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

Jovan Komatovic[1], Joachim Neu[2], and Tim Roughgarden[2,3]

[1]École Polytechnique Fédérale de Lausanne (EPFL), jovan.komatovic@epfl.ch
[2]a16z Crypto Research, {jneu,troughgarden}@a16z.com
[3]Columbia University, tim.roughgarden@gmail.com

Abstract

Multi-valued validated Byzantine agreement (MVBA), a fundamental primitive of distributed computing, allows n processes to agree on a valid ℓ-bit value, despite t faulty processes behaving maliciously. Among hash-based solutions for the asynchronous setting with adaptive faults, the state-of-the-art HMVBA protocol achieves optimal $O(n^2)$ message complexity, (near-)optimal $O(n\ell + n^2\lambda \log n)$ bit complexity, and optimal $O(1)$ time complexity. However, it only tolerates up to $t < \frac{1}{5}n$ adaptive failures. In contrast, the best known optimally resilient protocol, FIN-MVBA, exchanges $O(n^3)$ messages and $O(n^2\ell + n^3\lambda)$ bits. This highlights a fundamental question: can a hash-based protocol be designed for the asynchronous setting with adaptive faults that simultaneously achieves both optimal complexity and optimal resilience?

In this paper, we take a significant step toward answering the question. Namely, we introduce Reducer, an MVBA protocol that retains HMVBA's complexity while improving its resilience to $t < \frac{1}{4}n$. Like HMVBA and FIN-MVBA, Reducer relies exclusively on collision-resistant hash functions. A key innovation in Reducer's design is its internal use of strong multi-valued Byzantine agreement (SMBA), a variant of strong consensus we introduce and construct, which ensures agreement on a correct process's proposal. To further advance resilience toward the optimal one-third bound, we then propose Reducer++, an MVBA protocol that tolerates up to $t < (\frac{1}{3} - \epsilon)n$ adaptive failures, for any fixed constant $\epsilon > 0$. Unlike Reducer, Reducer++ does not rely on SMBA. Instead, it employs a novel approach involving hash functions modeled as random oracles to ensure termination. Reducer++ maintains constant time complexity, quadratic message complexity, and quasi-quadratic bit complexity, with constants dependent on ϵ.

1 Introduction

Multi-valued validated Byzantine agreement (MVBA), first introduced in [14], 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 ℓ-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 [31] 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 [29, 28, 51, 42] 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 unbiasable 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 [48]. By relying on threshold pseudorandom functions [15] or dedicated common-coin protocols [27, 33, 6], 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, overhead, and trust issues that come with trusted or private setups.

Finally, [14] has hinted and [29, 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, message, and bit complexity, and to maximize its resilience (the number t of faulty processes it can tolerate relative to the number n of all processes). The best known MVBA protocols in this context are HMVBA [29] and FIN-MVBA [28] (see Tab. 1). The HMVBA protocol relies solely on collision-resistant hash functions and achieves optimal $O(n^2)$ expected message complexity, (near-)optimal $O(n\ell + n^2\lambda \log n)$ expected bit complexity (where λ denotes the size of a hash value) and optimal $O(1)$ expected time complexity. A key drawback of HMVBA 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. However, this comes at a complexity cost. Specifically, FIN-MVBA exchanges $O(n^3)$ messages and $O(n^2\ell + n^3\lambda)$ bits.

Contributions. It is therefore natural to ask for a protocol that combines the strengths of HMVBA and FIN-MVBA. To this end, we first present Reducer, an adaptively-secure hash-based asynchronous MVBA protocol that matches the complexity of the HMVBA protocol while improving its resilience to $t < n/4$ (see Tab. 1). Like HMVBA and FIN-MVBA, Reducer relies solely on collision-resistant hash functions. A novel aspect of Reducer's design is its reliance on *strong multi-valued Byzantine agreement* (SMBA), a variant of strong consensus [32], which we introduce and construct and that ensures that correct processes reach agreement on the proposal of a correct process. In particular, during "good" iterations—where the common coin produces favorable outputs—Reducer utilizes the SMBA primitive as a crucial mechanism to guarantee the protocol's termination. To achieve Reducer's optimal complexity, an optimal SMBA algorithm was required. To this end, we introduce a cryptography-free SMBA algorithm (assuming a common coin) with optimal complexity. Both SMBA as a primitive and our SMBA algorithm may be of independent interest.

To approach the optimal one-third resilience found in FIN-MVBA, we then propose Reducer++, a hash-based MVBA protocol with further improved $t < (1/3 - \epsilon)n$ resilience, for any fixed constant $\epsilon > 0$ independent of n. Expected time, message, and bit complexity of Reducer++ remain constant, quadratic, and quasi-quadratic, respectively, with constants depending on ϵ. Reducer++, however,

Table 1: State-of-the-art adaptively-secure asynchronous MVBA algorithms.

Algorithm	Hash-based	Messages	Bits	Time	Resilience
CKPS01-MVBA [14] $^{\text{TS}}$	✘	$O(n^2)$	$O(n^2\ell + n^2\lambda + n^3)$	$O(1)$	$t < \frac{1}{3}n$
CKPS01-MVBA/HS [14] $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
VABA [4] $^{\text{TS}}$	✘	$O(n^2)$	$O(n^2\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
VABA/HS [4] $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-MVBA [42] $^{\text{TS}}$	✘	$O(n^2)$	$O(n\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
Dumbo-MVBA/HS [42] $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
sMVBA [34] $^{\text{TS}}$	✘	$O(n^2)$	$O(n^2\ell + n^2\lambda)$	$O(1)$	$t < \frac{1}{3}n$
sMVBA/HS [34] $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
FIN-MVBA [28] $^{\text{H-CR}}$	✔	$O(n^3)$	$O(n^2\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
FIN-MVBA [28] $^{\text{NO}}$	✔	$O(n^3)$	$O(n^2\ell + n^2\lambda + n^3 \log n)$	$O(1)$	$t < \frac{1}{3}n$
HMVBA [29] $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n\ell + n^2\lambda \log n)$	$O(1)$	$t < \frac{1}{5}n$
OciorMVBArr [20] $^{\text{NO}\ \dagger}$	✔	$O(n^2)$	$O(n\ell + n^2 \log n)$	$O(1)$	$t < \frac{1}{5}n$
OciorMVBA [20] $^{\text{NO}\ \dagger}$	✔	$O(n^2)$	$O(n\ell \log n + n^2 \log n)$	$O(\log n)$	$t < \frac{1}{3}n$
OciorMVBAh [20] $^{\text{H-CR}\ \dagger}$	✔	$O(n^3)$	$O(n\ell + n^3\lambda)$	$O(1)$	$t < \frac{1}{3}n$
FLT24-MVBA [30] $^{\text{H-CR}\ \dagger}$	✔	$O(n^2\kappa)$	$O(n\ell + n^2\lambda \log n + n^2\kappa\lambda)$	$O(\log \kappa)$	$t < \frac{1}{3}n$
This work: Reducer $^{\text{H-CR}}$	✔	$O(n^2)$	$O(n\ell + n^2\lambda \log n)$	$O(1)$	$t < \frac{1}{4}n$
This work: Reducer++ $^{\text{H-RO}}$	✔	$O(n^2)$	$O(n\ell + n^2\lambda \log n)$	$O(1)$	$t < (\frac{1}{3} - \epsilon)n$

Here, ℓ denotes the bit-length of the considered values, while λ represents the length of a hash value or a signature. Notations "H-CR", "H-RO", and "TS" indicate the use of collision-resistant hashes, hashes modeled as a random oracle, and threshold signatures, respectively, each of length λ; "NO" indicates use of no cryptography (assuming authenticated channels). Notation "/HS" refers to a hash-based variant in which the threshold signatures of the original protocol are substituted with lists of hash-based signatures from $n - t$ processes; here, we ignore the fact that some protocols crucially rely on an unforgeable unique threshold signature to generate a common coin. The κ parameter in the complexity of FLT24-MVBA is a statistical security parameter. Reducer++'s complexity exhibits a multiplicative constant factor of C^2, where $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$. As per convention in the asynchronous-agreement literature, the reported message, bit, and time complexities pertain to the protocol core, assuming the availability of common coins and consistently omitting their associated costs. Other asynchronous agreement protocols (including those with a weak common coin) are covered in §7. † Preprints announced shortly after preprint of this work.

necessitates an additional (but standard) assumption: the underlying hash function must be modeled as a random oracle. This assumption plays a crucial role in ensuring the protocol's termination.

Theoretical & practical perspective. No sub-cubic hash-based common-coin implementations are currently known. Therefore, from a theoretical perspective, our results (like [29, 30]) can be seen as a hash-only reduction from efficient MVBA to efficient common coin while preserving adaptive security. From a practical perspective, our protocols' randomness could be derived from natural shared entropy sources (e.g., proof-of-work blockchains). Importantly, our protocols can (be modified to) operate only with a weak common coin (which allows a constant probability of disagreement) so that expensive adversarial manipulation of the randomness source impacting the common-coin guarantees can only break liveness (and not safety!) of our protocols, and only for the duration of the manipulation, as discussed in §4. This approach may be sufficient for real-world applications.

Concurrent work. In a preprint announced shortly after the first version of this work, FLT24-MVBA [30] studies the same problem as we do, and achieves results that are incomparable and complementary to ours. On the positive side, FLT24-MVBA attains optimal resilience, tolerating up to one-third faulty processes, which improves upon the resilience of Reducer and Reducer++. However, our algorithms offer two significant advantages along two other dimensions: (1) Reducer and Reducer++ achieve better complexity, as FLT24-MVBA exchanges $O(n^2\kappa)$ messages, $O(n\ell + n^2\lambda \log n + n^2\lambda\kappa)$ bits, and terminates in $O(\log \kappa)$ time, where κ denotes a statistical security parameter. Note that κ is a security parameter [37, Sec. 3.1], not a constant (see also the

detailed discussion in §7.2): for constant κ, any FLT24-MVBA instance stalls indefinitely with constant probability. This would prohibit the composition of any more than a constant number of FLT24-MVBA instances and would thus severely limit its applicability, for instance in the standard and widespread "repeated MVBA" structure, which constructs atomic broadcast or state-machine replication from MVBA (e.g., in Tendermint [12]). (2) Our protocols satisfy the *quality* property [4], which ensures that, with constant probability, the decided value was proposed by a correct process. In contrast, FLT24-MVBA does not satisfy this property. This, too, for instance, renders FLT24-MVBA ill-suited for use in "repeated MVBA", since the absence of the quality property allows the adversary to indefinitely censor transactions from correct participants. We highlight that FLT24-MVBA, like other performant MVBA protocols (including ours), is sufficient to efficiently solve the asynchronous common subset (ACS) problem, although this requires the use of (hash-)signatures. A detailed discussion of FLT24-MVBA, its techniques, and how they relate to ours is in §7.2.

Roadmap. In §2, we provide a technical overview of our algorithms. We introduce the system model and formally define the MVBA problem in §3. We outline the building blocks of our algorithms in §4. In §5, we present the Reducer algorithm and provide an informal proof of its correctness and complexity. Then, §6 introduces Reducer++, along with a proof sketch of its correctness and complexity. Finally, §7 concludes the paper with a discussion of related work. The optional appendix contains all omitted algorithms and proofs.

2 Technical Overview

Given that both Reducer and Reducer++ are inspired by the HMVBA algorithm [29], which achieves an expected message complexity of $O(n^2)$, an expected bit complexity of $O(n\ell + n^2\lambda \log n)$ and terminates in $O(1)$ expected time (see Tab. 1), we start by revisiting HMVBA (§2.1). This review provides valuable insights into the design principles underlying our algorithms. We then proceed to outline the key mechanisms of Reducer (§2.2) and Reducer++ (§2.3). Finally, in §2.4, we discuss why Reducer's resilience is restricted to $t < \frac{1}{4}n$ and Reducer++'s to $t < (\frac{1}{3} - \epsilon)n$.

2.1 Revisiting HMVBA

At a high level, the HMVBA algorithm follows the "Disseminate-Elect-Agree" paradigm, which we describe below. Fig. 1 depicts the structure of HMVBA. For simplicity, let $n = 5t + 1$.

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 $[m_1, m_2, ..., m_n]$ of value v_i, where v_i is treated as a polynomial of degree t. Process p_i then utilizes Merkle-tree-based [44] cryptographic accumulators (see §4) in the following manner:

1. Process p_i computes the accumulation value (i.e., the Merkle root) z_i for the $[m_1, m_2, ..., m_n]$ set. We refer to this accumulation value z_i as the digest of p_i's proposal v_i.
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 $[m_1, m_2, ..., m_n]$ 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.[1]

[1] In the paper, when a process "broadcasts" a message, it simply unicasts the message to each process.

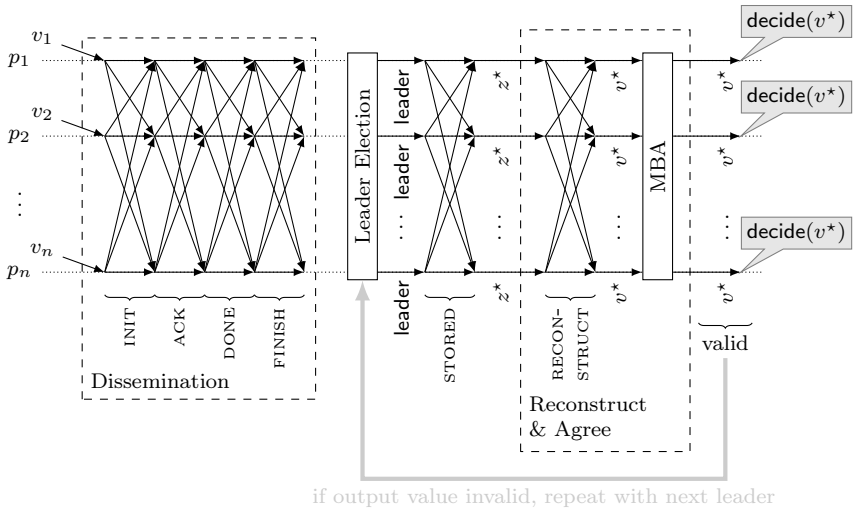

Figure 1: Depiction of HMVBA's structure. The depiction focuses on a good iteration k, where $\mathsf{leader}(k)$ has disseminated its valid proposal $v^\star(k)$ and the corresponding digest $z^\star(k)$. We abridge $\mathsf{leader} \triangleq \mathsf{leader}(k)$, $z^\star \triangleq z^\star(k)$, $v^\star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

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 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 those $3t + 1$ correct processes.

Second, if a correct process completes the dissemination phase, at least $(n - t) - t = 3t + 1$ processes have already successfully disseminated their valid proposals (as at most t processes can be corrupted). Thus, there are $3t + 1$ processes whose proposals can be reconstructed, even if the adversary corrupts (some of) them after dissemination. The HMVBA algorithm leverages this insight in the subsequent phases.

Election & agreement phases. After the dissemination phase is concluded, processes start executing HMVBA through *iterations*. Each iteration utilizes the multi-valued Byzantine agreement (MBA) primitive [13, 8, 46] 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 an idealized common coin, processes elect the leader of iteration k, denoted by $\mathsf{leader}(k)$. Then, the agreement phase starts. The goal of this phase is for processes to agree on the $\mathsf{leader}(k)$'s proposal. Specifically, once the leader is elected, each process p_i broadcasts via a STORED message the digest received from $\mathsf{leader}(k)$ 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 at least $(2t + 1) - t = t + 1$ 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.[2] 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 HMVBA and terminate. If not, processes continue to the next iteration.

Correctness analysis. We now briefly explain how HMVBA's design guarantees its correctness. Recall that if a correct process completes the dissemination phase, it is guaranteed that at least $(n-t)-t = 3t+1$ processes have already successfully disseminated their valid proposals. The HMVBA algorithm ensures that all correct processes decide (and terminate) in an iteration k whose leader is one of the aforementioned $3t+1$ processes. In the rest of the section, we refer to such iterations as *good*. Let us analyze how any such good iteration k of HMVBA unfolds.

The dissemination phase ensures that at least $(n-t)-t = 3t+1$ correct processes have stored (1) the digest $z^\star(k)$ of the $\mathsf{leader}(k)$'s valid proposal $v^\star(k)$, and (2) RS symbols of value $v^\star(k)$. 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 $z^\star(k)$ from at least $(n-t)+(n-2t)-n = 2t+1$ processes. Therefore, it is ensured that *all* correct processes adopt digest $z^\star(k)$. We emphasize that $t < \frac{1}{5}n$ is critical in this step. Namely, the HMVBA algorithm cannot guarantee that all correct processes adopt $z^\star(k)$ if $t \geq \frac{1}{5}n$. As we argue in §2.2, this "adoption issue" is the primary challenge one must overcome to achieve better resilience while preserving HMVBA's complexity.

After all correct processes have adopted digest $z^\star(k)$, making them agree on the $\mathsf{leader}(k)$'s valid value $v^\star(k)$ 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 $z^\star(k)$, all correct processes manage to (1) reconstruct $v^\star(k)$ after receiving $t+1$ such RS symbols via RECONSTRUCT messages, and (2) propose $v^\star(k)$ to the MBA primitive. Then, all correct processes decide $v^\star(k)$ from the MBA primitive due to its strong unanimity property. In other words, all correct processes output valid value $v^\star(k)$ from the R&A mechanism, which ensures that all correct processes decide $v^\star(k)$ from HMVBA in iteration k and terminate.

In summary, ensuring that all correct processes adopt the digest $z^\star(k)$ of the $\mathsf{leader}(k)$'s successfully disseminated valid proposal $v^\star(k)$ 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 HMVBA while maintaining its message, bit and time complexity, we propose Reducer that tolerates up to $t < \frac{1}{4}n$ failures. For simplicity, let $n = 4t+1$.

Key concepts behind Reducer. In Reducer (Fig. 2, Alg. 1), processes first disseminate their proposals using a dissemination phase identical to that of HMVBA. After the dissemination phase, processes start executing Reducer through iterations. Each Reducer's iteration k starts in the same way as HMVBA'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

[2]HMVBA [29] actually combines STORED and RECONSTRUCT messages: each STORED message contains both the digest and the RS symbol received from the leader. For clarity, we separate them as they serve distinct functions.

