ ). Therefore, process  $p_i$  (resp.,  $p_j$ ) quasi-decides  $v_i$  (resp.,  $v_j$ ) in iteration  $k_i$  (resp.,  $k_j$ ). Without loss of generality, let  $k_i \leq k_j$ .

As process  $p_i$  quasi-decides  $v_i$  in iteration  $k_i$ , process  $p_i$  quasi-decides a vector  $vec_i$  in iteration  $k_i$ ; note that  $v_i$  belongs to  $vec_i$ . By Lem. 9, process  $p_j$  also quasi-decides the non-empty vector  $vec_i$  in iteration  $k_i$ . We separate two cases:

- Let  $k_i = k_j$ . Due to the fact that the `Index()` request invoked in iteration  $k_i = k_j$  returns the same integer to all correct processes, we have that  $v_j = v_i$ . Thus, we reach a contradiction with  $v_j \neq v_i$  in this case.
- Let  $k_i < k_j$ . As  $p_j$  quasi-decides the non-empty vector  $vec_i$  in iteration  $k_i$ , we reach a contradiction with the fact that  $p_j$  decides in iteration  $k_j > k_i$ .

As neither of the above cases can occur, the proof is concluded.  $\square$

*Integrity.* To prove integrity, we first show that if any correct process proposes a value  $v$  to the  $\mathcal{MBA}[k][x][y]$  instance, for any sub-iteration  $(k, x, y)$ , and all processes are correct, then  $v$  is the proposal of a correct process to Reducer.

**Lemma 10.** *Let  $(k \in \mathbb{N}, x \in \{1, 2\}, y \in \{1, 2\})$  be any sub-iteration and let all processes be correct. If any correct process  $p_i$  proposes a value  $v$  to  $\mathcal{MBA}[k][x][y]$ , then  $v$  is the proposal of a correct process to Reducer.*

*Proof.* Recall that `leader(k)` denotes the leader of iteration  $k$ . We distinguish two cases:

- Let  $p_i$  execute line 61. In this case, process  $p_i$  has received (at least)  $t + 1$  RS symbols. Given that all processes are correct, all these RS symbols are sent by `leader(k)` during the dissemination phase and they all correspond to `leader(k)`'s proposal to Reducer. Therefore,  $v$  is the proposal of `leader(k)` to Reducer, which proves the statement of the lemma in this case.
- Let  $p_i$  execute line 63. The statement of the lemma trivially holds in this case as  $v$  is  $p_i$ 's proposal to Reducer.

As the statement of the lemma holds in both cases, the proof is concluded.  $\square$

The following theorem proves that Reducer satisfies integrity.

**Theorem 17 (Integrity).** *Reducer (Alg. 1) satisfies integrity.*

*Proof.* Suppose all processes are correct. Moreover, let a correct process  $p_i$  decide some value  $v$ ; note that value  $v$  must be valid by Thm. 15. Hence, process  $p_i$  quasi-decides  $v$  in some iteration  $k \in \mathbb{N}$ , which further implies that  $v$  is decided from  $\mathcal{MBA}[k][x][y]$  in some sub-iteration  $(k, x \in \{1, 2\}, y \in \{1, 2\})$ . Given that  $v$  is valid and  $\perp_{\text{MBA}}$  is invalid,  $v \neq \perp_{\text{MBA}}$ . Thus, the non-intrusion property of  $\mathcal{MBA}[k][x][y]$  guarantees that  $v$  was proposed to  $\mathcal{MBA}[k][x][y]$  by a correct process. Lem. 10 then proves that  $v$  is the proposal of a correct process to Reducer, which concludes the proof.  $\square$

*Termination.* Next, we prove that Reducer satisfies termination. We say that a correct process *completes the dissemination phase* if and only if the process executes line 34. The following lemma proves that at least one correct process completes the dissemination phase (and thus starts the first iteration of Reducer).

**Lemma 11.** *At least one correct process completes the dissemination phase.*

*Proof.* By contradiction, suppose no correct process completes the dissemination phase. Hence, no correct process stops responding with ACK messages (line 26) upon receiving INIT messages (line 23). As there are (at least)  $n - t$  correct processes, every correct process eventually broadcasts a DONE message (line 28). Similarly, every correct process eventually receives  $n - t$  DONE messages (line 29) and broadcasts a FINISH message (line 30). Thus, every correct process eventually receives a FINISH message from (at least)  $n - t$  processes (line 33) and completes the dissemination phase (line 34), thus contradicting the fact that no correct process completes the dissemination phase.  $\square$

Next, we prove that if a correct process completes the dissemination phase, then all correct processes complete the dissemination phase eventually.

**Lemma 12.** *If a correct process completes the dissemination phase, then every correct process eventually completes the dissemination phase.*

*Proof.* Let  $p_i$  be any correct process that completes the dissemination phase. This implies that  $p_i$  receives  $n - t = 3t + 1$  FINISH messages (line 33), out of which (at least)  $2t + 1$  messages are sent by correct processes. Therefore, every correct process eventually receives  $2t + 1 \geq t + 1$  FINISH messages (line 31) and broadcasts its FINISH message (line 32). Given that there are (at least)  $n - t$  correct processes, every correct process eventually receives  $n - t$  FINISH messages (line 33) and completes the dissemination phase (line 34).  $\square$

We are ready to prove that every correct process eventually completes the dissemination phase.

**Lemma 13.** *Every correct process eventually completes the dissemination phase.*

*Proof.* The lemma follows directly from Lems. 11 and 12.  $\square$

Recall that the specification of the SMBA primitive (see §4) assumes that only  $O(1)$  different proposals are input by correct processes. Hence, to prove that the  $SMBA$  instances utilized in Reducer operate according to their specification, we now prove that only  $O(1)$  different proposals are input by correct processes to any  $SMBA$  instance. Recall that we say that a correct process  $p_i$  *suggests* a digest  $z$  in an iteration  $k \in \mathbb{N}$  if and only if  $p_i$  broadcasts a SUGGEST message with digest  $z$  in iteration  $k$  (line 42). Let  $\text{suggested}_i(k)$  denote the set of digests suggested by any correct process  $p_i$  in any iteration  $k \in \mathbb{N}$ . The following lemma proves that each correct process suggests at most two digests in every iteration.

**Lemma 14.** *For every correct process  $p_i$  and every iteration  $k \in \mathbb{N}$ , the following holds:  $|\text{suggested}_i(k)| \leq 2$ .*

*Proof.* For every digest  $z \in \text{suggested}_i(k)$ , process  $p_i$  receives (at least)  $n - 3t = t + 1$  STORED messages in iteration  $k$  (due to the check at line 40). As  $p_i$  receives  $n - t = 3t + 1$  STORED messages (line 39) before broadcasting its SUGGEST message, there can be at most  $\frac{3t+1}{t+1} < 3$  suggested digests.  $\square$

Recall that we say a correct process  $p_i$  *1-commits* (resp., *2-commits*) a digest  $z$  in an iteration  $k \in \mathbb{N}$  if and only if  $\text{candidates}_i[1] = z$  (resp.,  $\text{candidates}_i[2] = z$ ) when process  $p_i$  reaches line 52 in iteration  $k$ . Similarly, a correct process  $p_i$  *commits* a digest  $z$  in an iteration  $k \in \mathbb{N}$  if and only if  $p_i$  1-commits or 2-commits  $z$  in iteration  $k$ . We denote by  $\text{committed}_i(k)$  the set of digests a correct process  $p_i$  commits in an iteration  $k \in \mathbb{N}$ . The following lemma proves that any correct process  $p_i$  commits only digests previously suggested by  $p_i$  (in the same iteration) or the default digest *default* (see line 8).

**Lemma 15.** *For every correct process  $p_i$  and every iteration  $k \in \mathbb{N}$ , the following holds:  $\text{committed}_i(k) \subseteq (\text{suggested}_i(k) \cup \{\text{default}\})$ .*

*Proof.* Consider any digest  $z \in \text{committed}_i(k)$ . To prove the lemma, it suffices to show that if  $z \in \text{committed}_i(k)$  and  $z \neq \text{default}$ , then  $z \in \text{suggested}_i(k)$ . By contradiction, suppose (1)  $z \in \text{committed}_i(k)$ , (2)  $z \neq \text{default}$ , and (3)  $z \notin \text{suggested}_i(k)$ . Let us consider three possibilities:

- Let  $\text{candidates}_i.\text{size} = 0$  when  $p_i$  reaches line 47. This case is impossible as  $p_i$  commits only *default* in this case, i.e.,  $z \neq \text{default}$  is not committed.
- Let  $\text{candidates}_i.\text{size} = 1$  when  $p_i$  reaches line 47. As  $z \notin \text{suggested}_i(k)$ , this case is also impossible as  $z$  is not committed by  $p_i$ .
- Let  $\text{candidates}_i.\text{size} = 2$  when  $p_i$  reaches line 47. Again, this case cannot occur as  $z$  cannot be committed given that  $z \notin \text{suggested}_i(k)$ .

As neither of the three cases can occur, the lemma holds.  $\square$

Next, we prove that if any correct process commits a digest  $z \neq \text{default}$  in any iteration  $k$ , then (at least)  $t + 1$  correct processes suggest  $z$  in iteration  $k$ .

**Lemma 16.** *Consider any correct process  $p_i$  and any iteration  $k \in \mathbb{N}$ . If process  $p_i$  commits a digest  $z \neq \text{default}$ , then (at least)  $t + 1$  correct processes suggest  $z$  in iteration  $k$ .*

*Proof.* If process  $p_i$  commits  $z \neq \text{default}$ , Lem. 15 proves that  $z \in \text{suggested}_i(k)$ . Hence, for process  $p_i$  to not remove  $z$  from its  $\text{candidates}_i$  list after receiving  $n - t = 3t + 1$  SUGGEST messages (line 43),  $p_i$  must have received at least  $2t + 1$  SUGGEST messages for  $z$  (line 45). Therefore, at least  $2t + 1 - t = t + 1$  correct processes suggest  $z$  in iteration  $k$ .  $\square$

Recall that, for any iteration  $k \in \mathbb{N}$ , we define the set  $\text{committed}(k)$ :

$$\text{committed}(k) = \{z \mid z \text{ is committed by a correct process in iteration } k\}.$$

The following lemma proves that  $|\text{committed}(k)| \in O(1)$ , for every iteration  $k$ .

**Lemma 17.** *For every iteration  $k \in \mathbb{N}$ ,  $|\text{committed}(k)| \in O(1)$ .*

*Proof.* Lem. 16 proves that every committed digest  $z \neq \text{default}$  is suggested by at least  $t + 1$  correct processes. Moreover, Lem. 14 proves that each correct process suggests at most two digests in each iteration. Therefore, there can be at most  $\frac{2(4t+1)}{t+1} = \frac{8t+2}{t+1} < 8$  non-*default* digests that are committed by correct processes in iteration  $k$ . Finally, as the special default digest *default* can also be committed,  $|\text{committed}(k)| < 9$ , which concludes the proof.  $\square$

Finally, we are ready to prove that only  $O(1)$  different digests are proposed by correct processes to any  $\mathcal{SMBA}$  instance.

**Lemma 18.** *Let  $(k \in \mathbb{N}, x \in \{1, 2\}, y \in \{1, 2\})$  be any sub-iteration. Then, only  $O(1)$  different digests are proposed to  $\mathcal{SMBA}[k][x][y]$  by correct processes.*

*Proof.* If a correct process  $p_i$  proposes a digest  $z$  to  $\mathcal{SMBA}[k][x][y]$  (line 55), then  $p_i$  commits  $z$  in iteration  $k$  (lines 53, 69 and 70). Hence,  $z \in \text{committed}(k)$ . As  $|\text{committed}(k)| \in O(1)$  (by Lem. 17), the proof is concluded.  $\square$

We now prove that if all correct processes start any iteration  $k$ , all correct processes eventually start sub-iteration  $(k, 1, 1)$ .

**Lemma 19.** *Let  $k \in \mathbb{N}$  be any iteration such that all correct processes start iteration  $k$ . Then, all correct processes eventually start sub-iteration  $(k, 1, 1)$ .*

*Proof.* As all correct processes start iteration  $k$  and there are at least  $n - t = 3t + 1$  correct processes, all correct processes eventually receive  $n - t = 3t + 1$  STORED messages (line 39) and broadcast a SUGGEST message (line 42). Therefore, all correct processes eventually receive  $n - t = 3t + 1$  SUGGEST messages (line 43) and start sub-iteration  $(k, 1, 1)$  at line 52.  $\square$

Next, we prove that if all correct processes start any sub-iteration  $(k, x, y)$ , then all correct processes eventually complete sub-iteration  $(k, x, y)$ .

**Lemma 20.** *Let  $(k \in \mathbb{N}, x \in \{1, 2\}, y \in \{1, 2\})$  be any sub-iteration such that all correct processes start sub-iteration  $(k, x, y)$ . Then, all correct processes eventually complete sub-iteration  $(k, x, y)$ .*



*Proof.* Given that all correct processes start sub-iteration  $(k, x, y)$ , all correct processes propose to  $\mathcal{SMBA}[k][x][y]$  (line 55). By Lem. 18,  $\mathcal{SMBA}[k][x][y]$  behaves according to its specification (see §4). Thus, the termination property of  $\mathcal{SMBA}[k][x][y]$  ensures that all correct processes eventually decide from  $\mathcal{SMBA}[k][x][y]$ . Then, all correct processes broadcast a RECONSTRUCT message (line 57), thus ensuring that every correct process eventually receives  $n-t = 3t+1$  RECONSTRUCT messages (as there are at least  $n-t \geq 3t+1$  correct processes). Hence, all correct processes eventually propose to  $\mathcal{MBA}[k][x][y]$  (line 64). The termination property of  $\mathcal{MBA}[k][x][y]$  ensures that all correct processes complete sub-iteration  $(k, x, y)$ , thus concluding the proof.  $\square$

The following lemma proves that every sub-iteration is eventually started and completed by all correct processes.

**Lemma 21.** *Every iteration and sub-iteration are eventually started and completed by all correct processes.*

*Proof.* By Lem. 13, all correct processes start iteration 1. Therefore, Lem. 19 proves that all correct processes start sub-iteration  $(1, 1, 1)$ . By inductively applying Lems. 19 and 20, we prove that every sub-iteration is eventually started and completed by all correct processes.  $\square$

To not pollute the presentation, we might not explicitly rely on Lem. 21 in the rest of the proof. Recall that, by Def. 1, an iteration  $k \in \mathbb{N}$  is good if and only if  $\text{leader}(k) \in \mathcal{D}_{\text{first}}$ . Recall that  $|\mathcal{D}_{\text{first}}| \geq n - 2t = 2t + 1$ . Moreover, recall that, for every good iteration  $k$ , (1)  $v^*(k)$  denotes the valid proposal of  $\text{leader}(k)$ , and (2)  $z^*(k)$  denotes the digest of  $v^*(k)$ . The lemma below proves that every correct process suggests  $z^*(k)$  in every good iteration  $k$ .

**Lemma 22.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, every correct process suggests  $z^*(k)$  in iteration  $k$ .*

*Proof.* As  $\text{leader}(k) \in \mathcal{D}_{\text{first}}$ ,  $\text{leader}(k)$  stores  $z^*(k)$  at  $n-t = 3t+1$  processes in the dissemination phase, out of which at most  $t$  can be faulty. Thus,  $\text{leader}(k)$  stores  $z^*(k)$  at  $\geq 2t+1$  correct processes. This further implies that each correct process receives  $z^*(k)$  in STORED messages from at least  $t+1$  processes, which means that each correct process broadcasts a SUGGEST message with  $z^*(k)$  (due to the check at line 40) and, thus, suggests  $z^*(k)$  in iteration  $k$ .  $\square$

Next, we prove that every correct process commits  $z^*(k)$  in every good iteration  $k$ .

**Lemma 23.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, every correct process commits  $z^*(k)$  in iteration  $k$ .*

*Proof.* By Lem. 22, all correct processes suggest  $z^*(k)$  in iteration  $k$ . Hence, for each correct process  $p_i$ ,  $\text{candidates}_i[1] = z^*(k)$  or  $\text{candidates}_i[2] = z^*(k)$  when process  $p_i$  broadcasts a SUGGEST message (line 42). Furthermore, each correct

process  $p_i$  receives a SUGGEST message with  $z^*(k)$  from at least  $n - t - t = 2t + 1$  processes, which means that  $p_i$  does not remove  $z^*(k)$  from  $\text{candidates}_i$  at line 46. Hence, the lemma holds.  $\square$

The following lemma proves that if any correct process commits any digest (including *default*) in any good iteration  $k$ , then (at least)  $2t + 1 - f$  correct processes suggest  $z$  in iteration  $k$ , where  $f \leq t$  denotes the *actual* number of faulty processes.

**Lemma 24.** *Consider any correct process  $p_i$  and any good iteration  $k \in \mathbb{N}$ . If process  $p_i$  commits a digest  $z$  ( $z$  might be equal to *default*), then (at least)  $2t + 1 - f$  correct processes suggest  $z$  in iteration  $k$ , where  $f \leq t$  denotes the actual number of faulty processes.*

*Proof.* By Lem. 22, all correct processes suggest  $z^*(k)$  in iteration  $k$ . Thus, when process  $p_i$  reaches line 47,  $\text{candidates}_i.\text{size} > 0$  as  $\text{candidates}_i[1] = z^*(k)$  or  $\text{candidates}_i[2] = z^*(k)$ . Hence, process  $p_i$  does not execute line 48. Therefore, for each digest  $z$  that process  $p_i$  commits,  $z$  is suggested by at least  $2t + 1 - f$  correct processes in iteration  $k$  (due to the check at line 45).  $\square$

The following lemma proves that  $|\text{committed}(k)| \leq 3$  in every good iteration.