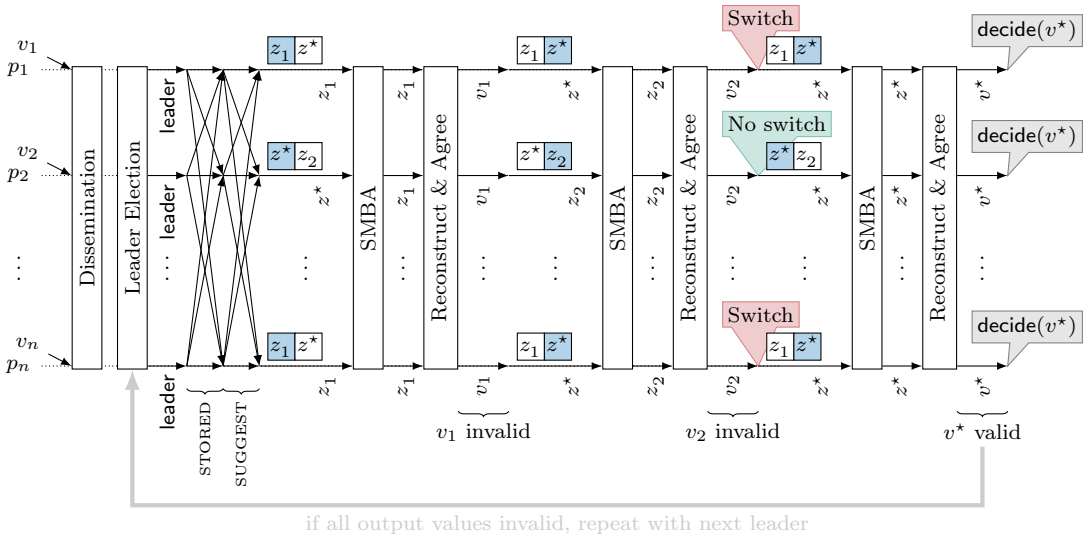

Figure 2: Depiction of Reducer's structure. The depiction focuses on a good iteration k in which the first two SMBA invocations decide adversarial digests z_1 and z_2, respectively. Finally, the third invocation decides the "good" digest $z^\star(k)$ of the leader(k)'s valid proposal $v^\star(k)$. See Fig. 1 for "Dissemination" and "Reconstruct & Agree" sub-protocols. We abridge leader $\triangleq$ leader(k), $z^\star \triangleq z^\star(k)$, $v^\star \triangleq v^\star(k)$.

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 HMVBA with $n = 4t+1$) cannot ensure that any correct process receives the "good" digest $z^\star(k)$ of the leader(k)'s valid proposal $v^\star(k)$ in a majority ($> \frac{3t+1}{2}$) of the received STORED messages. Instead, if the adversary corrupts the leader and makes it disseminate an additional adversarial digest, that adversarial digest might appear in a majority. Crucially, however, it is guaranteed that every correct process p_i receives $z^\star(k)$ in (at least) $t + 1$ received STORED messages. Indeed, as at least $(n - t) - t = 2t + 1$ correct processes have stored $z^\star(k)$, any set of $n - t = 3t + 1$ STORED messages must include at least $(n - t) + (n - 2t) - n = t + 1$ messages for $z^\star(k)$. Therefore, in Reducer, once a correct process p_i receives $n - t = 3t + 1$ STORED messages, p_i marks any digest received in at least $t + 1$ such messages as a *candidate* digest. Concretely, each correct process p_i has its local list $candidates_i$ that initially contains all digests received in $t + 1$ STORED messages. Importantly, given that k is a good iteration, $z^\star(k)$ belongs to the $candidates_i$ list at every correct process p_i. Moreover, since $\frac{3t+1}{t+1} < 3$, each correct process has at most two candidates. However, if an adaptive adversary corrupts leader(k) after its election and forces it to disseminate adversarial digests, the adversary can ensure the existence of linearly many different candidates *across all* correct processes:

$$|\{\text{candidate of } \textit{some} \text{ correct process}\}| \in O(n).$$

The (first) core idea of our Reducer algorithm is to *reduce* the number of different candidates across all correct processes in a good iteration to a constant.[3] 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

[3] A "reducing" technique similar to ours was previously employed for asynchronous Byzantine agreement by Mostéfaoui *et al.* [46]. However, their technique is insufficient for our algorithms, as detailed in §7.

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 $z^\star(k)$.

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 $z^\star(k)$ "survives" this communication step at p_i as (1) every correct process suggests $z^\star(k)$, 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. Given that each correct process suggests at most two digests out of which one is $z^\star(k)$, there are (at most) $n-t = 3t+1$ suggestions coming from correct processes for adversarial non-$z^\star(k)$ digests (assuming there are t faulty processes). Moreover, each adversarial non-$z^\star(k)$ 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-$z^\star(k)$ candidates get through the suggestion step. Consequently, a maximum of three candidates, including $z^\star(k)$, remain.

Establishing order in the chaos of candidates. At this point, each correct process has up to two candidate digests (one of which is $z^\star(k)$), and across all correct processes there are only up to three different candidate digests. As we will see below, we isolate and construct a special agreement primitive—strong multi-valued Byzantine agreement (SMBA), a variant of strong consensus [32], defined in §4—which ensures 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 and decide $z^\star(k)$.

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 $z^\star(k)$, p_i duplicates $z^\star(k)$, resulting in $candidates_i = [z^\star(k), z^\star(k)]$. 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. For any $c \in \{1, 2\}$, let us define the $\mathsf{committed}(k, c)$ set:

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

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 $\mathsf{committed}(k)$ set:

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

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

First, suppose $|\mathsf{committed}(k)| = 2$. Let $\mathsf{committed}(k) = \{z, z^\star(k)\}$, and assume that, without loss of generality, $z^\star(k)$ is lexicographically smaller than z. Clearly, each correct process p_i has its $candidates_i$ list as either $[z^\star(k), z^\star(k)]$ or $[z^\star(k), z]$. (Recall that each correct process necessarily commits $z^\star(k)$.) 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 $z^\star(k)$ during the first invocation. Agreement on $z^\star(k)$ would be sufficient for the processes to reconstruct $v^\star(k)$, agree on it, and thus terminate.

Now, suppose $|\mathsf{committed}(k)| = 3$, with $\mathsf{committed}(k) = \{z_1, z_2, z^\star(k)\}$. Assume that $z^\star(k)$ is lexicographically smaller than z_1 and z_2. Then, each correct process p_i has its $candidates_i$ list

as either $[z^\star(k), z^\star(k)]$, $[z^\star(k), z_1]$, or $[z^\star(k), z_2]$. (We again stress that every correct process is guaranteed to commit $z^\star(k)$.) The same procedure as in the case above leads to termination: during the first invocation of the SMBA primitive, correct processes agree on $z^\star(k)$, then agree on $v^\star(k)$, and terminate. The above argument naturally carries over to the case where $z^\star(k)$ is lexicographically greater than z_1 and z_2: the processes agree on $z^\star(k)$ during the second invocation of the SMBA primitive, ensuring termination.

Finally, consider the case where $\mathsf{committed}(k) = \{z_1, z_2, z^\star(k)\}$ and $z^\star(k)$ is lexicographically between z_1 and z_2. This is where the situation becomes more intricate. Each correct process p_i now has its $candidates_i$ list as either $[z^\star(k), z^\star(k)]$, $[z_1, z^\star(k)]$, or $[z^\star(k), z_2]$. 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 $z^\star(k)$ 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 $z^\star(k)$, and the decided digest from the second invocation is either $z^\star(k)$ or z_2. If either of these two invocations decides $z^\star(k)$, we are in a favorable position, following the same reasoning as previously discussed.

It is left to deal with the case where the digests decided by the two invocations are z_1 and z_2, respectively. Our approach is to "retry" the first invocation. Let us elaborate on this. After correct processes agree on z_1 and z_2 during the first and second invocation, respectively, they will initiate the third invocation in the following manner: All correct processes that proposed the digest z_1 decided in the first invocation now propose their other committed candidate, which must be $z^\star(k)$. All correct processes that proposed a digest other than z_1 to the first invocation stick with the same committed candidate for the third invocation—this digest must also be $z^\star(k)$. As a result, all correct processes propose $z^\star(k)$ in the third invocation (which represents a "repetition" of the first invocation). Therefore, agreement on $z^\star(k)$ is ensured in the third invocation, which, following the earlier arguments, implies Reducer's termination.

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

2.3 Overview of Reducer++

As already noted, Reducer++ improves Reducer's resilience to $t < (\frac{1}{3} - \epsilon)n$ Byzantine failures, for any fixed constant $\epsilon > 0$, while maintaining its complexity. Let $n = (3 + \epsilon)t + 1$ in this subsection.

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 HMVBA). 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 ϵt (and not t, like in Reducer and HMVBA). 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 least $(n - t) - t = (1 + \epsilon)t + 1$ SUGGEST messages. This ensures that each correct process commits the

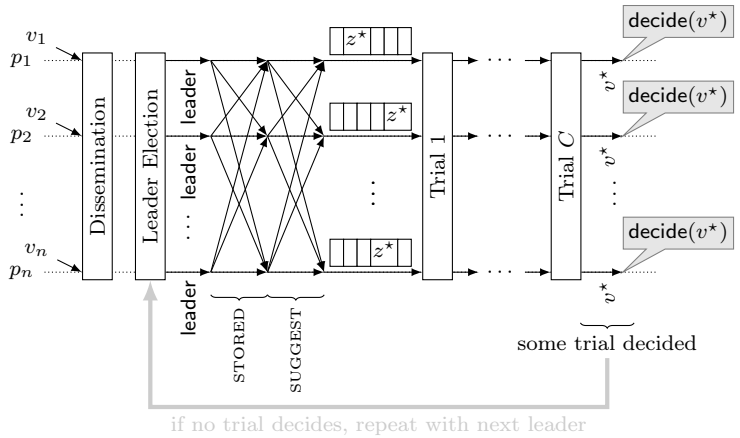

Figure 3: Depiction of Reducer++'s structure. The depiction focuses on a good iteration k where correct processes decide on the leader(k)'s valid proposal $v^\star(k)$ whose digest is $z^\star(k)$. See Figs. 1 and 4 for "Dissemination" and "Trial" sub-protocols, respectively. We abridge leader $\triangleq$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

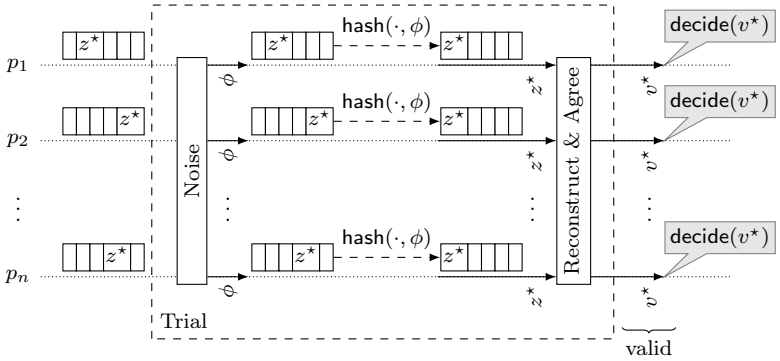

Figure 4: Depiction of Reducer++'s adoption procedure. The depiction focuses on a case where ϕ happens to be such that the "good" digest $z^\star(k)$ is smallest according to hash$(\cdot, \phi)$ and is thus adopted by all correct processes. See Fig. 1 for "Reconstruct & Agree" sub-protocol. We abridge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

"good" digest $z^\star(k)$ 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. Recall that, in contrast to Reducer++, Reducer deterministically terminates 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 $|\mathsf{committed}(k)| \leq C$, 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, $|\mathsf{committed}(k)| \leq \frac{((3+\epsilon)t+1)\lceil \frac{3}{\epsilon} \rceil}{\epsilon t+1} \leq C$. (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 $z^\star(k)$, correct processes decide the leader(k)'s valid proposal $v^\star(k)$ from Reducer++ by relying on the R&A mechanism of Reducer (and HMVBA) and terminate. Therefore, for Reducer++ to terminate with constant probability in iteration k, it is crucial to ensure that all correct processes adopt $z^\star(k)$ 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 λ-bit value ϕ. Intuitively, each correct process p_i (1) concatenates each committed digest z with the obtained random value ϕ, and (2) hashes the concatenation using the hash function $\mathsf{hash}(\cdot)$ modeled as a random oracle. Concretely, once ϕ is obtained, each correct process p_i constructs its local set H_i in the following way:

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

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

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

Let us analyze the probability that all correct processes adopt $z^\star(k)$. Given that only polynomially many (in λ) random oracle queries can be made and the common coin outputs a λ-bit random value, the procedure described above emulates a random permutation of the committed digests. Concretely, for every $z \in \mathsf{committed}(k)$, $\mathsf{hash}(z, \phi)$ is uniformly random with all but negligible probability. Hence, the probability that all correct processes adopt $z^\star(k)$ is equal to the probability that, given the obtained random value ϕ, $\mathsf{hash}(z^\star(k), \phi)$ is lexicographically smallest in the $\mathcal{H} = \{h \mid h = \mathsf{hash}(z, \phi) \wedge z \in \mathsf{committed}(k)\}$ set. As all members of $\mathcal{H}$ are uniformly random (except with negligible probability), this probability is $\frac{1}{|\mathcal{H}|} = \frac{1}{|\mathsf{committed}(k)|} \geq \frac{1}{C}$ given that $|\mathcal{H}| = |\mathsf{committed}(k)| \leq C$.[4]

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 $z^\star(k)$. To illustrate why this is the case, we now showcase a simple adversarial attack.

Note that, once the random value ϕ gets revealed, the adversary learns the value $\mathsf{hash}(z^\star(k), \phi)$ it needs to "beat". At this point, the adversary can find an adversarial digest z_A such that $\mathsf{hash}(z_A, \phi) < \mathsf{hash}(z^\star(k), \phi)$. Then, by corrupting $\mathsf{leader}(k)$ and making it disseminate z_A, the adversary can introduce 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 $\mathsf{hash}(z_A, \phi) < \mathsf{hash}(z^\star(k), \phi)$, these correct processes would 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 ϕ, 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 $|\mathsf{committed}(k)| \leq C$. Therefore, given that $z^\star(k)$ 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

[4]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.

correct processes adopt $z^\star(k)$, 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 $\geq \frac{n-2t}{n} = \frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \approx \frac{1}{3}$, 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 $O(C^2)$. As for the exchanged information, correct processes send $O(n^2)$ messages and $O(n\ell+n^2\lambda\log n)$ bits in the dissemination phase. Additionally, each iteration exchanges $O(Cn^2)$ messages and $O\big(C(n\ell + n^2\lambda\log n)\big)$ bits. As Reducer++ terminates in expected $O(C)$ iterations, Reducer++ yields an expected message complexity of $O(C^2n^2)$ and an expected bit complexity of $O\big(C^2(n\ell + n^2\lambda\log n)\big)$.

2.4 Discussion on Reducer's and Reducer++'s Resilience

This subsection discusses the resilience limits of our algorithms.

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., $|\mathsf{committed}(k)| \leq 3$). The two aforementioned constraints guarantee that correct processes will eventually reach an invocation of the SMBA primitive where they all propose and decide $z^\star(k)$. In brief, Reducer cannot improve upon the $t < \frac{1}{4}n$ resilience threshold because such a "$z^\star(k)$-agreeing" invocation of the SMBA primitive may never be reached 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 $n = 4t$, $\mathsf{leader}(k)$ has successfully disseminated the digest $z^\star(k)$ of its valid proposal $v^\star(k)$ to at least $(n - t) - t = 2t$ correct processes. Hence, each correct process is guaranteed to hear $z^\star(k)$ only from $(n - t) + (n - 2t) - n = n - 3t = t$ processes once it receives $n-t = 3t$ STORED messages in iteration k. Thus, to ensure that every correct process marks the "good" digest $z^\star(k)$ 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 $z^\star(k)$, 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-$z^\star(k)$) digests implies that the number of different candidates "surviving" the suggestion phase across all correct processes might be greater than three: $|\mathsf{committed}(k)| > 3$. Concretely, it can be shown that $|\mathsf{committed}(k)| \leq 7$ with $n = 4t$.

Observation 3: Reducer among $n = 4t$ would require an impossible variant of the SMBA primitive. Let us demonstrate a problematic scenario that may arise given $|\mathsf{committed}(k)| \leq 7$. Suppose $\mathsf{committed}(k) = \{z_1, z_2, z^\star(k), z_3, z_4\}$ with $z_1 < z_2 < z^\star(k) < z_3 < z_4$ 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 $[z_1, z_2, z^\star(k)]$.
- Correct processes in the $\mathcal{S}_2$ set commit $[z_2, z^\star(k), z_3]$.

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

Thus, for any $c \in \{1, 2, 3\}$, there exist $3 > 2$ different c-committed digests.

For Reducer to deal with the scenario above (when $n = 4t$), we would need to develop the SMBA primitive with strictly stronger properties than those required for $n = 4t + 1$. Specifically, for correct processes to learn about the proposals of other correct processes—the core principle underlying Reducer—the primitive must 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. (Without this guarantee, correct processes might keep deciding "useless" digests not held by any correct process.) Unfortunately, Fitzi and Garay [32] prove that, among $n = 4t$ processes, such a primitive cannot be implemented in asynchrony even for $d = 3$, indicating that Reducer's structure is not suitable for $n < 4t + 1$.

Reducer++ with optimal $n = 3t + 1$. We conclude the section by explaining why Reducer++ cannot achieve optimal resilience. One reason is that, when $t < \frac{1}{3}n$, Reducer++ cannot maintain its quasi-quadratic expected bit complexity. To ensure that the "good" digest $z^\star(k)$ 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 of size λ bits), thereby violating the desired quasi-quadratic upper bound. Incorporating the $\epsilon \cdot t$ gap into the resilience of Reducer++ allows each correct process to have only *constantly* many candidates, thus ensuring that the exchange of SUGGEST messages incurs $O(n^2\lambda)$ bits.