**Lemma 25.** *Let  $k \in \mathbb{N}$  be any good iteration  $k$ . Then,  $|\text{committed}(k)| \leq 3$ .*

*Proof.* Lem. 23 proves that all correct processes commit  $z^*(k)$  in iteration  $k$ . Thus,  $z^*(k) \in \text{committed}(k)$ . To prove the lemma, we analyze the cardinality of the  $\text{committed}(k) \setminus \{z^*(k)\}$  set; let  $X = |\text{committed}(k) \setminus \{z^*(k)\}|$ .

For every digest  $z \in \text{committed}(k) \setminus \{z^*(k)\}$ , Lem. 24 proves that  $z$  is suggested by (at least)  $2t + 1 - f$  correct processes in iteration  $k$ . By Lem. 22, all correct processes suggest  $z^*(k)$  in iteration  $k$ . Given that each correct process suggests at most two digests (by Lem. 14), there are at most  $n - f$  “correct suggestions” for non- $z^*(k)$  digests. By contradiction, suppose  $X \geq 3$ . Therefore, there are at least  $X(2t + 1 - f) \geq 3(2t + 1 - f) = 6t + 3 - 3f$  “correct suggestions” for non- $z^*(k)$ . Thus, we have  $n - f = 4t + 1 - f \geq 6t + 3 - 3f$ , which implies  $2f \geq 2t + 2$  and  $f \geq t + 1$ . This is impossible as  $f \leq t$ , which means  $X < 3$ , thus concluding the proof of the lemma.  $\square$

For every iteration  $k \in \mathbb{N}$  and every  $x \in \{1, 2\}$ , let us define the  $\text{committed}(k, x)$  set in the following way:

$$\text{committed}(k, x) = \{z \mid z \text{ is } x\text{-committed by a correct process in iteration } k\}.$$

Observe that the following holds: (1)  $\text{committed}(k, 1) \subseteq \text{committed}(k)$ , and (2)  $\text{committed}(k, 2) \subseteq \text{committed}(k)$ . The following lemma proves that there exists  $c \in \{1, 2\}$  in every good iteration  $k$  such that (1)  $z^*(k) \in \text{committed}(k, c)$ , and (2)  $|\text{committed}(k, c)| \leq 2$ . In other words, the following lemma proves that Eq. (⊙) is correct.

**Lemma 26.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, there exists  $c \in \{1, 2\}$  such that (1)  $z^*(k) \in \text{committed}(k, c)$ , and (2)  $|\text{committed}(k, c)| \leq 2$ .*

*Proof.* Recall that Lem. 23 proves that every correct process commits  $z^*(k)$  in iteration  $k$ . (Thus,  $z^*(k) \in \text{committed}(k)$ .) To prove the lemma, we consider three possible cases:

- Let  $z^*(k)$  be the lexicographically smallest digest in  $\text{committed}(k)$ . In this case, every correct process 1-commits  $z^*(k)$  in iteration  $k$ . (Otherwise,  $z^*(k)$  could not be the smallest digest among  $\text{committed}(k)$ .) Thus, the following holds:  $\text{committed}(k, 1) = \{z^*(k)\}$ .
- Let  $z^*(k)$  be the lexicographically greatest digest in  $\text{committed}(k)$ . In this case, every correct process 2-commits  $z^*(k)$  in iteration  $k$ . Thus, the statement of the lemma holds in this case as well.
- Let  $z^*(k)$  not be the lexicographically smallest nor the lexicographically greatest digest in  $\text{committed}(k)$ . Hence, there exist digests  $z_1$  and  $z_2$  in  $\text{committed}(k)$  such that (1)  $z_1 < z^*(k)$ , and (2)  $z^*(k) < z_2$ . (Lem. 25 then shows that  $\text{committed}(k) = \{z_1, z^*(k), z_2\}$ .) As (1)  $z_2 \in \text{committed}(k)$ , (2) every correct process commits  $z^*(k) < z_2$  in iteration  $k$  (by Lem. 23), and (3) each correct process commits at most two different digests, there exists a correct process that 1-commits  $z^*(k)$ , i.e.,  $z^*(k) \in \text{committed}(k, 1)$ . Moreover, since (1) every correct process commits  $z^*(k)$  in iteration  $k$  (by Lem. 23), and (2)  $z^*(k) < z_2$ , no correct process 1-commits  $z_2$  in iteration  $k$ . Therefore,  $z^*(k) \in \text{committed}(k, 1)$  and  $|\text{committed}(k, 1)| \leq 2$ , which concludes the proof in this case.

As the statement of the lemma holds in each scenario, the proof is concluded.  $\square$

We now prove that all correct processes quasi-decide  $v^*(k)$  in a good iteration.

**Lemma 27.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, all correct processes quasi-decide  $v^*(k)$  in iteration  $k$ .*

*Proof.* By Lem. 26, there exists  $c \in \{1, 2\}$  such that (1)  $z^*(k) \in \text{committed}(k, c)$ , and (2)  $|\text{committed}(k, c)| \leq 2$ . Moreover, recall that Lem. 23 proves that all correct processes commit  $z^*(k)$  in iteration  $k$ .

Consider sub-iteration  $(k, c, 1)$ . Observe that correct processes propose to  $\text{SMBA}[k][c][1]$  at most two different digests (as  $|\text{committed}(k, c)| \leq 2$ ) out of which one is  $z^*(k)$  (as  $z^*(k) \in \text{committed}(k, c)$ ). Therefore, the strong validity property of  $\text{SMBA}[k][c][1]$  ensures that the decided digest was proposed by a correct process. Let us distinguish two cases:

- Let the digest decided from  $\text{SMBA}[k][c][1]$  be  $z^*(k)$ . As  $k$  is a good iteration, correctly-encoded RS symbols (that correspond to value  $v^*(k)$ ) are stored at  $\geq n - 2t = 2t + 1$  correct processes. Therefore, each correct process receives RS symbols with correct witnesses for  $z^*(k)$  via RECONSTRUCT messages from at least  $t + 1$  processes and decodes value  $v^*(k)$  (line 61). Hence, all correct processes propose  $v^*(k)$  to  $\text{MBA}[k][c][1]$ , which ensures that all correct processes decide  $v^*(k)$  (due to its strong unanimity property). As  $v^*(k)$  is valid, all correct processes indeed quasi-decide  $v^*(k)$  in iteration  $k$ .

– Let the digest decided from  $\mathcal{SMB}\mathcal{A}[k][c][1]$  be  $z \neq z^*(k)$ . Due to the strong validity property of  $\mathcal{SMB}\mathcal{A}[k][c][1]$  (recall that only two different digests are proposed to  $\mathcal{SMB}\mathcal{A}[k][c][1]$  by correct processes),  $z \in \text{committed}(k, c)$ . Therefore, the following holds:

- All correct processes that proposed  $z$  to  $\mathcal{SMB}\mathcal{A}[k][c][1]$  propose  $z^*(k)$  to  $\mathcal{SMB}\mathcal{A}[k][c][2]$  (line 69 or line 70). (Recall that all correct processes commit  $z^*(k)$  in iteration  $k$ .)
- All correct processes that proposed  $z^*(k)$  to  $\mathcal{SMB}\mathcal{A}[k][c][1]$  propose  $z^*(k)$  to  $\mathcal{SMB}\mathcal{A}[k][c][2]$  (as the check at line 67 does not pass).

Therefore, all correct processes propose  $z^*(k)$  to  $\mathcal{SMB}\mathcal{A}[k][c][2]$ . The strong validity and termination properties of  $\mathcal{SMB}\mathcal{A}[k][c][2]$  ensure that all correct processes decide  $z^*(k)$  from  $\mathcal{SMB}\mathcal{A}[k][c][2]$ . As in the previous case, correctly-encoded RS symbols (that correspond to value  $v^*(k)$ ) are stored at  $\geq n - 2t = 2t + 1$  correct processes. Therefore, each correct process receives RS symbols with correct witnesses for  $z^*(k)$  via RECONSTRUCT messages from at least  $t + 1$  processes and decodes value  $v^*(k)$  (line 61). This means that all correct processes propose  $v^*(k)$  to  $\mathcal{MBA}[k][c][2]$ , which ensures that all correct processes decide  $v^*(k)$  (due to its strong unanimity property). Given that  $v^*(k)$  is a valid value, all correct processes quasi-decide  $v^*(k)$ .

As the statement of the lemma holds in both cases, the proof is concluded.  $\square$

Finally, we are ready to prove Reducer’s termination.