3 System Model & Problem Definition

System model. We consider n processes $p_1, p_2, ..., p_n$ connected through pairwise authenticated channels. This work considers a computationally bounded and *adaptive* adversary capable of corrupting up to $t > 0$ processes throughout (and not only at the beginning of) the protocol execution. For Reducer, we consider $t < \frac{1}{4}n$, while for Reducer++, we assume $t < (\frac{1}{3} - \epsilon)n$, where $\epsilon > 0$ is any fixed constant. 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. A process is said to be *correct* if it is never corrupted; a non-correct process is said to be *faulty*.

We focus on an *asynchronous* communication network where message delays are unbounded (but finite). Thus, we assume the adversary controls the network and can delay messages arbitrarily, but every message exchanged between correct processes must be delivered eventually. 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.

Multi-valued validated Byzantine agreement (MVBA). In this paper, we aim to design an MVBA [4, 42, 29, 16, 18, 41] protocol that operates in the model described above. Informally, MVBA requires correct processes to agree on a *valid* ℓ-bit value. The formal specification is given in Module 1.

4 Preliminaries

This section recapitulates the building blocks employed in our algorithms.

Module 1 MVBA

Associated values:
- set $\mathsf{Value}_{\mathsf{MVBA}}$ of ℓ-bit values

Associated functions:
- function $\mathsf{valid} : \mathsf{Value}_{\mathsf{MVBA}} \to \{true, false\}$; a value $v \in \mathsf{Value}_{\mathsf{MVBA}}$ is said to be valid if and only if $\mathsf{valid}(v) = true$

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

Assumed behavior:
- Every correct process proposes exactly once and it does so with a valid value.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *External validity:* No correct process decides an invalid value.
- *Weak validity:* If all processes are correct and a correct process decides a value $v \in \mathsf{Value}_{\mathsf{MVBA}}$, then v was proposed by a correct process.
- *Agreement:* No two correct processes decide different values.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.
- *Quality:* If a correct process decides a value $v \in \mathsf{Value}_{\mathsf{MVBA}}$, then the probability that v is a value determined by the adversary is at most $q < 1$.

Broadcasting operation. When a process *broadcasts* a message in our algorithms, the process simply unicasts the message to each process individually. As a result, this broadcasting operation is *unreliable*: if the sender is faulty, correct processes may receive different messages, or some may not receive any message at all.

Reed–Solomon codes. Our algorithms rely on Reed–Solomon (RS) codes [49]. Reducer and Reducer++ use RS codes as erasure codes; no (substitution-)error correction is required. We use $\mathsf{encode}(\cdot)$ and $\mathsf{decode}(\cdot)$ to denote RS' encoding and decoding algorithms. Namely, in a nutshell, $\mathsf{encode}(v)$ takes a value v, chunks it into the coefficients of a polynomial of degree t (for Reducer) or degree ϵt (for Reducer++), and outputs evaluations of the polynomial (i.e., the RS symbols) at n (the total number of processes) distinct locations. Similarly, $\mathsf{decode}(S)$ takes a set S of $t+1$ (for Reducer) or $\epsilon t + 1$ (for Reducer++) RS symbols (from distinct locations) and interpolates them into a polynomial of degree t (for Reducer) or degree ϵt (for Reducer++), whose coefficients are concatenated and output as decoded value v. The bit-size of an RS symbol obtained by the $\mathsf{encode}(v)$ algorithm is $O(\frac{|v|}{n} + \log n)$, where $|v|$ denotes the size of value v.

Hash functions. For Reducer, we assume a collision-resistant hash function $\mathsf{hash}(\cdot)$ 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 λ bits; we assume $\lambda > \log n$.[5]

Cryptographic accumulators. We adopt the definition of cryptographic accumulators from earlier works [10, 47]. 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 security parameter λ and a set $\mathcal{D}$ of n values $d_1, ..., d_n$, an accumulator has the following syntax:

- $\mathsf{Gen}(1^\lambda, n)$: Takes a parameter λ in unary representation 1^λ and an accumulation threshold n (an upper bound on the number of values that can be accumulated securely); returns a public

[5] Otherwise, $t \in O(n)$ faulty processes would have computational power exponential in λ.

accumulator key ak.

- $\mathsf{Eval}(ak, \mathcal{D})$: Takes an accumulator key ak and a set of values $\mathcal{D}$ to be accumulated; returns an accumulation value z for the set $\mathcal{D}$.
- $\mathsf{CreateWit}(ak, z, d_i, \mathcal{D})$: 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 $d_i \in \mathcal{D}$.
- $\mathsf{Verify}(ak, z, w_i, d_i)$: 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.

An accumulator scheme is secure if the adversary cannot produce a valid witness for a value d_i that was not in the set $\mathcal{D}$ used to produce the accumulation value z, i.e., for any accumulator key $ak \leftarrow \mathsf{Gen}(1^\lambda, n)$, it is computationally infeasible to obtain $(\{d_1, ..., d_n\}, d', w')$ such that (1) $d' \notin \{d_1, ..., d_n\}$, (2) $z \leftarrow \mathsf{Eval}(ak, \{d_1, ..., d_n\})$, and (3) $\mathsf{Verify}(ak, z, w', d') = \textit{true}$.

Concretely, we use Merkle trees [44] as our accumulators given they are 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 λ 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. For Merkle trees, accumulator security stems from collision resistance of the underlying hash function [44].

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. Since our algorithms require tracking the position of each RS symbol, the Merkle trees used in our algorithms effectively function as vector commitments [19].

Common coin. We follow the approach of prior works [9, 22, 45, 52, 28, 29] and assume the existence of an idealized common coin, an object introduced by Rabin [48], 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 process) 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 $\mathsf{Election}()$, and (2) obtaining a uniformly random integer in a specified constant range, denoted by $\mathsf{Index}()$. Reducer++ utilizes an additional common-coin object, denoted by $\mathsf{Noise}()$, that generates a uniformly random λ-bit value. All coins are independent. Our protocols can be easily adapted to rely only on a weak common coin, which allows a constant probability of disagreement among correct processes. This can be achieved by following the $\mathsf{Index}()$ common coin with an MBA instance (see §§ 5 and 6) to ensure agreement on the coin's random output.[6]

Multi-valued Byzantine agreement (MBA). Our algorithms internally utilize the well-known MBA primitive [13, 8, 46]. MBA is similar to the MVBA primitive: processes propose their values and decide on a common value. The formal specification of the MBA primitive is given in Module 2. In contrast to the MVBA primitive, MBA ensures justification (sometimes also called "non-intrusion"), but it does not guarantee external validity. Note that, whenever correct processes propose different values, MBA might decide futile $\perp_{\mathsf{MBA}}$. An MVBA algorithm must decide a valid (non-$\perp_{\mathsf{MBA}}$) value in this case, which represents a crucial difference between these two primitives. We treat the special value $\perp_{\mathsf{MBA}}$ as an invalid value. Specifically, the function $\mathsf{valid}(\cdot)$ (see Module 1) is defined for $\perp_{\mathsf{MBA}}$, with $\mathsf{valid}(\perp_{\mathsf{MBA}}) = \textit{false}$. We utilize this in our algorithms.

Our algorithms utilize MBA to agree on (1) ℓ-bit values (i.e., $\mathsf{Value}_{\mathsf{MBA}} \equiv \mathsf{Value}_{\mathsf{MVBA}}$), and (2) on $O(\lambda)$-bit digests (i.e., $\mathsf{Value}_{\mathsf{MBA}} \equiv \mathsf{Digest}$, where Digest denotes the set of all digests). For agreeing

[6] We underline that the SMBA and MBA algorithms utilized in our protocols require only a weak common coin.

Module 2 MBA

Associated values:
- set $\mathsf{Value}_{\mathsf{MBA}}$
- special value $\bot_{\mathsf{MBA}} \notin \mathsf{Value}_{\mathsf{MBA}}$; we assume $\mathsf{valid}(\bot_{\mathsf{MBA}}) = \mathit{false}$ ▷ we assume that $\bot_{\mathsf{MBA}}$ is an invalid value

Events:
- *input* $\mathsf{propose}(v \in \mathsf{Value}_{\mathsf{MBA}})$: a process proposes value v.
- *output* $\mathsf{decide}(v' \in \mathsf{Value}_{\mathsf{MBA}} \cup \{\bot_{\mathsf{MBA}}\})$: a process decides value v'.

Assumed behavior:
- Every correct process proposes exactly once.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *Strong unanimity:* If all correct processes propose the same value $v \in \mathsf{Value}_{\mathsf{MBA}}$ and a correct process decides a value $v' \in \mathsf{Value}_{\mathsf{MBA}} \cup \{\bot_{\mathsf{MBA}}\}$, then $v' = v$.
- *Agreement:* No two correct processes decide different values.
- *Justification:* If any correct process decides a value $v' \in \mathsf{Value}_{\mathsf{MBA}}$ ($v' \neq \bot_{\mathsf{MBA}}$), then v' is proposed by a correct process.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.

on ℓ-bit values, our algorithms rely on our MBA implementation MBA_ℓ (relegated to §A) with $O(n^2)$ expected message complexity, $O(n\ell + n^2\lambda \log n)$ 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 [46, 2] with $O(n^2)$ expected message complexity, $O(n^2\lambda)$ expected bit complexity and $O(1)$ expected time complexity. Both utilized MBA algorithms tolerate up to $t < \frac{1}{3}n$ failures.

Strong multi-valued Byzantine agreement (SMBA). Finally, the Reducer algorithm relies on *strong multi-valued Byzantine agreement* (SMBA), a variant of strong consensus [32], which we isolate as a separate primitive and construct, and which may be of independent interest. Concretely, Reducer utilizes SMBA to enable correct processes to agree on a digest. The formal specification of the SMBA primitive can be found in Module 3. We stress that the specification of the SMBA primitive assumes that correct processes propose only a constant number of different digests; this assumption is introduced solely for complexity reasons (see §B for more details). Crucially, our Reducer algorithm—specifically its reducing technique—enforces this condition: only $O(1)$ different digests are proposed by correct processes in any instance of the SMBA primitive utilized in Reducer. 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, 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 cryptography-free SMBA algorithm SMBA_λ (relegated to §B), which has an expected message complexity of (n^2), an expected bit complexity of $O(n^2\lambda)$ and an expected time complexity of $O(1)$; SMBA_λ tolerates up to $t < \frac{1}{4}n$ adaptive corruptions.

SMBA vs. strong consensus. Here, we point out subtle differences between the SMBA primitive (defined above) and the well-known strong consensus primitive [32]. Like SMBA, strong consensus requires agreement on the proposal of a correct process. Importantly, the strong consensus primitive provides strong validity, agreement, integrity, and termination only if correct processes propose values from a domain of possible inputs where (1) the domain is known to the protocol, and (2) the domain contains at most two values. Fitzi and Garay [32] prove that strong consensus (with two different values) can be solved in asynchrony if and only if $n > 3t$. In contrast, our SMBA primitive does not require knowledge of the domain of correct processes' proposals, and provides agreement,

Module 3 SMBA

Associated values:
- set Digest of $O(\lambda)$-bit digests ▷ SMBA is exclusively utilized for agreement on digests

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

Assumed behavior:
- Every correct process proposes exactly once.
- Only $O(1)$ different digests are proposed by correct processes.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *Strong validity:* If up to two different digests are proposed by correct processes and a correct process decides a digest $z' \in$ Digest, then z' is proposed by a correct process.
- *Agreement:* No two correct processes decide different values.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.

integrity, and termination irrespective of the size of the set comprising all correct processes' proposals, while it provides strong validity if it so happens that the set comprising all correct processes' proposals contains at most two values. In other words, SMBA "knows" that the *size* of the domain of correct inputs is at most two, but does not "know" the *values* in said domain, and only its validity is conditional on this knowledge, not its agreement, integrity, or termination.

Given that the SMBA problem is harder than the strong consensus problem, our SMBA_λ algorithm naturally solves the strong consensus problem. Moreover, we underline that our SMBA_λ algorithm can trivially be adapted to solve the strong consensus problem even with *optimal* $t < \frac{1}{3}n$ resilience (see §B for more details).[7] Additionally, SMBA_λ can be easily modified to allow us to solve the strong consensus problem with x different proposals, for any $x \in O(1)$, with optimal resilience of $n > (x+1)t$ [32], no employed cryptography, and optimal complexity.

5 Reducer: Pseudocode & Proof Sketch

This section presents our Reducer algorithm. Recall that Reducer exchanges $O(n^2)$ messages and $O(n\ell + n^2\lambda \log n)$ bits, and terminates in $O(1)$ time. Reducer tolerates up to $t < \frac{1}{4}n$ failures. We start by introducing the pseudocode of the Reducer algorithm (§5.1). Then, we provide an informal analysis of Reducer's correctness and complexity (§5.2). A formal proof can be found in §C.

5.1 Pseudocode

The pseudocode of Reducer is given in Alg. 1. Given that the correctness of our solution crucially depends on the exact number of "reduced" digests held by correct processes in an iteration, the presented solution assumes $n = 4t+1$. Note that it is trivial to adapt the solution for any $n > 4t+1$ in the following way: (1) First $4t+1$ processes (i.e., the $4t+1$ processes with smallest identifiers) execute the Reducer algorithm among $4t+1$ processes (as explained in the rest of the section). (2) The aforementioned $4t+1$ processes then utilize the cryptography-free asynchronous data dissemination (ADD) primitive [25] that efficiently disseminates the decided value to all $n > 4t+1$ processes.

Pseudocode description. Lines 1 to 18 define the employed primitives, the rules governing the behavior of correct processes, as well as the constants and local variables. When processes start

[7] Recall that the resilience of the SMBA_λ algorithm when solving the SMBA problem is $t < \frac{1}{4}n$.

Algorithm 1 Reducer: Pseudocode (for process p_i) [part 1 of 2]

```
1  Uses:
2      ▷ SMBA_λ exchanges O(n^2) messages and O(n^2 λ) bits and terminates in O(1) time
3      SMBA algorithm SMBA_λ, instances SMBA[k][x], ∀k ∈ ℕ, ∀x ∈ {1, 2, 3}
4      ▷ MBA_ℓ exchanges O(n^2) messages and O(nℓ + n^2 λ log n) bits and terminates in O(1) time
5      MBA algorithm MBA_ℓ with Value_MBA = Value_MVBA, instances MBA[k][x], ∀k ∈ ℕ, ∀x ∈ {1, 2, 3}
6  Rules:
7      - Any message with an invalid witness is ignored.
8      - Only one INIT message is processed per process.
9  Constants:
10     Digest default                                                    ▷ default digest
11 Local variables:
12     Value_MVBA v_i ← p_i's proposal
13     Boolean dissemination_completed_i ← false
14     Map(Process → [RS, Digest, Witness]) symbols_i ← empty map
15     List(Digest) candidates_i ← empty list          ▷ will be reset every iteration
16     Digest adopted_digest_i ← ⊥
17     Digest first_digest_i ← ⊥
18     List(Value_MVBA) quasi_decisions_i ← empty list
19 upon propose(Value_MVBA v_i)                                          ▷ start of the algorithm
20     ▷ dissemination phase starts
21     List(RS) [m_1, m_2, ..., m_n] ← encode(v_i)      ▷ encode(v_i) treats v_i as a polynomial of degree t
22     Digest z_i ← Eval([(1, m_1), (2, m_2), ..., (n, m_n)])           ▷ compute the digest
23     for each Process p_j:
24         Witness w_j ← CreateWit(z_i, (j, m_j), [(1, m_1), (2, m_2), ..., (n, m_n)])   ▷ compute the witness
25         send ⟨INIT, m_j, z_i, w_j⟩ to process p_j
26 upon receiving ⟨INIT, RS m_i, Digest z_j, Witness w_i⟩ from a process p_j:
27     if dissemination_completed_i = false:
28         symbols_i[p_j] ← [m_i, z_j, w_i]                  ▷ store the received RS symbol
29         send ⟨ACK⟩ to process p_j
30 upon receiving ⟨ACK⟩ from n − t processes (for the first time):
31     broadcast ⟨DONE⟩
32 upon receiving ⟨DONE⟩ from n − t processes (for the first time):
33     broadcast ⟨FINISH⟩ if p_i has not broadcast ⟨FINISH⟩ before
34 upon receiving ⟨FINISH⟩ from t + 1 processes (for the first time):
35     broadcast ⟨FINISH⟩ if p_i has not broadcast ⟨FINISH⟩ before
```

executing the Reducer algorithm, they first disseminate their proposals in the *dissemination phase* (lines 19-37). The dissemination phase of Reducer is identical to that of HMVBA 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 messages, 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*.

Algorithm 1 Reducer: Pseudocode (for process p_i) [part 2 of 2]

```
36 upon receiving ⟨FINISH⟩ from n − t processes (for the first time)
37     dissemination_completed_i ← true                                  ▷ dissemination phase completes
38     for each k = 1, 2, ...:                                           ▷ iteration k starts
39         candidates_i ← empty list                                     ▷ reset the list of candidates
40         Process leader(k) ← Election()                                ▷ elect a random leader
41         broadcast ⟨STORED, k, symbols_i[leader(k)].digest()⟩          ▷ disseminate the leader's digest
42         wait for n − t = 3t + 1 STORED messages for iteration k
43         for each Digest z included in n − 3t = t + 1 received STORED messages:
44             candidates_i.append(z)
45         broadcast ⟨SUGGEST, k, candidates_i⟩                          ▷ disseminate p_i's candidates
46         wait for n − t = 3t + 1 SUGGEST messages for iteration k:
47         for each Digest z ∈ candidates_i:
48             if z is not included in n − 2t = 2t + 1 received SUGGEST messages:
49                 candidates_i.remove(z)
50         if candidates_i.size = 0:                                     ▷ if no candidate "survives" the suggestion step
51             candidates_i[1] ← default; candidates_i[2] ← default      ▷ commit the default digest
52         else if candidates_i.size = 1:                                ▷ if exactly one candidate "survives" the suggestion step
53             candidates_i[2] ← candidates_i[1]                         ▷ copy the candidate
54         Sort candidates_i in the lexicographic order                  ▷ these digests are committed
55         for each x = 1, 2, 3:
56             if x ≠ 3:
57                 adopted_digest_i ← candidates_i[x]                    ▷ adopt the x-th committed digest if x ≠ 3
58             else if first_digest_i = candidates_i[1]: ▷ is the first committed digest decided from SMBA[k][1]?
59                 adopted_digest_i ← candidates_i[2]                    ▷ if yes, adopt the other committed digest
60             else:
61                 adopted_digest_i ← candidates_i[1]                    ▷ if not, stick with the first committed digest
62             Digest z ← SMBA[k][x].propose(adopted_digest_i)
63             if x = 1: first_digest_i ← z    ▷ store first decided digest as it is important for third sub-iteration
64             ▷ Reconstruct & Agree
65             broadcast ⟨RECONSTRUCT, k, x, symbols_i[leader(k)]⟩
66             wait for n − t = 3t + 1 RECONSTRUCT messages for sub-iteration (k, x)
67             Set(RS) S_i ← the set of all received RS symbols with valid witnesses for digest z
68             if |S_i| ≥ t + 1:                                         ▷ check if a value can be decoded using S_i
69                 Value_MVBA r_i ← decode(S_i)                          ▷ if yes, set r_i to the decoded value
70             else:
71                 Value_MVBA r_i ← v_i                                  ▷ if not, set r_i to p_i's proposal
72             Value_MVBA ∪ {⊥_MBA} v ← MBA[k][x].propose(r_i)           ▷ propose r_i
73             if valid(v) = true:
74                 quasi_decisions_i.append(v)                           ▷ quasi-decide v
75         if quasi_decisions_i.size > 0 and p_i has not previously decided:
76             Integer I ← Index()                                       ▷ obtain a random integer I in the [1, 3] range
77             trigger decide(quasi_decisions_i[(I mod quasi_decisions_i.size) + 1])   ▷ for quality
```

Each iteration $k \in \mathbb{N}$ unfolds as follows (lines 38-77):

1. Processes randomly elect the iteration's leader, denoted by $\mathsf{leader}(k)$ (line 40).
2. Processes establish their candidate digests through STORED and SUGGEST messages (lines 41-54), 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 55. Moreover, we define the $\mathsf{committed}(k, c)$ set, for any $c \in \{1, 2\}$, as

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

The default digest (line 10) is introduced solely to ensure that processes have always two committed candidates (see line 51), even if they do not commit any candidate through the standard STORED and SUGGEST steps.

3. Processes aim to agree on a valid value (lines 55-74). To achieve this, in every good iteration k—$\mathsf{leader}(k)$ has disseminated proposal $v^\star(k)$ with digest $z^\star(k)$—the following holds:

$$\big(z^\star(k) \in \mathsf{committed}(k,1) \wedge |\mathsf{committed}(k,1)| \leq 2\big) \vee \big(\{z^\star(k)\} = \mathsf{committed}(k,2)\big). \qquad (\circledcirc)$$

As noted in §2.2, if (only) the first disjunct holds true, it may require two repetitions for processes to agree on $z^\star(k)$ when correct processes propose two different digests from the $\mathsf{committed}(k,1)$ set to the SMBA primitive. In the first repetition, they may decide on a non-$z^\star(k)$ digest (the other digest from the $\mathsf{committed}(k,1)$ set), and only in the second repetition do they succeed in agreeing on $z^\star(k)$. For this reason, each iteration k is divided into three sub-iterations $(k,1)$, $(k,2)$, and $(k,3)$. In sub-iterations $(k,1)$ and $(k,2)$, processes aim to agree on their first and second committed digests, respectively. The sub-iteration $(k,3)$ serves as a repetition of the first sub-iteration.

Each sub-iteration (k,x) unfolds as follows (lines 56-74). Each process p_i begins by adopting a digest. If $x = 1$ or $x = 2$, p_i adopts its x-committed digest. Otherwise, p_i engages its "proposal-switching" logic: it checks whether the value it proposed to the first invocation of the SMBA primitive was decided. If so, p_i adopts its 2-committed digest; otherwise, p_i adopts its 1-committed digest. Process p_i then proposes its adopted digest (i.e., $adopted\_digest_i$) to the SMBA primitive. Once processes agree on a digest z via SMBA, processes start the R&A mechanism. Specifically, processes disseminate the RS symbols received from $\mathsf{leader}(k)$ 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.

4. The final phase (lines 75-77) of the iteration is designed to ensure the quality property. After completing all three sub-iterations, processes check if any value was quasi-decided. If so, processes obtain a random integer $I \in [1,3]$, which is then used to select one of the previously quasi-decided values for the decision.

5.2 Proof Sketch

This subsection provides a proof sketch of Reducer's correctness and complexity. Recall that a formal proof can be found in §C.

Correctness. We first give a proof sketch of the following theorem.

Theorem 1 (Reducer is correct)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) is a correct implementation of the MVBA primitive in the presence of a computationally bounded adversary.*

We now analyze each property separately.

Agreement. The agreement property is ensured by (1) the agreement property of the MBA primitive employed in each iteration (line 72), and (2) the fact that any Index() request (line 76) returns the same integer to all correct processes.

Weak validity. Suppose all processes are correct and a correct process p_i decides some value v in an iteration k. Due to the justification 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 69, v is the proposal of $\mathsf{leader}(k)$. Otherwise, v is p_j's proposal (line 71). In any case, v is proposed by a correct process.

External validity. The property is trivially satisfied due to the check at line 73.

Integrity. The property is trivially satisfied due to the check at line 75.

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_{first} denote the first correct process that broadcasts a FINISH message at line 33. Note that p_{first} broadcasts the FINISH message at line 33 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_{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* good *if and only if* $\mathsf{leader}(k) \in \mathcal{D}_{\text{first}}$.

We are now ready to show that Reducer terminates in the first good iteration k. Let $v^\star(k)$ denote the valid proposal of $\mathsf{leader}(k)$ and let $z^\star(k)$ denote the digest of $v^\star(k)$. As k is a good iteration, $\mathsf{leader}(k)$ has stored valid RS symbols of its proposal $v^\star(k)$ at $(n-t)-t = 2t+1$ correct processes. As discussed in §5.1 (and proven in §C), Reducer ensures that Eq. (◎) holds:

$$\big(z^\star(k) \in \mathsf{committed}(k,1) \land |\mathsf{committed}(k,1)| \leq 2\big) \lor \big(\{z^\star(k)\} = \mathsf{committed}(k,2)\big).$$

We now separate two cases:

- Let $\mathsf{committed}(k,2) = \{z^\star(k)\}$. Consider sub-iteration $(k,2)$. All correct processes decide $z^\star(k)$ from the SMBA primitive (line 62). Then, as $\mathsf{leader}(k)$ has disseminated RS symbols of its valid proposal $v^\star(k)$ to (at least) $n - 2t = 2t + 1$ correct processes, each correct process receives $(n - t) + (n - 2t) - n = n - 3t = t + 1$ RS symbols that correspond to digest $z^\star(k)$ (line 66). Hence, all correct processes decode $v^\star(k)$ (line 69) and propose $v^\star(k)$ to the MBA primitive (line 72). The strong unanimity property of the MBA primitive ensures that all correct processes decide $v^\star(k)$ from it and, thus, quasi-decide $v^\star(k)$ (line 74). Hence, termination is ensured.
- Let $z^\star(k) \in \mathsf{committed}(k,1)$ and $|\mathsf{committed}(k,1)| \leq 2$. Consider sub-iteration $(k,1)$. As $|\mathsf{committed}(k,1)| \leq 2$, correct processes propose at most two different digests to the SMBA primitive (line 62). The strong validity property guarantees that the decided digest z is proposed by a correct process. We further investigate two possibilities:
  - Let $z = z^\star(k)$. Following the same reasoning as in the previous scenario, all correct processes decode $v^\star(k)$ (line 69) and propose $v^\star(k)$ to the MBA primitive (line 72). The strong unanimity property of the MBA primitive ensures that all correct processes decide $v^\star(k)$ from it and, thus, quasi-decide $v^\star(k)$ (line 74). Hence, termination is guaranteed in this case.
  - Let $z \neq z^\star(k)$. In this case, the "proposal-switching" logic (lines 58-61) ensures that all correct processes propose $z^\star(k)$ to the SMBA primitive in sub-iteration $(k,3)$. (Recall that all correct processes commit $z^\star(k)$.) Therefore, $z^\star(k)$ is decided from the SMBA primitive (line 62) in sub-iteration $(k,3)$. This implies that all correct processes decode $v^\star(k)$ (line 69) and propose $v^\star(k)$ to the MBA primitive (line 72). Hence, all correct processes quasi-decide $v^\star(k)$ (line 74), showing that Reducer terminates even in this case.

Quality. Let P_1 denote the probability that the first iteration is good; note that $P_1 \geq \frac{n-2t}{n} = \frac{2t+1}{4t+1} \geq \frac{1}{2}$. 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 76) invoked in a good iteration k that quasi-decides a

non-adversarial value indeed selects a non-adversarial quasi-decided value; $P_2 \geq \frac{1}{3}$ as at most three 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 $1 - P_1 \cdot P_2 \leq \frac{5}{6} < 1$.

Complexity. Next, we provide a proof sketch of the following theorem.

Theorem 2 (Reducer's expected complexity)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, the following holds for Reducer (see Alg. 1) in the presence of a computationally bounded adversary:*

- *The expected message complexity is $O(n^2)$.*
- *The expected bit complexity is $O(n\ell + n^2\lambda \log n)$.*
- *The expected time complexity is $O(1)$.*

Let us informally prove the theorem.

Expected time complexity. As previously argued, Reducer terminates in a good iteration. Given that each iteration is good with a probability $P \geq \frac{1}{2}$, Reducer terminates in expected two iterations. As (1) each iteration takes $O(1)$ time to complete in expectation, and (2) the dissemination phase consists of $O(1)$ rounds of communication, the expected time complexity is $O(1)$.

Expected message complexity. Given that (1) each iteration exchanges $O(n^2)$ messages in expectation, and (2) Reducer terminates in $O(1)$ iterations in expectation, correct processes exchange $O(n^2)$ messages in expectation through the iterations of Reducer. Finally, correct processes exchange $O(n^2)$ messages in expectation throughout the entire Reducer protocol as the dissemination phase exchanges $O(n^2)$ messages as well.

Expected bit complexity. The dissemination phase exchanges $O(n\ell + n^2\lambda \log n)$ bits. Moreover, each iteration exchanges $O(n\ell + n^2\lambda \log n)$ bits in expectation. Given that there are $O(1)$ iterations in expectation, the expected bit complexity of Reducer is

$$\underbrace{O(n\ell + n^2\lambda \log n)}_{\text{dissemination phase}} + O(1) \cdot \underbrace{O(n\ell + n^2\lambda \log n)}_{\text{each iteration}} \subseteq O(n\ell + n^2\lambda \log n).$$

6 Reducer++: Pseudocode & Proof Sketch

This section introduces Reducer++, our MVBA algorithm that comes ϵ-close to optimal one-third resilience, for any fixed constant $\epsilon > 0$. Reducer++ exchanges $O(n^2C^2)$ messages and $O\big(C^2(n\ell + n^2\lambda \log n)\big)$ bits, and terminates in $O(C^2)$ time, where $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$. 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 pseudocode (§6.1). Then, we give an informal analysis of Reducer++'s correctness and complexity (§6.2). A formal proof is relegated to §D.

6.1 Pseudocode

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

Pseudocode description. Lines 1 to 17 define the employed primitives, the rules governing the behavior of correct processes, as well as the constants and local variables. When processes start executing the Reducer++ algorithm (line 18), they engage in the dissemination phase (line 19) that is identical to the dissemination phase of Reducer (and HMVBA) except that values are treated as

polynomials of degree ϵt. Once the dissemination phase is completed, processes execute Reducer++ through *iterations*. Each iteration $k \in \mathbb{N}$ proceeds as follows (lines 22-55):

1. Processes elect the leader—$\mathsf{leader}(k)$—using a common coin (line 24).
2. Processes determine their candidate digests through STORED and SUGGEST messages (lines 25-33), 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 $candidates_i$ 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., $|\mathsf{committed}(k)| \leq C$).
3. Processes aim to agree on a valid value through C sequential probabilistic trials (lines 34-52). Concretely, we divide iteration k into C sub-iterations $(k, 1)$, $(k, 2)$, ..., (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 ϕ by utilizing a common coin. Then, each correct process p_i adopts a digest by updating its $adopted\_digest_i$ variable:
   - If no digest is committed by p_i, process p_i adopts the fixed default digest.
   - Otherwise, process p_i adopts a digest as follows: (1) For each committed digest z, p_i appends $\mathsf{hash}(z, \phi)$ to the $hashed\_candidates_i$ list. (2) Process p_i sorts $hashed\_candidates_i$ in the lexicographic order. (3) Finally, process p_i adopts the committed digest z' that, along with ϕ, produced the smallest element of the sorted $hashed\_candidates_i$ 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 $\mathsf{leader}(k)$ 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.
4. The final phase (lines 53-55) 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 Proof Sketch

This subsection gives a proof sketch of Reducer++'s correctness and complexity. Recall that a formal proof is relegated to §D.

Correctness. We start by providing a proof sketch of the following theorem.

Theorem 3 (Reducer++ is correct)**.** *Given $n = (3 + \epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) is a correct implementation of the MVBA primitive in the presence of a computationally bounded adversary.*

Since the analysis of Reducer++'s agreement, weak validity, external validity, and integrity follows the arguments made in the analysis of Reducer (see §5.2), we primarily focus on Reducer++'s termination and quality.

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 $v^\star(k)$ denote the valid proposal of $\mathsf{leader}(k)$ and let $z^\star(k)$ denote the digest of $v^\star(k)$. As discussed in §2.3, the adversary can only inject $C - 1$ adversarial digests in iteration k (as

Algorithm 2 Reducer++: Pseudocode (for process p_i)

Uses:
▷ MBA_ℓ exchanges $O(n^2)$ messages and $O(n\ell + n^2\lambda \log n)$ bits and terminates in $O(1)$ time
MBA algorithm MBA_ℓ with $\mathsf{Value}_{\mathsf{MBA}} = \mathsf{Value}_{\mathsf{MVBA}}$, **instances** $\mathcal{MBA}[k][x]$, $\forall k, x \in \mathbb{N}$

Rules:
- Any message with an invalid witness is ignored.
- Only one INIT message is processed per process.

Constants:
Digest *default* ▷ default digest
Integer $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$ ▷ constant C is used by all processes

Local variables:
$\mathsf{Value}_{\mathsf{MVBA}}$ $v_i \leftarrow p_i$'s proposal
Boolean $dissemination\_completed_i \leftarrow false$
$\mathsf{Map}(\mathsf{Process} \rightarrow [\mathsf{RS}, \mathsf{Digest}, \mathsf{Witness}])$ $symbols_i \leftarrow$ empty map
$\mathsf{List}(\mathsf{Digest})$ $candidates_i \leftarrow$ empty list ▷ will be reset every iteration
$\mathsf{List}(\mathsf{Hash})$ $hashed\_candidates_i \leftarrow$ empty list ▷ will be reset every iteration
Digest $adopted\_digest_i \leftarrow \perp$
$\mathsf{List}(\mathsf{Value}_{\mathsf{MVBA}})$ $quasi\_decisions_i \leftarrow$ empty list

upon $\mathsf{propose}(\mathsf{Value}_{\mathsf{MVBA}}\ v_i)$: ▷ start of the algorithm
Execute the dissemination phase identical to that of Reducer (lines 19-37 of Alg. 1) except that values are treated as polynomials of degree ϵt

upon receiving $\langle$FINISH$\rangle$ from $n - t$ processes (for the first time):
$dissemination\_completed_i \leftarrow true$ ▷ dissemination phase completes
for each $k = 1, 2, ...$:
$candidates_i \leftarrow$ empty list; $hashed\_candidates_i \leftarrow$ empty list ▷ reset the candidates
$\mathsf{Process}$ $\mathsf{leader}(k) \leftarrow \mathsf{Election}()$ ▷ elect a random leader
broadcast $\langle$STORED$, k, symbols_i[\mathsf{leader}(k)].\mathsf{digest}()\rangle$ ▷ disseminate the leader's digest
wait for $n - t = (2 + \epsilon)t + 1$ STORED messages for iteration k
for each Digest z included in $n - 3t = \epsilon t + 1$ received STORED messages:
$candidates_i.\mathsf{append}(z)$
broadcast $\langle$SUGGEST$, k, candidates_i\rangle$ ▷ disseminate p_i's candidates
wait for $n - t = (2 + \epsilon)t + 1$ SUGGEST messages for iteration k
for each Digest $z \in candidates_i$:
if z is not included in $n - 2t = (1 + \epsilon)t + 1$ received SUGGEST messages:
$candidates_i.\mathsf{remove}(z)$
for each $x = 1, 2, ..., C$:
Integer $\phi \leftarrow \mathsf{Noise}()$ ▷ obtain a random λ-bit value ϕ
if $candidates_i.\mathsf{size} = 0$: $adopted\_digest_i \leftarrow default$ ▷ if no digest is committed, then adopt *default*
else:
for each $j = 1, 2, ..., candidates_i.\mathsf{size}$:
$hashed\_candidates_i[j] = \mathsf{hash}(candidates_i[j], \phi)$
▷ find the smallest hashed candidate and adopt the associated digest
Sort $hashed\_candidates_i$ in the lexicographic order
Hash $smallest_i \leftarrow hashed\_candidates_i[1]$
Let $z' \in candidates_i$ be the digest such that $smallest_i = \mathsf{hash}(z', \phi)$
$adopted\_digest_i \leftarrow z'$
▷ Reconstruct & Agree
broadcast $\langle$RECONSTRUCT$, k, x, symbols_i[\mathsf{leader}(k)]\rangle$
wait for $n - t = (2 + \epsilon)t + 1$ RECONSTRUCT messages for sub-iteration (k, x)
$\mathsf{Set}(\mathsf{RS})$ $S_i \leftarrow$ the set of received RS symbols with valid witnesses for $adopted\_digest_i$
if $|S_i| \geq \epsilon t + 1$: $\mathsf{Value}_{\mathsf{MVBA}}$ $r_i \leftarrow \mathsf{decode}(S_i)$ ▷ if a value can be decoded, set r_i to the decoded value
else: $\mathsf{Value}_{\mathsf{MVBA}}$ $r_i \leftarrow v_i$ ▷ if not, set r_i to p_i's proposal
$\mathsf{Value}_{\mathsf{MVBA}} \cup \{\perp_{\mathsf{MBA}}\}$ $v \leftarrow \mathcal{MBA}[k][x].\mathsf{propose}(r_i)$ ▷ propose r_i
if $\mathsf{valid}(v) = true$: $quasi\_decisions_i.\mathsf{append}(v)$ ▷ quasi-decide v
if $quasi\_decisions_i.\mathsf{size} > 0$ and p_i has not previously decided:
Integer $I \leftarrow \mathsf{Index}()$ ▷ obtain a random integer I in the $[1, C]$ range
trigger $\mathsf{decide}(quasi\_decisions_i[(I \bmod quasi\_decisions_i.\mathsf{size}) + 1])$ ▷ for quality

$z^\star(k)$ is committed by every correct process and $|\mathsf{committed}(k)| \leq C$), resulting in at least one fair probabilistic trial that the adversary cannot rig.