**Theorem 18 (Termination).** *Reducer (Alg. 1) satisfies termination. Concretely, Reducer terminates in  $O(1)$  iterations in expectation and has  $O(1)$  expected time complexity.*

*Proof.* Lem. 27 proves that all correct processes quasi-decide in a good iteration  $k \in \mathbb{N}$ . Each iteration is good with (at least)  $P = \frac{2t+1}{4t+1} \approx \frac{1}{2}$  probability (due to the fact that  $|\mathcal{D}_{\text{first}}| \geq 2t + 1$  and  $n = 4t + 1$ ). Let  $E_k$  denote the event that Reducer does not terminate after  $k$ -th iteration. Therefore,  $\Pr[E_k] \leq (1-P)^k \leftarrow 0$  as  $k \leftarrow \infty$ . Moreover, let  $K$  be the random variable that denotes the number of iterations required for Reducer to terminate. Then,  $\mathbb{E}[K] = 1/P \approx 2$ , which proves that Reducer terminates in  $O(1)$  iterations in expectation. Given that (1) each iteration takes  $O(1)$  time in expectation, and (2) the dissemination phase takes  $O(1)$  time, Reducer has  $O(1)$  expected time complexity.  $\square$

*Quality.* To conclude the proof of Reducer’s correctness, we prove that Reducer satisfies the quality property.

**Theorem 19 (Quality).** *Reducer (Alg. 1) satisfies quality.*

*Proof.* Lem. 27 proves that all correct processes quasi-decide value  $v^*(k)$  in a good iteration  $k$ ; recall that  $v^*(k)$  is a non-adversarial value. Hence, if the first iteration of Reducer is good, all correct processes quasi-decide  $v^*(1)$  in iteration 1. In that case, for Reducer to decide  $v^*(1)$ , it is required that the `Index()` request selects  $v^*(1)$ . The probability of that happening is at least  $\frac{1}{4}$  as there can be at

most four quasi-decided values. Thus, the probability that Reducer decides a non-adversarial value is (at least)  $\frac{2t+1}{4t+1} \cdot \frac{1}{4} \approx \frac{1}{8}$ . Therefore, quality is ensured as the probability that an adversarial value is decided is at most  $\frac{7}{8} < 1$ .  $\square$

*Bit complexity.* First, we prove that correct processes send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.

**Lemma 28.** *Correct processes collectively send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.*

*Proof.* Each RS symbol is of size  $O(\frac{\ell}{n} + \log n)$  bits. Let  $p_i$  be any correct process. Process  $p_i$  sends  $n \cdot O(\frac{\ell}{n} + \log n + \lambda + \lambda \log n) = O(\ell + n\lambda \log n)$  bits via INIT messages. Moreover, process  $p_i$  sends  $O(n)$  bits via ACK, DONE and FINISH messages. Therefore, process  $p_i$  sends

$$\underbrace{O(\ell + n\lambda \log n)}_{\text{INIT}} + \underbrace{O(n)}_{\text{ACK, DONE \& FINISH}} \\ \subseteq O(\ell + n\lambda \log n) \text{ bits in the dissemination phase.}$$

This implies that correct processes collectively send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.  $\square$

Next, we prove that correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k)$  bits in every iteration  $k$ .

**Lemma 29.** *Correct processes collectively send  $O(n\ell + n^2\lambda \log n + n^2 \log k)$  bits in expectation in every iteration  $k \in \mathbb{N}$ .*

*Proof.* Correct processes send  $O(n^2\lambda + n^2 \log k)$  bits via STORED and SUGGEST messages. Recall that STORED messages include a single digest and SUGGEST messages include at most two digests. STORED and SUGGEST messages also include the number of the iteration to which they refer.

Now, consider any sub-iteration  $(k, x \in \{1, 2\}, y \in \{1, 2\})$ . Correct processes send  $O(n^2\lambda + n^2 \log k)$  bits in expectation while executing  $\mathcal{SMBA}[k][x][y]$ . Note that the  $O(n^2 \log k)$  term exists as (1) each message of  $\mathcal{SMBA}[k][x][y]$  needs to be tagged with “ $k, x \in \{1, 2\}, y \in \{1, 2\}$ ”, and (2)  $\mathcal{SMBA}[k][x][y]$  exchanges  $O(n^2)$  messages in expectation. Next, correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k)$  bits via RECONSTRUCT messages. Finally, correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k)$  bits in expectation while executing  $\mathcal{MBA}[k][x][y]$ . Again, the  $O(n^2 \log k)$  term exists as (1) each message of  $\mathcal{MBA}[k][x][y]$  needs to be tagged with “ $k, x \in \{1, 2\}, y \in \{1, 2\}$ ”, and (2)  $\mathcal{MBA}[k][x][y]$  exchanges  $O(n^2)$  messages in expectation. Therefore, correct processes collectively send

$$\underbrace{O(n^2\lambda + n^2 \log k)}_{\mathcal{SMBA}[k][x][y]} + \underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k)}_{\text{RECONSTRUCT}} + \underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k)}_{\mathcal{MBA}[k][x][y]} \\ \subseteq O(n\ell + n^2\lambda \log n + n^2 \log k) \text{ bits in sub-iteration } (k, x, y).$$

Given that iteration  $k$  has four sub-iterations, correct processes collectively send

$$\begin{aligned} & \underbrace{O(n^2\lambda + n^2 \log k)}_{\text{STORED \& SUGGEST}} + \underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k)}_{\text{sub-iterations}} \\ & \subseteq O(n\ell + n^2\lambda \log n + n^2 \log k) \text{ bits in iteration } k. \end{aligned}$$

Thus, the lemma holds.  $\square$

Finally, we prove that Reducer has  $O(n\ell + n^2\lambda \log n)$  expected bit complexity.

**Theorem 20 (Bit complexity).** *Reducer (Alg. 1) achieves  $O(n\ell + n^2\lambda \log n)$  expected bit complexity.*

*Proof.* Let  $B$  be the random variable that denotes the number of bits sent by correct processes in Reducer. Moreover, let  $K$  be the random variable that denotes the number of iterations it takes Reducer to terminate. By Thm. 18,  $\mathbb{E}[K] \in O(1)$ . Lem. 28 proves that all correct processes send  $O(n\ell + n^2\lambda \log n)$  during the dissemination phase. Similarly, Lem. 29 proves that, in each iteration  $k \in \mathbb{N}$ , correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k) \subseteq O(n\ell + n^2\lambda \log n + n^2 k)$  bits. Therefore, the following holds:

$$\begin{aligned} B & \in O\left(n\ell + n^2\lambda \log n + \sum_{k=1}^K (n\ell + n^2\lambda \log n + n^2 k)\right) \\ & \subseteq O\left(n\ell + n^2\lambda \log n + K \cdot (n\ell + n^2\lambda \log n) + \sum_{k=1}^K (n^2 k)\right) \\ & \subseteq O\left(K \cdot (n\ell + n^2\lambda \log n) + n^2 \cdot \frac{1}{2} \cdot K(K+1)\right) \\ & \subseteq O\left(K \cdot O(n\ell + n^2\lambda \log n) + n^2 \cdot (K^2 + K)\right). \end{aligned}$$

Hence, we compute  $\mathbb{E}[B]$  in the following way, using  $\mathbb{E}[K] \in O(1)$ :

$$\begin{aligned} \mathbb{E}[B] & \in O(n\ell + n^2\lambda \log n) \cdot \mathbb{E}[K] + O(n^2) \cdot \mathbb{E}[K^2] + O(n^2) \cdot \mathbb{E}[K] \\ & \subseteq O(n\ell + n^2\lambda \log n) + O(n^2) + O(n^2) \cdot \mathbb{E}[K^2] \\ & \subseteq O(n\ell + n^2\lambda \log n) + O(n^2) \cdot \mathbb{E}[K^2]. \end{aligned}$$

As  $K$  is a geometric random variable,  $\mathbb{E}[K^2] = \frac{2-P}{P^2} \in O(1)$ , where  $P \approx \frac{1}{2}$  is the probability that an iteration is good. Thus,  $\mathbb{E}[B] \in O(n\ell + n^2\lambda \log n)$ .  $\square$

## D Reducer++: Proof

This section formally proves the correctness and complexity of Reducer++. Recall that  $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$  is a constant defined at line 8 of Alg. 2.

*External validity.* First, we prove that Reducer++ satisfies external validity.

**Theorem 21 (External validity).** *Reducer++ (Alg. 2) satisfies external validity.*

*Proof.* The property is satisfied as every value quasi-decided by a correct process is valid as ensured by the check at line 56.  $\square$

*Agreement.* We say that a correct process  $p_i$  quasi-decides a vector  $vec$  in an iteration  $k \in \mathbb{N}$  if and only if  $quasi\_decisions_i = vec$  when process  $p_i$  reaches line 58. The following lemma proves that no two correct processes quasi-decide different vectors in any iteration  $k$ .