More specifically, the following holds for some trial $\mathcal{T}$: $\mathsf{start}(\mathcal{T}) = \mathsf{end}(\mathcal{T})$, where $\mathsf{start}(\mathcal{T})$ (resp., $\mathsf{end}(\mathcal{T})$) denotes the set of all digests committed by any correct process before (in global time) the first correct process starts (resp., ends) trial $\mathcal{T}$. Note that if a correct process adopts the "good" digest $z^\star(k)$, the process rebuilds value $v^\star(k)$ (line 49) as it necessarily receives $n - 3t = \epsilon t + 1$ RECONSTRUCT messages for digest $z^\star(k)$. Therefore, the probability that all correct processes adopt the "good" digest $z^\star(k)$ and input $v^\star(k)$ to the MBA primitive (of trial $\mathcal{T}$) before the first correct process ends trial $\mathcal{T}$—which is enough to guarantee that $v^\star(k)$ is decided due to the strong unanimity property of the MBA primitive—is given by

$$\frac{1}{|\mathsf{start}(\mathcal{T})|} = \frac{1}{|\mathsf{end}(\mathcal{T})|} \geq \frac{1}{C}$$

since $\mathsf{start}(\mathcal{T}) = \mathsf{end}(\mathcal{T}) \subseteq \mathsf{committed}(k)$ and $|\mathsf{committed}(k)| \leq C$. If this indeed happens, Reducer++ ensures that all correct processes (quasi-)decide $v^\star(k)$ in trial $\mathcal{T}$ and terminate. A formal argument for Reducer++'s termination can be found in §D.

Quality. If (1) the first iteration is good, (2) correct processes quasi-decide the valid non-adversarial value $v^\star(1)$, and (3) the Index() request (line 54) selects $v^\star(1)$ as the final decision, a non-adversarial value is indeed decided from Reducer++. Hence, this occurs with a probability of:

$$P \geq \frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \cdot \frac{1}{C} \cdot \frac{1}{C} \approx \frac{1}{3C^2}.$$

As a result, the probability that an adversarial value is decided is $1 - P \leq 1 - \frac{1}{3C^2} < 1$.

Complexity. Next, we provide a proof sketch of the following theorem.

Theorem 4 (Reducer++'s expected complexity)**.** *Given $n = (3+\epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, the following holds for Reducer++ (see Alg. 2) in the presence of a computationally bounded adversary:*

- *The expected message complexity is $O(n^2C^2)$.*
- *The expected bit complexity is $O\big(C^2(n\ell + n^2\lambda \log n)\big)$.*
- *The expected time complexity is $O(C^2)$.*

Let us now informally prove the theorem.

Expected time complexity. As previously discussed, Reducer++ terminates in a good iteration with at least $\frac{1}{C}$ probability. Given that each iteration is good with a probability of $P \geq \frac{(1+\epsilon)t+1}{(3+\epsilon)t+1} \approx \frac{1}{3}$, Reducer++ terminates in expected $3C$ iterations. As (1) each iteration takes $O(C)$ time to complete in expectation, and (2) the dissemination phase consists of $O(1)$ rounds of communication, the expected time complexity of Reducer++ is $O(C^2)$.

Expected message complexity. As (1) each iteration exchanges $O(n^2C)$ messages in expectation, and (2) Reducer++ terminates in $O(C)$ iterations in expectation, correct processes exchange $O(n^2C^2)$ messages in expectation through the iterations of Reducer++. Finally, correct processes exchange $O(n^2C^2)$ messages in expectation throughout the entire Reducer++ protocol as the dissemination phase exchanges only $O(n^2)$ messages.

Expected bit complexity. The dissemination phase exchanges $O(n\ell + n^2\lambda \log n)$ bits. Moreover, each iteration exchanges $O\big(C(n\ell + n^2\lambda \log n)\big)$ bits in expectation. Given that there are $O(C)$ iterations in expectation, the expected bit complexity of Reducer++ is

$$\underbrace{O(n\ell + n^2\lambda \log n)}_{\text{dissemination phase}} + O(C) \cdot \underbrace{O\big(C(n\ell + n^2\lambda \log n)\big)}_{\text{each iteration}} \subseteq O\big(C^2(n\ell + n^2\lambda \log n)\big).$$

7 Related Work

First, we examine earlier results on Byzantine agreement in asynchronous settings (§7.1). Second, we compare our techniques with those of closely-related works (§7.2).

7.1 Earlier Results

Byzantine agreement in the full information model with an adaptive adversary. Ben-Or [7] introduced the first solution to the asynchronous Byzantine agreement problem. In Ben-Or's algorithm, each process relies on a local coin. This algorithm has the notable advantage of operating in the *full information model* (where the adversary is aware of the internal states of all processes) and tolerating an adaptive adversary. However, since each process uses a local coin—ensuring an agreement probability of 2^{-n}—the algorithm suffers from exponential latency. Solving asynchronous Byzantine agreement in the full information model with an adaptive adversary, and without assuming a common coin, remains a significant challenge, as highlighted in several works [35, 36, 38, 39, 40, 43]. A major breakthrough occurred in 2018 with the first polynomial-time algorithm offering linear resilience [40], which corrected a technical flaw made in an earlier result [39]. Despite this progress, the resilience of the solution in [40] was limited to $1.14 \cdot 10^{-9} \cdot n$. More recently, the first polynomial-time algorithm achieving optimal resilience for this model was presented in [36], building on a near-optimal-resilience result from the same authors [35]. However, this achievement comes with a significant drawback: an expected time complexity of $\tilde{O}(n^{12})$, which is prohibitively high for practical applications.

Byzantine agreement with private channels. Faced with the significant challenges of solving Byzantine agreement in the full information model, the research community shifted its focus to a model incorporating *private channels*. In this alternative model, algorithms typically depend on a *weak common coin*, which permits a constant probability of disagreement on a random value. The use of a weak common coin for designing Byzantine agreement protocols was pioneered by Canetti and Rabin [17], who proposed an information-theoretically and adaptively secure asynchronous binary agreement algorithm with $O(n^7)$ bit complexity and $O(1)$ time complexity. Building on this approach, Abraham *et al.* [3] developed an adaptively secure asynchronous common subset (ACS) algorithm with $O(n^3\lambda)$ bit complexity and constant time complexity, relying on public-key cryptography (where λ represents the signature size). More recently, Abraham *et al.* [1] introduced a statistically secure ACS protocol with $O(n^5)$ bit complexity and $O(1)$ time complexity, designed for $t < \frac{1}{4}n$. To achieve optimal one-third resilience, this protocol incorporates asynchronous verifiable secret sharing (AVSS). While it is formally proven to be secure against a static adversary, the authors conjecture that their protocol can be extended to withstand an adaptive adversary as well. Similarly, the hash-based ACS protocol proposed by Das *et al.* [24] employs a weak common coin, achieving $O(1)$ time complexity and $O(n^3\lambda)$ bit complexity, but its security is limited to static adversaries.

Byzantine agreement with an idealized common coin. It has become standard practice [29, 28, 51, 42, 30] when constructing asynchronous Byzantine agreement protocols to do so 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 unbiasable random sequence to all processes. The rest of the protocol is the actual deterministic distributed-computing "core mechanism". A common coin can be implemented without a trusted dealer (assumed in the pioneering work of Rabin [48]) by utilizing threshold cryptography [15]. Moreover, the literature contains some dedicated common

coin protocols [27, 33, 6, 5, 50]. As discussed in §4, Reducer and Reducer++ can also be made to work with only weak common coins.

MVBA algorithms. The MVBA problem was introduced in [14] 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 [34]. Dumbo-MVBA [42] 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 limitations have sparked growing interest in designing adaptively-secure MVBA protocols that are hash-based from the ground up (assuming a common-coin object), the state-of-the-art of which are HMVBA [29], FIN-MVBA [28], and FLT24-MVBA [30] (see Tab. 1). HMVBA 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 $O(n^2\ell + n^3\lambda)$ or $O(n^2\ell + n^2\lambda + n^3 \log n)$ bit complexity. FIN-MVBA itself is an improvement over the hash-based MVBA protocol implied by the distributed key generation protocol of [27, 26], which suffers from $O(\log n)$ time complexity, and is only secure against static adversaries. Both HMVBA and FIN-MVBA satisfy the quality property. FLT24-MVBA [30] achieves optimal resilience, tolerating up to one-third faulty processes. Moreover, FLT24-MVBA exchanges $O(n\ell+n^2\lambda \log n+n^2\lambda\kappa)$ bits and terminates in $O(\log \kappa)$ time, where κ denotes a statistical security parameter (that cannot be treated as a constant, as explained in §7.2). It is also important to mention that FLT24-MVBA does not satisfy the quality property (see §1).

Other Byzantine agreement solutions. Beyond MVBA, Byzantine agreement primitives such as multi-valued Byzantine agreement (MBA), asynchronous common subset (ACS), and atomic broadcast (ABC) are well-studied in distributed systems. Mostefaoui *et al.* [46, 45] introduced a cryptography-free asynchronous MBA protocol with optimal resilience, $O(n^2\ell)$ bits and $O(1)$ time. In [47], 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 [52] 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, [28] presents a hash-based ACS protocol with $O(1)$ time complexity and $O(n^2\ell + n^3\lambda)$ bit complexity. Similarly, [1] achieves ACS with $O(1)$ time complexity without assuming a common coin, but only with $1/4$ resilience and $O(n^4 \log n)$ bit complexity; it is worth noting that [1] is safe against an adaptive adversary. Moreover, it is worth mentioning that [1] proposes a similar result with statistical security for $t < n/3$. Constructing ABC from MVBA is possible, as shown in [14]. ABC constructed from FIN-MVBA has $O(n^3)$ message complexity, which [51] 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.

7.2 Our Techniques vs. Techniques of Closely-Related Works

FLT24-MVBA [30]. The FLT24-MVBA algorithm starts with the dissemination phase in which each process disseminates its value using a method similar (but not identical) to the approach employed by HMVBA and our algorithms. The dissemination phase ensures that if a process p_i successfully disseminates its proposal, then at least $n - 2t$ correct processes can reconstruct it even if process p_i later gets corrupted. Then, processes elect κ leaders $L_1, L_2, ..., L_\kappa$ whose values they try to reconstruct; κ denotes a statistical security parameter.

The crux of the FLT24-MVBA algorithm is a primitive called synchronized multi-valued broadcast (SMB); this primitive is inspired by the MV broadcast primitive introduced by Mostefaoui *et al.* [46]. The primitive ensures that if $n - 2t$ correct processes broadcast the same input v, then all correct processes output a set containing at most two values. Furthermore, the outputs of any two correct processes are guaranteed to overlap: if one process outputs a single value, that value must appear in the other process's output set. If both processes output two values, their sets are identical. The SMB primitive acts as a robust filtering mechanism, ensuring that processes consider at most two different valid values per each elected leader.

Let us focus on a specific leader L_j. To select one of the two values as the final output of leader L_j, FLT24-MVBA employs an asynchronous reliable consensus (ARC) protocol. ARC ensures agreement among processes and guarantees termination if all correct processes share the same input value. Since SMB may leave two possible values, all correct processes participate in two parallel ARC protocols, one for each value. However, a termination issue may arise if one of the two values is not held by all correct processes. To address this, the algorithm utilizes a standard technique [23] based on the asynchronous binary agreement (ABA) primitive: an ABA protocol follows each ARC instance in order to decide whether that value (out of the two) should be selected. If a process outputs 1 in one ABA instance, it proposes 0s to all other ABA instances whose corresponding ARC counterparts have not yet terminated. Finally, as there exists at least one "good" leader L (except with negligible probability in κ), L's SMB instance will terminate, allowing all correct processes to reach consensus.

Comparison with FLT24-MVBA. The filtering SMB step and our reducing step serve the same fundamental purpose, making our algorithms similar in this regard. However, a key difference lies in how termination is handled. In FLT24-MVBA, the SMB-ARC-ABA sequence fails to terminate if it is tied to a "bad" leader. This constraint prevents FLT24-MVBA from adopting our "one-leader-per-iteration" structure, as electing a bad leader—which happens with constant probability—would then cause the entire algorithm to stall indefinitely.

To address this problem, FLT24-MVBA incorporates a statistical security parameter κ. It is essential to understand why κ cannot be treated as a constant. First, note that, if no "good" leader is elected, FLT24-MVBA stalls indefinitely, i.e., fails to terminate. Consequently, the protocol solves the MVBA problem with probability $1 - c^{\kappa}$, where c represents the fraction of "bad" leaders. Only treating κ as a security parameter (rather than as a constant) results in the desired negligible (in κ) error probability, and allows for instance the algorithm to be used in a polynomial (in κ) composition while maintaining this negligible error probability (e.g., in the "repeated MVBA" construction of atomic broadcast or state-machine replication; see §1). In contrast, treating κ as a constant would result in a constant error probability, making the algorithm ill-suited for composition.

The current design of FLT24-MVBA, where "the fastest leader wins", prevents the protocol from ensuring quality. For example, a single adaptive corruption can lead to the decision of an adversarial value. If a fast, corrupted leader propagates an adversarial value and quickly moves through the SMB, ARC, and ABA phases, correct processes are forced to follow it. To these processes, the leader may appear correct and be seen as their only path to termination. As this leader is significantly faster than others, only the ABA instance tied to its adversarial value will decide on 1, while all other instances will decide 0, ultimately forcing correct processes to decide on the adversarial value. In contrast, our algorithms guarantee that the original valid proposal of a "good" leader is eventually decided (with at least some constant probability), thereby ensuring quality—the importance of which is discussed in §1.

Comparison with [46]. While the ideas of [46] share the same spirit as ours, they are insufficient for our algorithms. To elaborate, [46] introduces two broadcasting primitives: (1) the reducing

(RD) broadcast primitive that reduces the number of values to a constant, and (2) the multi-valued validated (MV) broadcast primitive that outputs a set of values such that, if the output set of a correct process contains a single value v, then the output set of any other correct process contains v. (Recall that the MV broadcast primitive served as inspiration for the SMB primitive of FLT24-MVBA [30].) Crucially for our discussion, both the RD and MV broadcast primitives guarantee a delivery of a given value v only if *all* correct processes broadcast v. In other words, the RD and MV broadcast primitives "preserve" a value only if all correct processes hold it. If a majority—but not all—of the correct processes broadcast v, these primitives do not guarantee that any correct process will deliver v, meaning v might not be preserved. This design aligns with the focus of [46] on multi-valued Byzantine agreement (MBA; see Module 2), which satisfies (only) strong unanimity: if all correct processes propose the same value, that value must be decided; otherwise, any value can be decided. In this context, the RD and MV broadcast primitives are sufficient, as the MBA problem only requires preserving a value when all correct processes propose it.

Our algorithms, on the other hand, necessitate the preservation of a value even when it is not held by all correct processes, which represents the primary reason why the techniques of [46] cannot be directly applied to our algorithms. To clarify, the main sub-problem in our algorithms can be seen as follows: If $n - 2t$ correct processes hold a "good" digest z in a good iteration, each correct process must obtain z and rebuild the corresponding "good" value v (which occurs only with constant probability in Reducer++). This must be satisfied even if other correct processes hold adversarial non-z digests, as such an attack can occur in any good iteration. As a result, our algorithms require stronger techniques (SMBA in Reducer and hash-based adoption procedure in Reducer++) than those of [46]: as only $n - 2t$ (and not all!) correct processes hold a "good" digest, RD and MV broadcasts may never deliver z, thus preventing termination. Finally, we design our own MBA algorithm instead of using the one from [46] as that one requires $O(n^2\ell)$ bits.

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. In: TCC (4). Lecture Notes in Computer Science, vol. 15367, pp. 451–482. Springer (2024)

[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., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: PODC. pp. 363–373. ACM (2021)

[4] Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: PODC. pp. 337–346. ACM (2019)

[5] Bacho, R., Lenzen, C., Loss, J., Ochsenreither, S., Papachristoudis, D.: Grandline: Adaptively secure DKG and randomness beacon with (log-)quadratic communication complexity. In: CCS. pp. 941–955. ACM (2024)

[6] Bandarupalli, A., Bhat, A., Bagchi, S., Kate, A., Reiter, M.K.: Random beacons in monte carlo: Efficient asynchronous random beacon *without* threshold cryptography. In: CCS. pp. 2621–2635. ACM (2024)

[7] Ben-Or, M.: Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In: PODC. pp. 27–30. ACM (1983)

[8] Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distributed Comput. **16**(4), 249–262 (2003)

[9] Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC. pp. 183–192. ACM (1994)

[10] Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper - reconfiguration-friendly random beacons with quadratic communication. In: CCS. pp. 3502–3524. ACM (2021)

[11] Bracha, G.: Asynchronous byzantine agreement protocols. Inf. Comput. **75**(2), 130–143 (1987)

[12] Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on BFT consensus. arXiv:1807.04938v3 [cs.DC] (2018), http://arxiv.org/abs/1807.04938v3

[13] Cachin, C., Guerraoui, R., Rodrigues, L.E.T.: Introduction to Reliable and Secure Distributed Programming (2. ed.). Springer (2011)

[14] Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO. Lecture Notes in Computer Science, vol. 2139, pp. 524–541. Springer (2001)

[15] Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. J. Cryptol. **18**(3), 219–246 (2005)

[16] Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: SRDS. pp. 191–202. IEEE Computer Society (2005)

[17] Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: STOC. pp. 42–51. ACM (1993)

[18] Castro, M., Liskov, B.: Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst. **20**(4), 398–461 (2002)

[19] Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 7778, pp. 55–72. Springer (2013)

[20] Chen, J.: Ociormvba: Near-optimal error-free asynchronous mvba. arXiv:2501.00214v1 [cs.CR] (2024), http://arxiv.org/abs/2501.00214v1

[21] Civit, P., Dzulfikar, M.A., Gilbert, S., Guerraoui, R., Komatovic, J., Vidigueira, M., Zablotchi, I.: Partial synchrony for free: New upper bounds for byzantine agreement. In: SODA. pp. 4227–4291. SIAM (2025)

[22] 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

[23] 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)

[24] Das, S., Duan, S., Liu, S., Momose, A., Ren, L., Shoup, V.: Asynchronous consensus without trusted setup or public-key cryptography. In: CCS. pp. 3242–3256. ACM (2024)

[25] Das, S., Xiang, Z., Ren, L.: Asynchronous data dissemination and its applications. In: CCS. pp. 2705–2721. ACM (2021)

[26] 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

[27] 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)

[28] Duan, S., Wang, X., Zhang, H.: FIN: practical signature-free asynchronous common subset in constant time. In: CCS. pp. 815–829. ACM (2023)

[29] 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

[30] Feng, H., Lu, Z., Tang, Q.: $\widetilde{\text{o}}$ptimal adaptively secure hash-based asynchronous common subset. Cryptology ePrint Archive, Paper 2024/1710 (2024), https://eprint.iacr.org/2024/1710

[31] Fischer, M.J., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J. ACM **32**(2), 374–382 (1985)

[32] Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: PODC. pp. 211–220. ACM (2003)

[33] 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)

[34] 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)

[35] Huang, S., Pettie, S., Zhu, L.: Byzantine agreement in polynomial time with near-optimal resilience. In: STOC. pp. 502–514. ACM (2022)

[36] Huang, S., Pettie, S., Zhu, L.: Byzantine agreement with optimal resilience via statistical fraud detection. In: SODA. pp. 4335–4353. SIAM (2023)

[37] Katz, J., Lindell, Y.: Introduction to Modern Cryptography. Chapman and Hall/CRC Press (2007)

[38] Kimmett, B.: Improvement and partial simulation of King & Saia's expected-polynomial-time Byzantine agreement algorithm. Ph.D. thesis, University of Victoria, Canada (2020)

[39] King, V., Saia, J.: Byzantine agreement in expected polynomial time. J. ACM **63**(2), 13:1–13:21 (2016)

[40] King, V., Saia, J.: Correction to byzantine agreement in expected polynomial time, jacm 2016. arXiv:1812.10169v2 [cs.DC] (2018), `http://arxiv.org/abs/1812.10169v2`

[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] 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)

[43] Melnyk, D.: Byzantine Agreement on Representative Input Values Over Public Channels. Ph.D. thesis, ETH Zurich, Zürich, Switzerland (2020)

[44] Merkle, R.C.: A digital signature based on a conventional encryption function. In: CRYPTO. Lecture Notes in Computer Science, vol. 293, pp. 369–378. Springer (1987)

[45] 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)

[46] 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)

[47] 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)

[48] Rabin, M.O.: Randomized byzantine generals. In: FOCS. pp. 403–409. IEEE Computer Society (1983)

[49] 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

[50] 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)

[51] 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`

[52] Zhang, H., Duan, S.: PACE: fully parallelizable BFT from reproposable byzantine agreement. In: CCS. pp. 3151–3164. ACM (2022)

A MBA_ℓ: Pseudocode & Proof

This section presents MBA_ℓ, our adaptively-secure asynchronous MBA protocol employed by both Reducer and Reducer++. MBA_ℓ exchanges $O(n^2)$ messages and $O(n\ell + n^2\lambda \log n)$ bits, and terminates in $O(1)$ time. Moreover, MBA_ℓ tolerates up to $t < \frac{1}{3}n$ failures. Recall that the specification of the MBA primitive is given in Module 2.

Module 4 Graded Consensus

Associated values:
- set $\mathsf{Value}_{\mathsf{MBA}}$ of ℓ-bit values

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

Assumed behavior:
- Every correct process proposes exactly once.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *Strong unanimity:* If all correct processes propose the same value $v \in \mathsf{Value}_{\mathsf{MBA}}$ and a correct process decides a pair $(v' \in \mathsf{Value}_{\mathsf{MBA}}, g' \in \{0,1\})$, then $v' = v$ and $g' = 1$.
- *Consistency:* If any correct process decides a pair $(v \in \mathsf{Value}_{\mathsf{MBA}}, 1)$, then no correct process decides any pair $(v' \in \mathsf{Value}_{\mathsf{MBA}}, \cdot)$ with $v' \neq v$.
- *Justification:* If any correct process decides a pair $(v' \in \mathsf{Value}_{\mathsf{MBA}}, \cdot)$, then v' is proposed by a correct process.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.

Module 5 Binary Byzantine Agreement

Events:
- *input* $\mathsf{propose}(v \in \{0,1\})$: a process proposes binary value v.
- *output* $\mathsf{decide}(v' \in \{0,1\})$: a process decides binary value v'.

Assumed behavior:
- Every correct process proposes exactly once.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *Strong unanimity:* If all correct processes propose the same value $v \in \{0,1\}$ and a correct process decides a value $v' \in \{0,1\}$, then $v' = v$.
- *Agreement:* No two correct processes decide different values.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.

A.1 Pseudocode

The pseudocode of the MBA_ℓ algorithm is given in Alg. 3. Before discussing MBA_ℓ's pseudocode, we formally introduce the graded consensus and binary Byzantine agreement primitives employed in MBA_ℓ.

Graded consensus. The formal specification of the primitive is given in Module 4.

In our MBA_ℓ algorithm, we rely on a *deterministic* implementation [21] of the graded consensus primitive that exchanges $O(n^2)$ messages and $O(n\ell + n^2\lambda \log n)$ bits, and terminates in $O(1)$ time. This implementation tolerates up to $t < \frac{1}{3}n$ faulty processes and relies solely on a collision-resistant hash function. Note that, as the implementation is deterministic, it is inherently secure against an adaptive adversary.

Binary Byzantine agreement. The binary agreement primitive is similar to the MBA primitive (see Module 2). However, there are two major differences: (1) correct processes only propose 0 and 1, and (2) correct processes cannot decide a special value, i.e., only 0 and 1 can be decided from BA. The formal specification is given in Module 5.

In our MBA_ℓ algorithm, we utilize a binary agreement algorithm proposed by Mostefaoui and Raynal [45]; this algorithm is safe and live against an adaptive adversary, exchanges $O(n^2)$ messages and bits in expectation, terminates in $O(1)$ time in expectation, and relies on no cryptography.

Pseudocode description. When a correct process p_i proposes its value v_i (line 6), it forwards the

Algorithm 3 MBA_ℓ: Pseudocode (for process p_i)

```
1  Uses:
2      ▷ [21] exchanges O(n^2) messages and O(nℓ + n^2 λ log n) bits and terminates in O(1) time
3      Graded consensus algorithm [21] on the Value_MBA set, instance GC
4      ▷ [45] exchanges O(n^2) messages and O(n^2) bits and terminates in O(1) time
5      Binary Byzantine agreement algorithm [45], instance BA
6  upon propose(Value_MBA v_i):                                   ▷ start of the algorithm
7      (Value_MBA, {0,1}) (adopted_value, g) ← GC.propose(v_i)
8      {0,1} g' ← BA.propose(g)
9      if g' = 1:
10         trigger decide(adopted_value)
11     else:
12         trigger decide(⊥_MBA)
```

value to the $\mathcal{GC}$ graded consensus algorithm (line 7). Then, process p_i proposes the grade decided from $\mathcal{GC}$ to the $\mathcal{BA}$ binary Byzantine agreement algorithm (line 8). If the bit decided from $\mathcal{BA}$ is 1, then process p_i decides the value decided from $\mathcal{GC}$ (line 10). Otherwise, process p_i decides $\perp_{\mathsf{MBA}}$ (line 12).

A.2 Proof of Correctness & Complexity

This section proves MBA_ℓ's correctness and complexity.

Proof of correctness. To prove the correctness of MBA_ℓ, we prove the following lemma.

Lemma 1 (MBA_ℓ is correct)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) is a correct implementation of the MBA primitive in the presence of a computationally bounded adversary.*

In the rest of the section, we say that a correct process p_i *adopts* a value v if and only if $adopted\_value = v$ at process p_i when process p_i reaches line 8. First, we prove that if any correct process proposes 1 to the $\mathcal{BA}$ instance of the BA primitive, then no two correct processes adopt different values.

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

Proof. Let p_i be any correct process that proposes 1 to $\mathcal{BA}$. Hence, p_i decides with grade 1 from $\mathcal{GC}$. The statement of the claim then holds due to the consistency property of $\mathcal{GC}$. □

We are ready to prove MBA_ℓ's strong unanimity.

Proposition 1 (MBA_ℓ satisfies strong unanimity)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) satisfies strong unanimity in the presence of a computationally bounded adversary.*

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. Furthermore, each correct process proposes 1 to $\mathcal{BA}$, which ensures that all correct processes decide 1 from $\mathcal{BA}$ (due to its strong unanimity property). Finally, each correct process decides v. □

Next, we prove MBA_ℓ's agreement.

Proposition 2 (MBA_ℓ satisfies agreement)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) satisfies agreement in the presence of a computationally bounded adversary.*

Proof. Let $g^\star$ denote the binary value decided from $\mathcal{BA}$. We distinguish two cases:

- Let $g^\star = 0$. In this case, all correct processes that decide do so with $\perp_{\mathsf{MBA}}$. The agreement property is satisfied in this case.
- Let $g^\star = 1$. Therefore, every correct process decides its adopted value. Moreover, the strong unanimity property of $\mathcal{BA}$ ensures that a correct process proposes 1 to $\mathcal{BA}$. Therefore, the agreement is satisfied due to Claim 1.

As agreement is ensured in any possible case, the proof is concluded. □

We proceed by proving MBA_ℓ's justification.

Proposition 3 (MBA_ℓ satisfies justification)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) satisfies justification in the presence of a computationally bounded adversary.*

Proof. Suppose a correct process p_i decides a non-$\perp_{\mathsf{MBA}}$ value v. Hence, process p_i adopts v. Finally, the justification property of $\mathcal{GC}$ guarantees that v is proposed to MBA_ℓ by a correct process, which concludes the proof. □

Next, we prove the integrity property.

Proposition 4 (MBA_ℓ satisfies integrity)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) satisfies integrity in the presence of a computationally bounded adversary.*

Proof. The integrity property is satisfied due to the integrity property of $\mathcal{BA}$. □

Finally, we prove MBA_ℓ's termination.

Proposition 5 (MBA_ℓ satisfies termination)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, MBA_ℓ (see Alg. 3) satisfies termination in the presence of a computationally bounded adversary.*

Proof. The termination property is satisfied as both $\mathcal{GC}$ and $\mathcal{BA}$ terminate. □

Proof of complexity. We now proceed to prove MBA_ℓ's complexity.

Lemma 2 (MBA_ℓ's expected complexity)**.** *Given $t < \frac{1}{3}n$ and the existence of a collision-resistant hash function, the following holds for MBA_ℓ (see Alg. 3) in the presence of a computationally bounded adversary:*

- *The expected message complexity is $O(n^2)$.*
- *The expected bit complexity is $O(n\ell + n^2\lambda \log n)$.*
- *The expected time complexity is $O(1)$.*

Proof. As the worst-case message complexity of $\mathcal{GC}$ is $O(n^2)$ and the expected message complexity of $\mathcal{BA}$ is $O(n^2)$, the expected message complexity of MBA_ℓ is $O(n^2)$. Given that correct processes send $O(n\ell + n^2\lambda \log n)$ bits in $\mathcal{GC}$ and $O(n^2)$ bits in $\mathcal{BA}$, the expected bit complexity of MBA_ℓ is indeed $O(n\ell + n^2\lambda \log n)$. Finally, as both $\mathcal{GC}$ and $\mathcal{BA}$ terminate in $O(1)$ time in expectation, the expected time complexity of MBA_ℓ is $O(1)$. □

Module 6 CRB

Associated values:
- set Digest of $O(\lambda)$-bit digests
- special value $\perp_{\mathsf{CRB}} \notin \mathsf{Digest}$

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

Assumed behavior:
- Every correct process broadcasts exactly once.
- Only $O(1)$ different digests are broadcast by correct processes.

Properties: ▷ ensured only if correct processes behave according to the assumptions stated above
- *Validity:* If up to two different digests are broadcast by correct processes, no correct process delivers the special value $\perp_{\mathsf{CRB}}$.
- *Justification:* If any correct process delivers a digest $z' \in \mathsf{Digest}$ ($z' \neq \perp_{\mathsf{CRB}}$), then z' is broadcast by a correct process.
- *Integrity:* No correct process delivers unless it has previously broadcast.
- *Termination:* All correct processes deliver at least once.
- *Totality:* If any correct process delivers $z \in \mathsf{Digest} \cup \{\perp_{\mathsf{CRB}}\}$, then all correct processes eventually deliver z.

B SMBA_λ: Pseudocode & Proof

This section presents SMBA_λ, our adaptively-secure asynchronous SMBA protocol employed in $\mathsf{Reducer}$. SMBA_λ exchanges $O(n^2)$ messages and $O(n^2\lambda)$ bits, and terminates in $O(1)$ time. The specification of the SMBA primitive can be found in Module 3.

B.1 Collective Reliable Broadcast (CRB): Pseudocode & Proof

First, we introduce the CRB primitive that plays a major role in our SMBA_λ algorithm.

Primitive definition. The CRB primitive closely resembles the reliable broadcast primitive described in [13]. Its formal specification is provided in Module 6.

We emphasize that, per Module 6, any correct process may deliver multiple times. Moreover, the termination property ensures that each correct process delivers at least once.

B.1.1 Pseudocode

The pseudocode of CRB_λ, our CRB algorithm, can be found in Alg. 4. Our CRB_λ algorithm exchanges $O(n^2)$ messages and $O(n^2\lambda)$ bits, and terminates in $O(1)$ time. Moreover, the CRB_λ algorithm tolerates up to $t < \frac{1}{4}n$ faulty processes. We underline that the CRB_λ algorithm is heavily inspired by Bracha's reliable broadcast algorithm [11].

Pseudocode description. We describe the pseudocode of the CRB_λ algorithm from the perspective of a correct process p_i. Upon broadcasting its digest z_i (line 16), process p_i broadcasts an INIT message for z_i (line 17). Once process p_i receives the same digest z in (at least) $t+1$ INIT messages (line 18), which proves that z is broadcast by a correct process, process p_i broadcasts an ECHO message for z (line 19). Similarly, once process p_i receives the same digest z in (at least) $2t+1$ ECHO (line 20) or $t+1$ READY messages (line 22), process p_i broadcasts a READY message for z (line 21 or line 23). Finally, once process p_i receives $2t+1$ READY messages for the same digest (line 24), it delivers the digest (line 25).

Upon realizing that there are at least three different digests broadcast by correct processes (line 32), process p_i broadcasts a BROKEN message (line 33). Similarly, upon receiving $t+1$ BROKEN

Algorithm 4 CRB_λ: Pseudocode (for process p_i)

```
1  Rules:
2      - Only one INIT message is processed per process.
3      - At most one ECHO message is broadcast per digest.
4      - At most one READY message is broadcast per digest.
5      - At most one BROKEN message is broadcast.
6      - No computational steps are taken unless a digest was previously broadcast.
7  Local variables:
8      Digest z_i ← p_i's broadcast digest
9      Map(Digest → Integer) num_i ← {0, for every z ∈ Digest}
10 Local functions:
11     - distinct() = |{z ∈ Digest | num_i[z] > 0}|.
12     - eliminated():
13         (1) Let Z = {z ∈ Digest | num_i[z] > 0}.
14         (2) Sort Z in the ascending order according to the num_i map.
15         (3) Return the greatest integer x ∈ ℕ_≥0 such that num_i[Z[1]] + ... + num_i[Z[x]] ≤ t.
16 upon broadcast(Digest z_i):                                  ▷ start of the algorithm
17     broadcast ⟨INIT, z_i⟩
18 upon exists Digest z such that ⟨INIT, z⟩ is received from t + 1 processes:
19     broadcast ⟨ECHO, z⟩
20 upon exists Digest z such that ⟨ECHO, z⟩ is received from 2t + 1 processes:
21     broadcast ⟨READY, z⟩
22 upon exists Digest z such that ⟨READY, z⟩ is received from t + 1 processes:
23     Broadcast ⟨READY, z⟩
24 upon exists Digest z such that ⟨READY, z⟩ is received from 2t + 1 processes:
25     trigger deliver(z)
26 upon ⟨BROKEN⟩ is received from t + 1 processes:
27     broadcast ⟨BROKEN⟩
28 upon ⟨BROKEN⟩ is received from 2t + 1 processes:
29     trigger deliver(⊥_CRB)
30 upon receiving ⟨INIT, Digest z⟩:
31     num_i[z] ← num_i[z] + 1
32 upon (1) distinct() − eliminated() ≥ 3, and (2) ≥ n − t ≥ 3t + 1 INIT messages are received:
33     broadcast ⟨BROKEN⟩
```

messages (line 26), process p_i rebroadcasts the message (line 27). Once process p_i receives $2t + 1$ BROKEN messages (line 28), it delivers the special value $\perp_{\mathsf{CRB}}$ (line 29).

B.1.2 Proof of Correctness & Complexity

This subsection proves CRB_λ's correctness and complexity.

Proof of correctness. To prove CRB_λ's correctness, we prove the following lemma.