**Lemma 30.** *Let  $k \in \mathbb{N}$  be any iteration. Suppose a correct process  $p_i$  quasi-decides a vector  $vec_i$  in iteration  $k$  and another correct process  $p_j$  quasi-decides a vector  $vec_j$  in iteration  $k$ . Then,  $vec_i = vec_j$ .*

*Proof.* The lemma follows from the agreement property of the  $MBA[k][x]$  instance (of the MBA primitive), for every  $x \in [1, C]$ .  $\square$

Next, we prove the agreement property of Reducer++.

**Theorem 22 (Agreement).** *Reducer++ (Alg. 2) satisfies agreement.*

*Proof.* By contradiction, suppose (1) there exists a correct process  $p_i$  that decides a value  $v_i$ , and (2) there exists a correct process  $p_j$  that decides a value  $v_j \neq v_i$ . Let  $p_i$  (resp.,  $p_j$ ) decide  $v_i$  (resp.,  $v_j$ ) in some iteration  $k_i \in \mathbb{N}$  (resp.,  $k_j \in \mathbb{N}$ ). Therefore, process  $p_i$  (resp.,  $p_j$ ) quasi-decides  $v_i$  (resp.,  $v_j$ ) in iteration  $k_i$  (resp.,  $k_j$ ). Without loss of generality, let  $k_i \leq k_j$ .

As process  $p_i$  quasi-decides  $v_i$  in iteration  $k_i$ , process  $p_i$  quasi-decides a vector  $vec_i$  in iteration  $k_i$ ; note that  $v_i$  belongs to  $vec_i$ . By Lem. 30, process  $p_j$  also quasi-decides the non-empty vector  $vec_i$  in iteration  $k_i$ . We separate two cases:

- Let  $k_i = k_j$ . Due to the fact that the `Index()` request invoked in iteration  $k_i = k_j$  returns the same integer to all correct processes, we have that  $v_j = v_i$ . Thus, we reach a contradiction with  $v_j \neq v_i$  in this case.
- Let  $k_i < k_j$ . As  $p_j$  quasi-decides the non-empty vector  $vec_i$  in iteration  $k_i$ ,  $p_j$  decides in iteration  $k_i$  (if it has not done so in an earlier iteration). Therefore, we reach a contradiction with the fact that  $p_j$  decides in iteration  $k_j > k_i$ .

As neither of the above cases can occur, the proof is concluded.  $\square$

*Integrity.* First, we show that if any correct process proposes a value  $v$  to the  $MBA[k][x]$  instance, for any sub-iteration  $(k, x)$ , and all processes are correct, then  $v$  is the proposal of a correct process to Reducer++.

**Lemma 31.** *Let  $(k \in \mathbb{N}, x \in [1, C])$  be any sub-iteration and let all processes be correct. If any correct process  $p_i$  proposes a value  $v$  to  $MBA[k][x]$ , then  $v$  is the proposal of a correct process to Reducer++.*

*Proof.* Recall that `leader(k)` denotes the leader of iteration  $k$ . We now separate two cases:

- Let  $p_i$  execute line 52. In this case, process  $p_i$  has received (at least)  $et + 1$  RS symbols. Given that all processes are correct, all these RS symbols are sent by  $\text{leader}(k)$  during the dissemination phase and they all correspond to  $\text{leader}(k)$ 's proposal to Reducer++. Therefore,  $v$  is the proposal of  $\text{leader}(k)$  to Reducer++, which proves the statement of the lemma in this case.
- Let  $p_i$  execute line 54. The statement of the lemma trivially holds in this case as  $v$  is  $p_i$ 's proposal to Reducer++.

As the statement of the lemma holds in both cases, the proof is concluded.  $\square$

We are now ready to prove Reducer++'s integrity.

**Theorem 23 (Integrity).** *Reducer++ (Alg. 2) satisfies integrity.*

*Proof.* Suppose all processes are correct. Moreover, let a correct process  $p_i$  decide some value  $v$ ; note that value  $v$  must be valid by Thm. 21. Hence, process  $p_i$  quasi-decides  $v$  in some iteration  $k \in \mathbb{N}$ , which further implies that  $v$  is decided from  $\mathcal{MBA}[k][x]$  in some sub-iteration  $(k, x \in [1, C])$ . Given that  $v$  is valid and  $\perp_{\text{MBA}}$  is invalid,  $v \neq \perp_{\text{MBA}}$ . Thus, the non-intrusion property of  $\mathcal{MBA}[k][x]$  guarantees that  $v$  was proposed to  $\mathcal{MBA}[k][x]$  by a correct process. Lem. 31 then shows that  $v$  is the proposal of a correct process to Reducer++, which concludes the proof.  $\square$

*Termination.* We proceed to prove that Reducer++ satisfies termination. As in the proof of Reducer's correctness, we say that a correct process  $p_i$  *completes the dissemination phase* if and only if  $p_i$  executes line 20. Recall that the dissemination phase of Reducer++ is identical to the dissemination phase of Reducer. For completeness, the following lemma proves that all correct processes eventually complete the dissemination phase of Reducer++.

**Lemma 32.** *Every correct process eventually completes the dissemination phase.*

*Proof.* Given that the dissemination phase of Reducer++ is identical to the dissemination phase of Reducer, the lemma follows directly from Lem. 13.  $\square$

Next, we prove that if all correct processes start any iteration  $k$ , all correct processes eventually start sub-iteration  $(k, 1)$ .

**Lemma 33.** *Let  $k \in \mathbb{N}$  be any iteration such that all correct processes start iteration  $k$ . Then, all correct processes eventually start sub-iteration  $(k, 1)$ .*

*Proof.* As all correct processes start iteration  $k$  and there are at least  $n - t = (2 + \epsilon)t + 1$  correct processes, all correct processes eventually receive  $n - t = (2 + \epsilon)t + 1$  STORED messages (line 25) and broadcast a SUGGEST message (line 28). Hence, all correct processes eventually receive  $n - t = (2 + \epsilon)t + 1$  SUGGEST messages (line 29) and start sub-iteration  $(k, 1)$  at line 34.  $\square$

We now prove that if all correct processes start any sub-iteration  $(k, x)$ , then all correct processes eventually complete sub-iteration  $(k, x)$ .



**Lemma 34.** *Let  $(k \in \mathbb{N}, x \in [1, C])$  be any sub-iteration such that all correct processes start sub-iteration  $(k, x)$ . Then, all correct processes eventually complete sub-iteration  $(k, x)$ .*

*Proof.* Given that all correct processes start sub-iteration  $(k, x)$ , each correct process broadcasts a RECONSTRUCT message (line 48), thus ensuring that every correct process eventually receives  $n - t = (2 + \epsilon)t + 1$  RECONSTRUCT messages (line 49). Hence, all correct processes eventually propose to  $\mathcal{MBA}[k][x]$  (line 55). The termination property of  $\mathcal{MBA}[k][x]$  then ensures that all correct processes complete sub-iteration  $(k, x)$ .  $\square$

The following lemma proves that every sub-iteration is eventually started and completed by all correct processes.

**Lemma 35.** *Every sub-iteration is eventually started and completed by all correct processes.*

*Proof.* By Lem. 32, all correct processes start iteration 1. Therefore, Lem. 33 proves that all correct processes start sub-iteration  $(1, 1)$ . By inductively applying Lems. 33 and 34, we prove that every sub-iteration is eventually started and completed by all correct processes.  $\square$

To not pollute the presentation, we might not explicitly rely on Lem. 35 in the rest of the proof. As in the proof of Reducer's correctness, we say that a correct process  $p_i$  suggests a digest  $z$  in an iteration  $k \in \mathbb{N}$  if and only if  $p_i$  broadcasts a SUGGEST message with digest  $z$  in iteration  $k$  (line 28). Let  $\text{suggested}_i(k)$  denote the set of digests suggested by any correct process  $p_i$  in any iteration  $k \in \mathbb{N}$ . The following lemma proves that each correct process suggests at most  $\lceil \frac{3}{\epsilon} \rceil$  digests in every iteration.