Lemma 3 (CRB_λ is correct)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) is a correct implementation of the CRB primitive in the presence of a computationally unbounded adversary.*

In the rest of the proof, let $\mathcal{B}_C$ denote the set of all digests broadcasts by correct processes. We start by proving that CRB_λ satisfies the validity property. To this end, we show that if there exists a correct process that broadcasts a BROKEN message at line 33, then $|\mathcal{B}_C| \geq 3$.

Claim 2. *If there exists a correct process that broadcasts a* BROKEN *message at line 33, then* $|\mathcal{B}_C| \geq 3$.

Proof. Let p_i be any correct process that broadcasts a BROKEN message at line 33. By contradiction, suppose $|\mathcal{B}_C| \leq 2$. Let *init* denote the set of all INIT messages received by p_i prior to broadcasting the BROKEN message at line 33. Let us define a set $\mathcal{I}$:

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

Note that $|\mathcal{I}| \geq n - t \geq 3t + 1$ (due to the second condition of the rule at line 32). We now define a set of digests *correct_digs* in the following way:

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

Let $X = |correct\_digs|$. 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 *faulty_digs*:

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

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

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

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 $\mathsf{eliminated}()$ function (line 14). We say that a digest z is *eliminated* if and only if (1) $z = Z[i]$, and (2) $i \in [1, \mathsf{eliminated}()]$. (If $\mathsf{eliminated}() = 0$, no digest is eliminated.)

As process p_i sends the BROKEN message at line 33, $\mathsf{distinct}() - \mathsf{eliminated}() \geq 3$ at process p_i (line 32). Therefore, there are at least three non-eliminated digests. We distinguish two possible scenarios:

- Let $X = 1$. Let $correct\_digs = \{z\}$, for some digest z. Note that $num_i[z] \geq 2t+1$ (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 $z' \in correct\_digs \cup faulty\_digs$ such that z precedes z' in Z. If such z' existed, $num_i[z'] \geq 2t + 1$, which further implies that $z' \in correct\_digs$. As $correct\_digs = \{z\}$, this is impossible. Therefore, z must be the last digest in the list Z.
  For every digest $z' \in faulty\_digs \setminus \{z\}$, all INIT messages for z' received by p_i are sent by faulty processes. Hence, every digest $z' \in faulty\_digs \setminus \{z\}$ is eliminated. Thus, $\mathsf{distinct}() - \mathsf{eliminated}() = |\{z\} \cup faulty\_digs| - |faulty\_digs \setminus \{z\}| = 1$, which represents a contradiction with the fact that $\mathsf{distinct}() - \mathsf{eliminated}() \geq 3$. This means that this case is impossible.
- Let $X = 2$. Let $correct\_digs = \{z_1, z_2\}$, for some digests z_1 and z_2. Note that $num_i[z_1] \geq t+1$ or $num_i[z_2] \geq t + 1$ (as there are at least $2t + 1$ INIT messages received by p_i from correct processes). Without loss of generality, let $num_i[z_1] \geq t + 1$. Therefore, the digest z_1 cannot be eliminated. Observe also that there cannot exist a digest $z_3 \in faulty\_digs \setminus correct\_digs$ such that z_1 precedes z_3 in Z. Indeed, for such digest z_3 to exist, $num\_i[z_3] \geq t + 1$, which then implies that z_3 must belong to $correct\_digs$. 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, $\mathsf{distinct}() - \mathsf{eliminated}() = 2$, which contradicts the fact that $\mathsf{distinct}() - \mathsf{eliminated}() \geq 3$. 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 $B_{eliminated} = \sum_{z \in eliminated} num_i[z]$. Note that $B_{eliminated} \leq t$ due to the definition of the eliminated digests. Importantly, as z_2 is not eliminated, we have $B_{eliminated} + num_i[z_2] \geq t+1$, which further implies $num_i[z_2] \geq t + 1 - B_{eliminated}$.
  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 $z_3 \in faulty\_digs \setminus correct\_digs$, $B_{eliminated}$ messages are used on the values preceding z_2 and there are F messages issued by faulty processes, $num_i[z_3] \leq F - B_{eliminated}$. Moreover, $num_i[z_3] \geq num_i[z_2]$ (as z_2 precedes z_3 in Z). This implies $num_i[z_2] \leq F - B_{eliminated}$. Hence, $t+1-B_{eliminated} \leq num_i[z_2] \leq F - B_{eliminated}$, 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 $\mathsf{distinct}() - \mathsf{eliminated}() \geq 3$ 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 $z' = Z[\mathsf{eliminated}() + 1]$ and $z'' = Z[\mathsf{eliminated}() + 2]$. 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 $z', z'' \in faulty\_digs \setminus correct\_digs$).
  Let *eliminated* denote the set of eliminated digests. For each digest $z \in eliminated$, 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 $z \in eliminated$, $num_i[z] = C_z + B_z$.) Note that $C_{z_2} > 0$ (as $z_2 \in correct\_digs$); for every digest $z \in eliminated \setminus \{z_2\}$, $C_z = 0$. Let $B_{eliminated} = \sum_{z \in eliminated} B_z$. Observe that $C_{z_2} + B_{eliminated} + num_i[z'] \geq t + 1$ as, otherwise, z' would also be eliminated. Therefore, $num_i[z'] \geq t+1-C_{z_2}-B_{eliminated}$. As $num_i[z''] \geq num_i[z']$, $num_i[z''] \geq t + 1 - C_{z_2} - B_{eliminated}$. Thus, $num_i[z'] + num_i[z''] \geq 2t + 2 - 2C_{z_2} - 2B_{eliminated}$. Moreover, $num_i[z'] + num_i[z''] \leq F - B_{eliminated}$ (as all INIT messages for z' or z'' are sent by faulty processes and there are already $B_{eliminated}$ faulty processes that sent INIT messages for digests different from z' and z''). Hence, we have:

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

  This implies $2t + 2 - 2C_{z_2} - 2B_{eliminated} \leq F - B_{eliminated}$, which implies $2C_{z_2} \geq 2t + 2 - B_{eliminated} - F$.
  Observe that $num_i[z''] \geq num_i[z'] \geq C_{z_2}$ (as neither z' nor z'' are eliminated, $num_i[z_2] \geq C_{z_2}$ and z_2 is eliminated). Therefore, $num_i[z'] + num_i[z''] \geq 2C_{z_2}$. Therefore, $F - B_{eliminated} \geq 2C_{z_2}$.
  Thus, we have $2t + 2 - B_{eliminated} - F \leq 2C_{z_2} \leq F - B_{eliminated}$. Therefore, $2t + 2 - B_{eliminated} - F \leq F - B_{eliminated}$, which implies $2t + 2 \leq 2F$. As $F \leq t$, this is impossible.

The claim holds as its statement holds in all possible cases. □

We are now ready to prove the CRB_λ satisfies validity.

Proposition 6 (CRB_λ satisfies validity)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) satisfies validity in the presence of a computationally unbounded adversary.*

Proof. Let $|\mathcal{B}_C| \leq 2$. By contradiction, suppose there exists a correct process p_i that delivers $\perp_{\mathsf{CRB}}$. Therefore, there exists a correct process that broadcasts a BROKEN message. The first correct process that broadcasts a BROKEN message does so at line 33. Hence, Claim 2 proves that $|\mathcal{B}_C| \geq 3$, which implies contradiction with $|\mathcal{B}_C| \leq 2$. Thus, CRB_λ satisfies validity. □

Next, we prove the termination property of CRB_λ. To do so, we first show that if there exists a digest z broadcast by at least $t + 1$ correct processes, then every correct process delivers z within $O(1)$ asynchronous rounds.

Claim 3. *Let there exist a digest z broadcast by at least $t+1$ correct processes. Then, every correct process delivers z within $O(1)$ asynchronous rounds.*

Proof. As $t + 1$ correct processes broadcast z, every correct process eventually receives $t + 1$ INIT messages for z (line 18) and broadcasts an ECHO message for z (line 19). (This occurs within a single message delay.) Therefore, every correct process eventually broadcasts a READY message for z (line 21) upon receiving an ECHO message for z from (at least) $2t + 1$ processes (line 20). (This incurs another message delay.) Finally, every correct process eventually receives a READY message for z from (at least) $2t+1$ processes (line 24) and delivers z (line 25) in $3 \in O(1)$ message delays. □

The following claim proves that all correct processes deliver the special value $\perp_{\mathsf{CRB}}$ within $O(1)$ asynchronous rounds given that (1) $|\mathcal{B}_C| > 3$, and (2) no digest is broadcast by $t + 1$ (or more) correct processes.

Claim 4. *If (1) $|\mathcal{B}_C| > 3$, and (2) no digest is broadcast by $t+1$ correct processes, then every correct process delivers $\perp_{\mathsf{CRB}}$ within $O(1)$ asynchronous rounds.*

Proof. To prove the claim, we prove that all correct processes broadcast a BROKEN message within $O(1)$ message delays, which then implies that all correct processes receive $2t + 1$ BROKEN messages (line 28) and deliver $\perp_{\mathsf{CRB}}$ (line 29) within $O(1)$ message delays. Let $|\mathcal{B}_C| = x > 3$. Without loss of generality, let $\mathcal{B}_C = \{z_1, z_2, ..., z_{x-1}, z_x\}$, for some digests $z_1, z_2, ..., z_x$. Consider any correct process p_i and time τ 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 τ; note that $|\mathcal{I}| \geq n - t \geq 3t + 1$ as there are at least $n - t \geq 3t + 1$ correct processes. Next, we define a set of digests *correct_digs* in the following way:

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

Note that $\textit{correct\_digs} = \{z_1, z_2, ..., z_{x-1}, z_x\}$. For each digest $z \in \textit{correct\_digs}$, 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 τ. Observe that, for every $z \in \textit{correct\_digs}$, $num_i[z] = C_z + B_z$. Moreover, $C_{z_1} + C_{z_2} + ... + C_{z_{x-1}} + C_{z_x} \geq n - t \geq 3t + 1$ as there are at least $n - t \geq 3t + 1$ correct processes.

Let Z denote the sorted list of digests constructed by the $\mathsf{eliminated}()$ function (line 14). We say that a digest z is *eliminated* if and only if (1) $z = Z[i]$, and (2) $i \in [1, \mathsf{eliminated}()]$. (If $\mathsf{eliminated}() = 0$, no digest is eliminated.) To prove the claim, it suffices to show that at most $x - 3$ digests from the *correct_digs* set are eliminated at time τ as this (along with the fact that

$|\mathcal{I}| \geq n - t \geq 3t + 1$) guarantees that the rule at line 32 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 $z_1, z_2, ..., z_{x-3}, z_{x-2}$ be eliminated at time τ. This implies that $num_i[z_1] + num_i[z_2] + ... + num_i[z_{x-2}] = (C_{z_1} + B_{z_1}) + (C_{z_2} + B_{z_2}) + ... + (C_{z_{x-2}} + B_{z_{x-2}}) \leq t$, which further implies that $C_{z_1} + C_{z_2} + ... + C_{z_{x-2}} \leq t$. As $C_{z_1} + C_{z_2} + ... + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$ and $C_{z_1} + C_{z_2} + ... + C_{z_{x-2}} \leq t$, $C_{z_{x-1}} + C_{z_x} \geq 2t + 1$. Therefore, $C_{z_{x-1}} \geq t + 1$ or $C_{z_x} \geq t + 1$, which contradicts the fact that no digest is broadcast by $t + 1$ correct processes.
- Suppose exactly $x - 1$ digests from the *correct_digs* set are eliminated. Without loss of generality, let $z_1, z_2, ..., z_{x-3}, z_{x-2}, z_{x-1}$ be eliminated at time τ. This implies that $num_i[z_1] + num_i[z_2] + ... + num_i[z_{x-2}] + num_i[z_{x-1}] = (C_{z_1} + B_{z_1}) + (C_{z_2} + B_{z_2}) + ... + (C_{z_{x-2}} + B_{z_{x-2}}) + (C_{z_{x-1}} + B_{z_{x-1}}) \leq t$, which further implies that $C_{z_1} + C_{z_2} + ... + C_{z_{x-2}} + C_{z_{x-1}} \leq t$. As $C_{z_1} + C_{z_2} + ... + C_{z_{x-1}} + C_{z_x} \geq 3t + 1$ and $C_{z_1} + C_{z_2} + ... + C_{z_{x-2}} + C_{z_{x-1}} \leq t$, $C_{z_x} \geq 2t + 1$. This contradicts the fact that no digest is broadcast by $t + 1$ correct processes.
- Suppose exactly x digests from the *correct_digs* set are eliminated. This is impossible as $num[z_1]+num[z_2]+...+num[z_x] \geq 3t+1$ due to the fact that $C_{z_1}+C_{z_2}+...+C_{z_{x-1}}+C_{z_x} \geq 3t+1$.

As neither of the aforementioned cases can occur, the claim holds. □

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

Proposition 7 (CRB_λ satisfies termination)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) satisfies termination in the presence of a computationally unbounded adversary. Precisely, every correct process delivers within $O(1)$ asynchronous rounds.*

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

- Let $|\mathcal{B}_C| \leq 3$. In this case, there exists a digest z broadcast by at least $t+1$ correct processes. Therefore, every correct process delivers z within $O(1)$ message delays (by Claim 3). 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 by $t+1$ correct processes. In this case, the termination property is ensured due to Claim 3.
  - Let there be no digest broadcast by $t + 1$ correct processes. The termination property holds in this case due to Claim 4.

As termination is satisfied in every possible case, the proposition holds. □

Note that Proposition 7 proves that the worst-case time complexity of CRB_λ is $O(1)$: correct processes deliver the first digest in $O(1)$ time. Next, we prove CRB_λ's integrity.

Proposition 8 (CRB_λ satisfies integrity)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) satisfies integrity in the presence of a computationally unbounded adversary.*

Proof. The integrity property is satisfied due to the rule at line 6. □

The following proposition proves CRB_λ's justification.

Proposition 9 (CRB_λ satisfies justification)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) satisfies justification in the presence of a computationally unbounded adversary.*

Proof. Suppose a correct process p_i delivers a digest $z \in \mathsf{Digest}$. Therefore, there exists a READY message for z broadcast by a correct process. Note that the first correct process to broadcast a READY message for z does so at line 21. Hence, there exists a correct process that broadcasts an ECHO message for z, which implies $z \in \mathcal{B}_C$ (as at least $t+1$ INIT messages for z are received). □

Finally, we prove that CRB_λ satisfies totality.

Proposition 10 (CRB_λ satisfies totality)**.** *Given $t < \frac{1}{4}n$, CRB_λ (see Alg. 4) satisfies totality in the presence of a computationally unbounded adversary. Precisely, if any correct process delivers $z \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$, then every correct process delivers z within $O(1)$ asynchronous rounds from the aforementioned delivery.*

Proof. Suppose a correct process p_i delivers $z \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$. We distinguish two possibilities:

- Let $z \in \mathsf{Digest}$. (This implies that $z \neq \bot_{\mathsf{CRB}}$.) 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 and broadcast a READY message for z. Thus, every correct process receives $2t+1$ READY messages for z within $O(1)$ message delays and delivers z.
- Let $z = \bot_{\mathsf{CRB}}$. Following the same argument as above (just applied to BROKEN messages), we conclude that every correct process delivers $\bot_{\mathsf{CRB}}$ within $O(1)$ message delays.

As the statement of the proposition holds in both cases, the proof is concluded. □

Observe that Proposition 10 proves that the totality property is satisfied within $O(1)$ asynchronous rounds from the first delivery.

Proof of complexity. We now prove CRB_λ's complexity.

Lemma 4 (CRB_λ's worst-case complexity)**.** *Given $t < \frac{1}{4}n$, the following holds for CRB_λ (see Alg. 4) in the presence of a computationally unbounded adversary:*

- *The worst-case message complexity is $O(n^2)$.*
- *The worst-case bit complexity is $O(n^2\lambda)$.*

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, process p_i broadcasts at most one INIT and at most one BROKEN message. Therefore, process p_i sends

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

Moreover, process 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 $O(n)$ messages and $O(n\lambda)$ bits, which implies that all correct processes send $O(n^2)$ messages and $O(n^2\lambda)$ bits. □

B.2 Pseudocode

The pseudocode of SMBA_λ is given in Alg. 5. Internally, SMBA_λ relies on (1) instances $\mathcal{MBA}_1$ and $\mathcal{MBA}_2$ of the MBA primitive run on digests (lines 3 and 5), and (2) an instance $\mathcal{CRB}$ of the CRB algorithm CRB_λ (line 7). Concretely, for $\mathcal{MBA}_1$ and $\mathcal{MBA}_2$, we rely on the MBA algorithm

Algorithm 5 SMBA_λ: Pseudocode (for process p_i)

```
1  Uses:
2      ▷ [46] exchanges O(n^2) messages and O(n^2 λ) bits and terminates in O(1) time
3      MBA algorithm [46] with Value_MBA = Digest ∪ {⊥_CRB}, instance MBA_1
4      ▷ [46] exchanges O(n^2) messages and O(n^2 λ) bits and terminates in O(1) time
5      MBA algorithm [46] with Value_MBA = Digest, instance MBA_2
6      ▷ CRB_λ exchanges O(n^2) messages and O(n^2 λ) bits
7      CRB algorithm CRB_λ, instance CRB
8  Constants:
9      Digest default                                              ▷ default digest
10 Local variables:
11     Digest z_i ← p_i's proposal
12     Set(Digest) delivered_i ← ∅
13 upon propose(Digest z_i):                                       ▷ start of the algorithm
14     invoke CRB.broadcast(z_i)
15 upon CRB.deliver(Digest ∪ {⊥_CRB} z):
16     delivered_i ← delivered_i ∪ {z}
17     if |delivered_i| = 1:
18         invoke MBA_1.propose(z)
19 upon MBA_1.decide(Digest ∪ {⊥_CRB, ⊥_MBA} z'):                  ▷ we assume ⊥_CRB ≠ ⊥_MBA
20     if z' ≠ ⊥_MBA:
21         Digest ∪ {⊥_CRB} z* ← z'
22         if z* = ⊥_CRB:
23             z* ← default
24         invoke MBA_2.propose(z*)
25     else:
26         wait for |delivered_i| = 2
27         Let z* be the lexicographically smallest digest (≠ ⊥_CRB) in delivered_i
28         invoke MBA_2.propose(z*)
29 upon MBA_2.decide(Digest ∪ {⊥_MBA} z''):
30     if z'' = ⊥_MBA:
31         z'' ← default
32     trigger decide(z'')
```

introduced in [46] that (1) exchanges $O(n^2)$ messages and $O(n^2\lambda)$ bits in expectation, and (2) terminates in $O(1)$ time in expectation.