**Lemma 36.** *For every correct process  $p_i$  and every iteration  $k$ , the following holds:  $|\text{suggested}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$ .*

*Proof.* For every digest  $z \in \text{suggested}_i(k)$ , process  $p_i$  receives (at least)  $n - 3t = \epsilon t + 1$  STORED messages in iteration  $k$  (line 26). As  $p_i$  receives  $n - t = (2 + \epsilon)t + 1$  STORED messages (line 25) before broadcasting its SUGGEST message, there can be at most  $\frac{(2+\epsilon)t+1}{\epsilon t+1}$  suggested digests. Knowing that  $t \geq 1$ , we can bound  $\frac{(2+\epsilon)t+1}{\epsilon t+1}$  in the following way:

$$\frac{(2 + \epsilon)t + 1}{\epsilon t + 1} < \frac{(2 + \epsilon)t + 1}{\epsilon t} = \frac{2t + \epsilon t + 1}{\epsilon t} \leq \frac{2t + \epsilon t + t}{\epsilon t} = \frac{3 + \epsilon}{\epsilon} = \frac{3}{\epsilon} + 1 \leq \lceil \frac{3}{\epsilon} \rceil + 1.$$

Therefore,  $|\text{suggested}_i(k)| \leq \frac{(2+\epsilon)t+1}{\epsilon t+1} < \lceil \frac{3}{\epsilon} \rceil + 1$ , which concludes the proof.  $\square$

We say that a correct process *commits* a digest  $z$  in an iteration  $k \in \mathbb{N}$  if and only if  $z \in \text{candidates}_i$  when process  $p_i$  reaches line 34 in iteration  $k$ . Let  $\text{committed}_i(k)$  denote the set of digests committed by correct process  $p_i$  in iteration  $k \in \mathbb{N}$ . We now prove that, for every correct process  $p_i$  and every iteration  $k \in \mathbb{N}$ ,  $\text{committed}_i(k) \subseteq \text{suggested}_i(k)$ .

**Lemma 37.** *For every correct process  $p_i$  and every iteration  $k \in \mathbb{N}$ , the following holds: (1)  $\text{committed}_i(k) \subseteq \text{suggested}_i(k)$ , and (2)  $|\text{committed}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$ .*

*Proof.* We have that  $\text{committed}_i(k) \subseteq \text{suggested}_i(k)$  directly from the fact that correct process  $p_i$  only removes digests from its  $\text{candidates}_i$  list after broadcasting its SUGGEST message in iteration  $k$  (line 32). Moreover, as  $|\text{suggested}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$  (by Lem. 36),  $|\text{committed}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$ .  $\square$

To prove Reducer++'s termination, we rely on the notion of good iterations. Recall that, by Def. 1, an iteration  $k \in \mathbb{N}$  is said to be good if and only if  $\text{leader}(k) \in \mathcal{D}_{\text{first}}$ . Moreover, recall that, for every good iteration  $k$ , (1)  $v^*(k)$  denotes the valid proposal of  $\text{leader}(k)$ , and (2)  $z^*(k)$  denotes the digest of  $v^*(k)$ . The following lemma proves that every correct process suggests  $z^*(k)$  in every good iteration  $k \in \mathbb{N}$ .

**Lemma 38.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, every correct process suggests  $z^*(k)$  in iteration  $k$ .*

*Proof.* As  $\text{leader}(k) \in \mathcal{D}_{\text{first}}$ ,  $\text{leader}(k)$  stores  $z^*(k)$  at  $(2 + \epsilon)t + 1$  processes in the dissemination phase, out of which at most  $t$  can be faulty. Thus,  $\text{leader}(k)$  stores  $z^*(k)$  at  $\geq (1 + \epsilon)t + 1$  correct processes. This implies that each correct process receives  $z^*(k)$  in STORED messages from at least  $\epsilon t + 1$  processes, which further means that each correct process broadcasts a SUGGEST message with  $z^*(k)$  and thus suggests  $z^*(k)$  in iteration  $k$ .  $\square$

Next, we show that every correct process commits  $z^*(k)$  in every good iteration  $k$ .

**Lemma 39.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, every correct process commits  $z^*(k)$  in iteration  $k$ .*

*Proof.* By Lem. 38, all correct processes suggest  $z^*(k)$  in iteration  $k$ . Therefore, each correct process receives a SUGGEST message with  $z^*(k)$  from at least  $(1 + \epsilon)t + 1$  processes, which means that each correct process commits  $z^*(k)$ .  $\square$

For every iteration  $k \in \mathbb{N}$ , we define the  $\text{committed}(k)$  set:

$$\text{committed}(k) = \{z \mid z \text{ is committed by a correct process in iteration } k\}.$$

The following lemma proves that  $|\text{committed}(k)| \leq C$  in every good iteration  $k$ . Recall that  $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$ .

**Lemma 40.** *For every good iteration  $k \in \mathbb{N}$ ,  $|\text{committed}(k)| \leq C$ .*

*Proof.* For every digest  $z \in \text{committed}(k)$ , at least  $(1 + \epsilon)t + 1 - t = \epsilon t + 1$  correct processes suggest  $z$  in iteration  $k$ . Moreover, by Lem. 36, each correct process suggests at most  $\lceil \frac{3}{\epsilon} \rceil$  digests. Given that there are at most  $n = (3 + \epsilon)t + 1$  correct processes, we bound the cardinality of the  $\text{committed}(k)$  set:

$$|\text{committed}(k)| \leq \frac{((3 + \epsilon)t + 1) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon t + 1}.$$

As  $t \geq 1$ , we can further simplify:

$$\frac{((3 + \epsilon)t + 1) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon t + 1} \leq \frac{((3 + \epsilon)t + t) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon t + 1} < \frac{((3 + \epsilon)t + t) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon t} = \frac{(4 + \epsilon) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon}.$$

Thus,  $|\text{committed}(k)| < \frac{(4 + \epsilon) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon}$ . Given that  $\lceil \frac{3}{\epsilon} \rceil \leq \frac{3}{\epsilon} + 1$ , we have:

$$\frac{(4 + \epsilon) \cdot \lceil \frac{3}{\epsilon} \rceil}{\epsilon} \leq \frac{(4 + \epsilon) \cdot (\frac{3}{\epsilon} + 1)}{\epsilon} = \frac{\frac{12}{\epsilon} + 4 + 3 + \epsilon}{\epsilon} = \frac{\frac{12}{\epsilon} + \epsilon + 7}{\epsilon} = \frac{12}{\epsilon^2} + \frac{7}{\epsilon} + 1.$$

Therefore, we have:

$$|\text{committed}(k)| < \frac{12}{\epsilon^2} + \frac{7}{\epsilon} + 1 \leq \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil + 1.$$

Thus,  $|\text{committed}(k)| \leq \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil = C$ .  $\square$

Next, we prove a crucial lemma about the  $\mathcal{MBA}[k][x]$  instance employed in a sub-iteration  $(k, x)$ .

**Lemma 41.** *Consider any sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$ . Suppose a correct process  $p_i$  decides some value  $v'$  from  $\mathcal{MBA}[k][x]$ . Moreover, suppose all correct processes that proposed to  $\mathcal{MBA}[k][x]$  before  $p_i$  decides  $v'$  do so with the same value  $v$ . Then,  $v' = v$  (except with negligible probability).*

*Proof.* By contradiction, suppose  $v \neq v'$  with non-negligible probability. Let us denote by  $\mathcal{E}$  this execution of  $\mathcal{MBA}[k][x]$  that ends with process  $p_i$  deciding  $v'$ . We can construct a continuation  $\mathcal{E}'$  of  $\mathcal{E}$  in which all correct processes propose the same value  $v$ . Therefore,  $\mathcal{MBA}[k][x]$  violates the strong unanimity property in  $\mathcal{E}'$  and the probability measure of execution  $\mathcal{E}'$  is non-negligible (given that the probability measure of  $\mathcal{E}$  is non-negligible). Thus, we reach a contradiction with the fact that  $\mathcal{MBA}[k][x]$  satisfies strong unanimity with all but non-negligible probability, which concludes the proof of the lemma.  $\square$

We say that a sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$  *starts* at the moment when the first correct process invokes the `Noise()` request in sub-iteration  $(k, x)$  (line 36). Similarly, we say that a sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$  *ends* at the moment when the first correct process decides from  $\mathcal{MBA}[k][x]$  in sub-iteration  $(k, x)$  (line 55). For any sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$ , we define the `start`( $k, x$ ) set:

$$\text{start}(k, x) = \{z \mid z \text{ is committed by a correct process before sub-iteration } (k, x) \text{ starts}\}.$$

Next, for every sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$ , we define the `end`( $k, x$ ) set:

$$\text{end}(k, x) = \{z \mid z \text{ is committed by a correct process before sub-iteration } (k, x) \text{ ends}\}.$$

Note that the following holds:

- For every iteration  $k \in \mathbb{N}$  and every  $x \in [1, C]$ ,  $\text{start}(k, x) \subseteq \text{committed}(k)$ .
- For every iteration  $k \in \mathbb{N}$  and every  $x \in [1, C]$ ,  $\text{end}(k, x) \subseteq \text{committed}(k)$ .

- For every iteration  $k \in \mathbb{N}$ ,  $\text{start}(k, 1) \subseteq \text{end}(k, 1) \subseteq \text{start}(k, 2) \subseteq \text{end}(k, 2) \subseteq \dots \subseteq \text{start}(k, C) \subseteq \text{end}(k, C) \subseteq \text{committed}(k)$ . This is true as (1) each sub-iteration  $(k, \cdot)$  starts before it ends, and (2) each sub-iteration  $(k, x)$  ends before sub-iteration  $(k, x + 1)$  starts.

The following lemma proves that  $z^*(k) \in \text{start}(k, x)$ , for any sub-iteration  $(k, x)$  of a good iteration  $k$ .

**Lemma 42.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, for every sub-iteration  $(k, x \in [1, C])$ ,  $z^*(k) \in \text{start}(k, x)$ .*

*Proof.* By Lem. 39, all correct processes commit  $z^*(k)$  in iteration  $k$ . Hence, the correct processes that “starts” sub-iteration  $(k, 1)$  (i.e., invokes the first `Noise()` request) commits  $z^*(k)$  in sub-iteration  $k$ . Thus,  $z^*(k) \in \text{start}(k, 1)$ . As  $\text{start}(k, 1) \subseteq \text{start}(k, x)$ , for every  $x \in [2, C]$ , the lemma holds.  $\square$

We say that a correct process  $p_i$  *adopts* a digest  $z$  in a sub-iteration  $(k \in \mathbb{N}, x \in [1, C])$  if and only if  $\text{adopted\_digest}_i = z$  when process  $p_i$  reaches line 48 in sub-iteration  $(k, x)$ . Next, we prove that there exists a constant probability that all correct processes that propose to  $\text{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends propose  $v^*(k)$  given that (1)  $k$  is a good iteration, and (2)  $\text{start}(k, x) = \text{end}(k, x)$ .

**Lemma 43.** *Let  $k \in \mathbb{N}$  be any good iteration. Let  $(k, x \in [1, C])$  be any sub-iteration such that  $\text{start}(k, x) = \text{end}(k, x)$ . Then, there is at least  $\frac{1}{C}$  probability that all correct processes that propose to  $\text{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends propose  $v^*(k)$ .*

*Proof.* We prove the lemma through the following steps.

Step 1: *If a correct process  $p_i$  adopts  $z^*(k)$  in sub-iteration  $(k, x)$  and proposes  $v$  to  $\text{MBA}[k][x]$ , then  $v = v^*(k)$ .*

As  $k$  is a good iteration, correctly-encoded RS symbols (that correspond to value  $v^*(k)$ ) are stored at (at least)  $n - t - t = (1 + \epsilon)t + 1$  correct processes. Therefore, process  $p_i$  receives RS symbols with correct witnesses for  $z^*(k)$  via RECONSTRUCT messages from at least  $\epsilon t + 1$  processes and decodes  $v^*(k)$  (line 52). This means that process  $p_i$  indeed proposes  $v^*(k)$  to  $\text{MBA}[k][x]$ .

Step 2: *There is at least  $\frac{1}{C}$  probability that all correct processes that propose to  $\text{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do adopt  $z^*(k)$  in sub-iteration  $(k, x)$ .* Recall that Lem. 39 proves that every correct process commits  $z^*(k)$  in iteration  $k$ . Hence, no correct process  $p_i$  updates its  $\text{adopted\_digest}_i$  variable at line 38. Therefore, process  $p_i$  updates its  $\text{adopted\_digest}_i$  variable at line 46.

Let  $\phi$  denote the output of the `Noise()` request in sub-iteration  $(k, x)$ . Moreover, for every  $z \in \text{start}(k, x)$ , let  $h(z) = \text{hash}(z, \phi)$ . By Lem. 42,  $z^*(k) \in \text{start}(k, x)$ . Let  $H = \{h(z) \mid z \in \text{start}(k, x)\}$ . To prove the statement, we show that the probability  $h(z^*(k))$  is the lexicographically smallest hash value among  $H$  is at least  $\frac{1}{C}$ . Indeed, if  $h(z^*(k))$  is the lexicographically smallest hash value among  $H$ , then all correct processes that propose to  $\text{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do adopt  $z^*(k)$  at line 46.

Since processes (including the faulty ones) make only polynomially many random oracle queries, the probability that the random oracle model is queried on some  $(z \in \text{start}(k, x), \phi)$  is negligible. Therefore, each hash value among  $H$  is drawn independently uniformly at random (except with negligible probability). This implies that each hash value  $h(z) \in H$  has an equal chance of being the smallest hash value among  $H$ ; that chance is  $\frac{1}{|\text{start}(k, x)|} \geq \frac{1}{C}$  as  $|\text{start}(k, x)| \leq |\text{committed}(k)| \leq C$  (by Lem. 40). Hence, there is (at least)  $\frac{1}{C}$  probability that  $h(z^*(k))$  is the smallest hash value among  $H$ , which proves the statement.

Epilogue: By the statement of the second step, there is at least  $\frac{1}{C}$  probability that all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends adopt digest  $z^*(k)$  in sub-iteration  $(k, x)$ . Therefore, by the statement of the first step, there is at least  $\frac{1}{C}$  probability that all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do propose value  $v^*(k)$ .  $\square$

Next, we prove that there exists a sub-iteration  $(k, x)$  within any good iteration  $k$  such that  $\text{start}(k, x) = \text{end}(k, x)$ .

**Lemma 44.** *Let  $k \in \mathbb{N}$  be any good iteration. Then, there exists a sub-iteration  $(k, x \in [1, C])$  such that  $\text{start}(k, x) = \text{end}(k, x)$ .*

*Proof.* Recall that  $|\text{committed}(k)| \leq C$  (by Lem. 40). Moreover, Lem. 42 proves that  $z^*(k) \in \text{start}(k, x)$ , for every sub-iteration  $(k, x \in [1, C])$ . Lastly, note that the sub-iteration  $(k, x)$  ends before sub-iteration  $(k, x + 1)$  starts.

By contradiction, suppose  $\text{start}(k, x) \neq \text{end}(k, x)$ , for every sub-iteration  $(k, x \in [1, C])$ . As  $\text{start}(k, x) \subseteq \text{end}(k, x)$ , for every sub-iteration  $(k, x \in [1, C])$ , we have that  $\text{start}(k, x) \subset \text{end}(k, x)$ . Moreover,  $\text{start}(k, x) \subset \text{start}(k, x + 1)$ , for every  $x \in [1, C - 1]$ . Thus,  $|\text{start}(k, C)| \geq C$ , which then implies that  $|\text{end}(k, C)| > C$ . This contradicts  $\text{end}(k, C) \leq C$ , which completes the proof.  $\square$

The following lemma proves that if all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  (of a good iteration  $k$ ) ends do propose  $v^*(k)$ , then all correct processes quasi-decide  $v^*(k)$  in sub-iteration  $(k, x)$ .

**Lemma 45.** *Let  $k \in \mathbb{N}$  be any good iteration. Moreover, let  $(k \in \mathbb{N}, x \in [1, C])$  be any sub-iteration of iteration  $k$ . Suppose all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do propose  $v^*(k)$ . Then, all correct processes quasi-decide  $v^*(k)$  in sub-iteration  $(k, x)$ .*

*Proof.* By Lem. 41, the first correct process that decides from  $\mathcal{MBA}[k][x]$  does decide  $v^*(k)$ . As  $\mathcal{MBA}[k][x]$  ensures agreement and termination, all correct processes eventually decide  $v^*(k)$  from  $\mathcal{MBA}[k][x]$ , which proves the lemma.  $\square$

Finally, we are ready to prove the termination property of `Reducer++`.

**Theorem 24 (Termination).** *Reducer++ (Alg. 2) satisfies termination. Concretely, Reducer++ terminates in  $3C$  iterations in expectation and has  $O(C^2)$  expected time complexity.*

*Proof.* Lem. 45 proves that all correct processes quasi-decide the valid value  $v^*(k)$  in a sub-iteration  $(k, x)$  if (1)  $k$  is a good iteration, and (2) all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do propose  $v^*(k)$ . Given a good iteration  $k$  and its sub-iteration  $(k, x \in [1, C])$  with  $\text{start}(k, x) = \text{end}(k, x)$ , Lem. 43 proves that all correct processes that propose to  $\mathcal{MBA}[k][x]$  before sub-iteration  $(k, x)$  ends do propose  $v^*(k)$  with probability (at least)  $\frac{1}{C}$ . Moreover, Lem. 44 proves that there exists a sub-iteration  $(k, x)$  within any good iteration  $k$  such that  $\text{start}(k, x) = \text{end}(k, x)$ . Hence, the probability that correct processes terminate in any iteration  $k$  is (at least)  $P = \frac{|\mathcal{D}_{\text{first}}|}{n} \cdot \frac{1}{C} \geq \frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \cdot \frac{1}{C} \approx \frac{1}{3C}$ . Moreover, let  $E_k$  denote the event that Reducer++ has not terminated by the end of the  $k$ -th iteration. Due to independent randomness for each iteration,  $\Pr[E_k] \leq (1 - P)^k \rightarrow 0$  as  $k \rightarrow \infty$ . Let  $K$  be the random variable that denotes the number of iterations required for Reducer++ to terminate. Then,  $\mathbb{E}[K] = 1/P \approx 3C$ , which proves that Reducer++ terminates in  $3C$  iterations in expectation.

Finally, let  $T$  be the random variable that denotes the time required for Reducer++ to terminate. Moreover, let  $T(k)$  be the random variable that denotes the time required for  $k$ -th iteration to complete. We have that  $\mathbb{E}[T(k)] \in O(C)$ , for every iteration  $k$ , as every iteration has  $C$  sub-iterations, each of which takes  $O(1)$  time in expectation. We can express  $T$  in the following way:

$$T = \sum_{k=1}^K T(k).$$

Using the law of total expectation, we have:

$$\mathbb{E}[T] = \mathbb{E}\left[\sum_{k=1}^K T(k)\right] = \mathbb{E}\left[\mathbb{E}\left[\sum_{k=1}^K T(k) \mid K\right]\right].$$

Furthermore, we have:

$$\mathbb{E}\left[\sum_{k=1}^K T(k) \mid K\right] = \mathbb{E}[T(1)] + \dots + \mathbb{E}[T(K)] = K \cdot O(C).$$

Finally, we can compute  $\mathbb{E}[T]$ :

$$\mathbb{E}[T] = \mathbb{E}[K \cdot O(C)] = O(C) \cdot \mathbb{E}[K] \in O(C^2).$$

Thus, Reducer++ terminates in expected  $O(C^2)$  time.  $\square$

*Quality.* To conclude the proof of Reducer++'s correctness, we prove that Reducer++ satisfies the quality property.

**Theorem 25 (Quality).** *Reducer++ (Alg. 2) satisfies quality.*

*Proof.* Lems. 43 to 45 prove that all correct processes quasi-decide value  $v^*(k)$  in a good iteration  $k$  with (at least)  $\frac{1}{C}$  probability; recall that  $v^*(k)$  is a non-adversarial value. Hence, if the first iteration of Reducer++ is good, all correct processes quasi-decide  $v^*(1)$  in iteration 1 with probability  $\frac{1}{C}$ . In that case, for Reducer++ to decide  $v^*(1)$ , it is required that the Index() request selects  $v^*(1)$ . The probability of that happening is at least  $\frac{1}{C}$  as there can be at most  $C$  quasi-decided values. Thus, the probability that Reducer++ decides a non-adversarial value is (at least)  $\frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \cdot \frac{1}{C} \cdot \frac{1}{C} \approx \frac{1}{3C^2}$ . Therefore, quality is ensured as the probability that an adversarial value is decided is at most  $1 - \frac{1}{3C^2} < 1$ .  $\square$

*Bit complexity.* We start by proving that correct processes send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.

**Lemma 46.** *Correct processes collectively send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.*

*Proof.* Recall that each RS symbol is of size  $O(\frac{\ell}{n} + \log n)$  bits. Let  $p_i$  be any correct process. Process  $p_i$  sends  $n \cdot O(\frac{\ell}{n} + \log n + \lambda + \lambda \log n) = O(\ell + n\lambda \log n)$  bits via INIT messages. Moreover, process  $p_i$  sends  $O(n)$  bits via ACK, DONE and FINISH messages. Therefore, process  $p_i$  sends

$$\underbrace{O(\ell + n\lambda \log n)}_{\text{INIT}} + \underbrace{O(n)}_{\text{ACK, DONE \& FINISH}} \\ \subseteq O(\ell + n\lambda \log n) \text{ bits in the dissemination phase.}$$

This implies that correct processes collectively send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase.  $\square$

Next, we prove that correct processes send  $O(C(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C))$  bits in every iteration  $k$ .

**Lemma 47.** *Correct processes collectively send  $O(C(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C))$  bits in every iteration  $k \in \mathbb{N}$ .*

*Proof.* Correct processes send  $O(n^2\lambda \cdot \lceil \frac{3}{\epsilon} \rceil + n^2 \log k)$  bits via STORED and SUGGEST messages. Recall that STORED messages include a single digest and SUGGEST messages include at most  $\lceil \frac{3}{\epsilon} \rceil$  digests due to Lem. 36. STORED and SUGGEST messages also include the number of the iteration to which they refer.

Now, consider any sub-iteration  $(k, x \in [1, C])$ . Correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)$  bits via RECONSTRUCT messages. Moreover, correct processes send  $O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)$  bits in expectation while executing  $\mathcal{MBA}[k][x]$ . The  $O(n^2 \log k + n^2 \log C)$  term exists as (1) each message of  $\mathcal{MBA}[k][x]$  needs to be tagged with “ $k, x \in [1, C]$ ”, and (2)  $\mathcal{MBA}[k][x]$  exchanges  $O(n^2)$  messages in expectation. Thus, correct processes send

$$\underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)}_{\text{RECONSTRUCT}} + \underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)}_{\mathcal{MBA}[k][x]} \\ \subseteq O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C) \text{ bits in sub-iteration } (k, x).$$

Given that iteration  $k$  has  $C$  sub-iterations, correct processes collectively send

$$\underbrace{O(n^2\lambda \cdot \lceil \frac{3}{\epsilon} \rceil + n^2 \log k)}_{\text{STORED \& SUGGEST}} + C \cdot \underbrace{O(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)}_{\text{sub-iterations}}$$

$$\subseteq O(C(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)) \text{ bits in iteration } k.$$

Thus, the lemma holds.  $\square$

Finally, we are ready to prove that `Reducer++` exchanges  $O(C^2(n\ell + n^2\lambda \log n + n^2C))$  bits in expectation.

**Theorem 26 (Bit complexity).** *Reducer++ (Alg. 2) achieves  $O(C^2(n\ell + n^2\lambda \log n + n^2C))$  expected bit complexity.*

*Proof.* Let  $K$  be the random variable that denotes the number of iterations `Reducer++` takes to terminate. By Thm. 24,  $\mathbb{E}[K] = 3C$ . Let  $B$  be the random variable that denotes the number of bits sent in `Reducer++`. By Lem. 46, correct processes send  $O(n\ell + n^2\lambda \log n)$  bits in the dissemination phase. Similarly, in each iteration  $k \in \mathbb{N}$ , correct processes send  $O(C(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)) \subseteq O(C(n\ell + n^2\lambda \log n + n^2k + n^2 \log C))$  bits (by Lem. 47). Thus, we have the following:

$$B \in O\left(n\ell + n^2\lambda \log n + \sum_{k=1}^K C(n\ell + n^2\lambda \log n + n^2k + n^2 \log C)\right)$$

$$\subseteq O\left(n\ell + n^2\lambda \log n + K \cdot C(n\ell + n^2\lambda \log n + n^2 \log C) + \sum_{k=1}^K Cn^2k\right)$$

$$\subseteq O\left(n\ell + n^2\lambda \log n + K \cdot C(n\ell + n^2\lambda \log n + n^2 \log C) + \frac{1}{2} \cdot Cn^2 \cdot K(K+1)\right)$$

$$\subseteq O\left(n\ell + n^2\lambda \log n + K \cdot C(n\ell + n^2\lambda \log n + n^2 \log C) + Cn^2(K^2 + K)\right).$$

Hence, we compute  $\mathbb{E}[B]$  in the following way, using that  $\mathbb{E}[K] = 3C$ :

$$\mathbb{E}[B] \in O(n\ell + n^2\lambda \log n) + O(C(n\ell + n^2\lambda \log n + n^2 \log C)) \cdot \mathbb{E}[K] + O(Cn^2) \cdot (\mathbb{E}[K^2] + \mathbb{E}[K])$$

$$\subseteq O(n\ell + n^2\lambda \log n) + O(C^2(n\ell + n^2\lambda \log n + n^2 \log C)) + O(Cn^2) \cdot (\mathbb{E}[K^2] + \mathbb{E}[K])$$

$$\subseteq O(C^2(n\ell + n^2\lambda \log n + n^2 \log C)) + O(Cn^2) \cdot (\mathbb{E}[K^2] + \mathbb{E}[K]).$$

Given that  $K$  is a geometric random variable,  $\mathbb{E}[K^2] = \frac{2-P}{P^2} \in O(C^2)$ , where  $P \approx \frac{1}{3C}$  is the probability that `Reducer++` terminates in any specific iteration. Thus, we have:

$$\mathbb{E}[B] \in O(C^2(n\ell + n^2\lambda \log n + n^2 \log C)) + O(Cn^2) \cdot (O(C^2) + O(C))$$

$$\subseteq O(C^2(n\ell + n^2\lambda \log n + n^2 \log C)) + O(Cn^2) \cdot O(C^2)$$

$$\subseteq O(C^2(n\ell + n^2\lambda \log n + n^2 \log C)) + O(C^3n^2)$$

$$\subseteq O(C^2(n\ell + n^2\lambda \log n + n^2 \log C + n^2C))$$

$$\subseteq O(C^2(n\ell + n^2\lambda \log n + n^2C)).$$



Thus, the theorem holds.

□