Pseudocode description. We explain the pseudocode of our SMBA protocol SMBA_λ from the perspective of a correct process p_i. Once p_i proposes its digest z_i (line 13), process p_i broadcasts z_i via $\mathcal{CRB}$ (line 14). When p_i delivers the first digest (or the special value $\bot_{\mathsf{CRB}}$) from $\mathcal{CRB}$ (line 15), p_i proposes it to $\mathcal{MBA}_1$ (line 18). Let p_i decide $z' \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}, \bot_{\mathsf{MBA}}\}$ from $\mathcal{MBA}_1$ (line 19); we assume $\bot_{\mathsf{CRB}} \neq \bot_{\mathsf{MBA}}$. Now, we distinguish two cases:

- Let $z' \neq \bot_{\mathsf{MBA}}$. Note that the justification property of $\mathcal{MBA}_1$ ensures that z' was proposed by a correct process. We further investigate two scenarios:
  - Let $z' = \bot_{\mathsf{CRB}}$. In this case, process p_i proposes the default digest *default* to $\mathcal{MBA}_2$ (lines 23 and 24). Note that, as $\bot_{\mathsf{CRB}}$ is delivered from $\mathcal{CRB}$, there must exist more than two different digests proposed by correct processes (due to $\mathcal{CRB}$'s validity property). Therefore, this case does *not* require the decision to be proposed by a correct process.
  - Let $z' \neq \bot_{\mathsf{CRB}}$. Then, process p_i proposes z' to $\mathcal{MBA}_2$ (lines 21 and 24). Observe that the justification property of $\mathcal{CRB}$ guarantees that z' is proposed by correct process.

- Let $z' = \bot_{\mathsf{MBA}}$. The strong unanimity property of $\mathcal{MBA}_1$ guarantees that not all correct processes proposed the same value $z \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$ to $\mathcal{MBA}_1$. (Otherwise, $\bot_{\mathsf{MBA}}$ could not have been decided.) Thus, there are at least two different values $z_1, z_2 \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$ delivered from $\mathcal{CRB}$ by correct processes. In this case, process p_i waits to deliver the second digest (or $\bot_{\mathsf{CRB}}$) from $\mathcal{CRB}$ (line 26). Once that happens (and it will due to $\mathcal{CRB}$'s totality property), process p_i proposes to $\mathcal{MBA}_2$ the lexicographically smallest digest (thus, not $\bot_{\mathsf{CRB}}$!) it has delivered from $\mathcal{CRB}$ (lines 27 and 28).

Finally, once process p_i decides $z'' \in \mathsf{Digest} \cup \{\bot_{\mathsf{MBA}}\}$ from $\mathcal{MBA}_2$ (line 29), p_i decides (1) the default digest *default* if $z'' = \bot_{\mathsf{MBA}}$ (lines 31 and 32), or (2) z'' otherwise (line 32).

B.3 Proof of Correctness & Complexity

This subsection formally proves the correctness and complexity of SMBA_λ.

Proof of correctness. We first prove the following lemma.

Lemma 5 (SMBA_λ is correct)**.** *Given $t < \frac{1}{4}n$, SMBA_λ (see Alg. 5) is a correct implementation of the SMBA primitive in the presence of a computationally unbounded adversary.*

We start by proving the agreement property of SMBA_λ.

Proposition 11 (SMBA_λ satisfies agreement)**.** *Given $t < \frac{1}{4}n$, SMBA_λ (see Alg. 5) satisfies agreement in the presence of a computationally unbounded adversary.*

Proof. The agreement property follows directly from the agreement property of $\mathcal{MBA}_2$. □

Next, we prove that SMBA_λ satisfies strong validity. In the rest of the proof, let $\mathcal{Z}_C$ denote the set of digests proposed by correct processes. We start by proving that only constantly many different digests are broadcast by correct processes via $\mathcal{CRB}$. (Recall that this assumption is required by $\mathcal{CRB}$; see Module 6.)

Claim 5. *Only $O(1)$ different digests are broadcast by correct processes via $\mathcal{CRB}$.*

Proof. The claim follows directly from the assumption that only $O(1)$ different digests are proposed by correct processes (see Module 3). □

Claim 5 proves that $\mathcal{CRB}$ behaves according to its specification. (To not pollute the presentation of the proof, we might not explicitly rely on Claim 5 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 $\mathcal{MBA}_2$ with $z \in \mathcal{Z}_C$.

Claim 6. *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 $\mathcal{MBA}_2$.*

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 $\bot_{\mathsf{CRB}}$ from $\mathcal{CRB}$. Moreover, the termination property of $\mathcal{CRB}$ ensures that all correct processes eventually deliver from $\mathcal{CRB}$ and, thus, propose to $\mathcal{MBA}_1$. Hence, the termination and agreement properties of $\mathcal{MBA}_1$ ensure that all correct processes eventually decide $z' \in \mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}, \bot_{\mathsf{MBA}}\}$ from $\mathcal{MBA}_1$. We distinguish two cases:

- Let $z' \neq \bot_{\mathsf{MBA}}$. The justification 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) $z' \neq \bot_{\mathsf{CRB}}$, and (2) $z' \in \mathcal{Z}_C$ due to $\mathcal{CRB}$'s justification property. This means that every correct process proposes z' to $\mathcal{MBA}_2$. Hence, the statement of the claim holds in this case.

- Let $z' = \bot_{\mathsf{MBA}}$. The strong unanimity property of $\mathcal{MBA}_1$ proves that at least two different values from the $\mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$ 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 $z_1 \neq \bot_{\mathsf{CRB}}$ and $z_2 \neq \bot_{\mathsf{CRB}}$. 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 $z_3 \neq \bot_{\mathsf{CRB}}$ 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) $\bot_{\mathsf{CRB}}$ as no correct process ever delivers $\bot_{\mathsf{CRB}}$ due to $\mathcal{CRB}$'s validity property (recall that $|\mathcal{Z}_C| \leq 2$). Hence, $\mathit{delivered}_i = \{z_1, z_2\}$ when $|\mathit{delivered}_i| = 2$ at process p_i. 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 claim holds even in this case.

The claim holds as its statement is satisfied in both possible scenarios. □

We are ready to prove that SMBA_λ satisfies strong validity.

Proposition 12 (SMBA_λ satisfies strong validity)**.** *Given $t < \frac{1}{4}n$, SMBA_λ (see Alg. 5) satisfies strong validity in the presence of a computationally unbounded adversary.*

Proof. Let $|\mathcal{Z}_C| \leq 2$. Let p_i be any correct process that decides some digest z'. Claim 6 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 $z \neq \bot_{\mathsf{MBA}}$ from $\mathcal{MBA}_2$. Thus, $z' = z$, which implies $z' \in \mathcal{Z}_C$. □

Next, we prove the integrity property of SMBA_λ.

Proposition 13 (SMBA_λ satisfies integrity)**.** *Given $t < \frac{1}{4}n$, SMBA_λ (see Alg. 5) satisfies integrity in the presence of a computationally unbounded adversary.*

Proof. The proposition holds due to the integrity property of $\mathcal{MBA}_2$. □

Finally, we prove the termination property of SMBA_λ.

Proposition 14 (SMBA_λ satisfies termination)**.** *Given $t < \frac{1}{4}n$, SMBA_λ (see Alg. 5) satisfies termination in the presence of a computationally unbounded adversary. Precisely, every correct process decides within $O(1)$ asynchronous rounds in expectation.*

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

- Let $z' \neq \bot_{\mathsf{MBA}}$. In this case, all correct processes propose to $\mathcal{MBA}_2$ within $O(1)$ time in expectation.
- Let $z' = \bot_{\mathsf{MBA}}$. The strong unanimity property of $\mathcal{MBA}_1$ proves that at least two different values from the $\mathsf{Digest} \cup \{\bot_{\mathsf{CRB}}\}$ 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 Proposition 10). 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, SMBA_λ indeed terminates in $O(1)$ expected time, which concludes the proof. □

Proof of complexity. To prove SMBA_λ's complexity, we prove the following lemma.

Lemma 6 (SMBA_λ's expected complexity)**.** *Given $t < \frac{1}{4}n$, the following holds for SMBA_λ (see Alg. 5) in the presence of a computationally unbounded adversary:*

- *The expected message complexity is $O(n^2)$.*
- *The expected bit complexity is $O(n^2\lambda)$.*

Proof. As $\mathcal{CRB}$, $\mathcal{MBA}_1$, and $\mathcal{MBA}_2$ exchange $O(n^2)$ messages and $O(n^2\lambda)$ bits in expectation, the expected message complexity of SMBA_λ is $O(n^2)$ and its expected bit complexity is $O(n^2\lambda)$. □

SMBA_λ for strong consensus. As noted earlier in §4, SMBA_λ can be easily adapted to solve the strong consensus [32] problem with up to $x = |\mathcal{X}|$ predetermined proposals $\mathcal{X}$, for any $x \in O(1)$. First, our CRB_λ algorithm, when operating among $n = (x+1)t + 1$ processes with up to x predetermined values, should be modified to never send BROKEN messages (see Alg. 4). The SMBA_λ algorithm modified for strong consensus operates as follows.

1. Each correct process broadcasts its proposal using the CRB_λ algorithm.
2. Each correct process delivers a value $v \in \mathcal{X}$ from CRB_λ. Due to the justification property of the CRB_λ algorithm, v was proposed by a correct process to SMBA_λ.
3. For each $index = 1, 2, ..., x-1$, execute the following logic:
   (a) Propose v to the $index$-th instance of the MBA primitive.
   (b) Decide some value $v' \in \mathcal{X} \cup \{\bot_{\mathsf{MBA}}\}$. Now, we differentiate two cases:
   - If $v' \neq \bot_{\mathsf{MBA}}$, then decide v' and terminate. Importantly, the justification property of the MBA primitive ensures that v' was proposed by a correct process (to SMBA_λ).
   - If $v' = \bot_{\mathsf{MBA}}$, then not all correct processes proposed the same value to the $index$-th instance of the MBA primitive. Hence, wait to deliver $index + 1$ different values from CRB_λ. Once this happens (and it will due to the totality property of the CRB_λ algorithm), update v to the lexicographically smallest value delivered from CRB_λ. The justification property of the CRB_λ algorithm ensures that v was proposed by a correct process to SMBA_λ.
4. Decide the lexicographically smallest value from the $\mathcal{X}$ set. If this step is reached, it means that all values from the $\mathcal{X}$ set have been proposed by correct processes to SMBA_λ; otherwise, correct processes would have decided in one of the $x-1$ iterations.

As $|\mathcal{X}| = x \in O(1)$, the modified SMBA_λ algorithm exchanges $O(xn^2)$ messages and $O(xn^2\ell)$ bits (where ℓ denotes the bit-size of the values from the $\mathcal{X}$ set), and terminates in $O(x)$ time.

C Reducer: Proof

This section formally proves the correctness and complexity of our MVBA algorithm Reducer. Recall that Reducer's pseudocode is given in Alg. 1.

C.1 Proof of Correctness

This subsection formally proves Thm. 1. Recall that $n = 4t + 1$.

External validity. We start by proving that Reducer satisfies external validity.

Lemma 7 (Reducer satisfies external validity)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies external validity in the presence of a computationally bounded adversary.*

Proof. The external validity property is satisfied as every value quasi-decided by a correct process is valid due to the check at line 73. □

Agreement. Next, we prove Reducer's 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 75 in iteration k. We now prove that, for every iteration $k \in \mathbb{N}$, different vectors cannot be quasi-decided by correct processes.

Proposition 15. *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 proposition follows from the agreement property of the $\mathcal{MBA}[k][x]$ instance (of the MBA primitive), for every $x \in \{1, 2, 3\}$. □

We are now ready to prove that Reducer ensures agreement.

Lemma 8 (Reducer satisfies agreement)**.** *Given $n = 4t+1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies agreement in the presence of a computationally bounded adversary.*

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 Proposition 15, 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 agreement property is satisfied. □

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

Proposition 16. *Let $(k \in \mathbb{N}, x \in \{1, 2, 3\})$ be any sub-iteration and let all processes be correct. If any correct process p_i proposes a value v to $\mathcal{MBA}[k][x]$, then v is the proposal of a correct process.*

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

- Let p_i execute line 69. 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. Therefore, v is the proposal of leader(k), which proves the statement of the proposition in this case.

- Let p_i execute line 71. The statement of the proposition trivially holds in this case as v is p_i's proposal.

As the statement of the proposition holds in both cases, the proof is concluded. □

The following lemma proves that Reducer satisfies weak validity.

Lemma 9 (Reducer satisfies weak validity)**.** *Given $n = 4t+1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies weak validity in the presence of a computationally bounded adversary.*

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 Lem. 7. 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 ($k, x \in \{1, 2, 3\}$). Given that v is valid and $\bot_{\mathsf{MBA}}$ is invalid, $v \neq \bot_{\mathsf{MBA}}$. Thus, the justification property of $\mathcal{MBA}[k][x]$ guarantees that v is proposed to $\mathcal{MBA}[k][x]$ by a correct process. Proposition 16 then proves that v is the proposal of a correct process, which concludes the proof. □

Integrity. Next, we prove Reducer's integrity.

Lemma 10 (Reducer satisfies integrity)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies integrity in the presence of a computationally bounded adversary.*

Proof. The lemma trivially holds due to the check at line 75. □

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 37. The following proposition proves that at least one correct process completes the dissemination phase (and thus starts the first iteration of Reducer).

Proposition 17. *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 29) upon receiving INIT messages (line 26). As there are (at least) $n - t$ correct processes, every correct process eventually broadcasts a DONE message (line 31). Similarly, every correct process eventually receives $n - t$ DONE messages (line 32) and broadcasts a FINISH message (line 33). Thus, every correct process eventually receives a FINISH message from (at least) $n - t$ processes (line 36) and completes the dissemination phase (line 37), thus contradicting the fact that no correct process completes the dissemination phase. □

The following proposition proves that if a correct process completes the dissemination phase, then all correct processes eventually complete the dissemination phase.

Proposition 18. *If any 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 36), 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 34) and broadcasts its FINISH message (line 35). Given that there are (at least) $n - t$ correct processes, every correct process eventually receives $n - t$ FINISH messages (line 36) and completes the dissemination phase (line 37). □

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

Proposition 19. *Every correct process eventually completes the dissemination phase.*

Proof. The proposition follows directly from propositions 17 and 18. □

Recall that the specification of the SMBA primitive (see Module 3) assumes that only $O(1)$ different proposals are input by correct processes. Hence, to prove that the $\mathcal{SMBA}$ 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 $\mathcal{SMBA}$ 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 45). Let $\mathsf{suggested}_i(k)$ denote the set of digests suggested by any correct process p_i in any iteration $k \in \mathbb{N}$. The following proposition proves that each correct process suggests at most two digests in every iteration.

Proposition 20. *For every correct process p_i and every iteration $k \in \mathbb{N}$, $|\mathsf{suggested}_i(k)| \leq 2$.*

Proof. For every digest $z \in \mathsf{suggested}_i(k)$, process p_i receives (at least) $n - 3t = t + 1$ STORED messages in iteration k (due to the check at line 43). As p_i receives $n - t = 3t + 1$ STORED messages (line 42) before broadcasting its SUGGEST message, there can be at most $\frac{3t+1}{t+1} < 3$ suggested digests, which concludes the proof. □

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 $candidates_i[1] = z$ (resp., $candidates_i[2] = z$) when process p_i reaches line 55 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 $\mathsf{committed}_i(k)$ the set of digests a correct process p_i commits in an iteration $k \in \mathbb{N}$. The following proposition proves that any correct process p_i commits only digests previously suggested by p_i (in the same iteration) or the default digest *default* (line 10).

Proposition 21. *For every correct process p_i and every iteration $k \in \mathbb{N}$, the following holds:*

$$\mathsf{committed}_i(k) \subseteq (\mathsf{suggested}_i(k) \cup \{default\}).$$

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

- Let $candidates_i.\mathsf{size} = 0$ when p_i reaches line 50. This case is impossible as p_i commits only *default* in this case, i.e., $z \neq default$ is not committed.
- Let $candidates_i.\mathsf{size} = 1$ when p_i reaches line 50. As $z \notin \mathsf{suggested}_i(k)$, this case is also impossible as z is not committed by p_i.
- Let $candidates_i.\mathsf{size} = 2$ when p_i reaches line 50. Again, this case cannot occur as z cannot be committed given that $z \notin \mathsf{suggested}_i(k)$.

As neither of the three cases can occur, the proposition holds. □

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

Proposition 22. *Consider any correct process p_i and any iteration $k \in \mathbb{N}$. If process p_i commits a digest $z \neq default$, then (at least) $t + 1$ correct processes suggest z in iteration k.*

Proof. If process p_i commits $z \neq \mathit{default}$, Proposition 21 proves that $z \in \mathsf{suggested}_i(k)$. Hence, for process p_i to not remove z from its $\mathit{candidates}_i$ list after receiving $n - t = 3t + 1$ SUGGEST messages (line 46), p_i must have received at least $2t + 1$ SUGGEST messages for z (line 48). Therefore, at least $2t + 1 - t = t + 1$ correct processes suggest z in iteration k. □

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

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

The following proposition proves that $|\mathsf{committed}(k)| \in O(1)$, for any iteration k.

Proposition 23. *For every iteration $k \in \mathbb{N}$, $|\mathsf{committed}(k)| \in O(1)$.*

Proof. Proposition 22 proves that every committed digest $z \neq \mathit{default}$ is suggested by at least $t + 1$ correct processes. Moreover, Proposition 20 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, $|\mathsf{committed}(k)| < 9$, which concludes the proof. □

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

Proposition 24. *Let $(k \in \mathbb{N}, x \in \{1, 2, 3\})$ be any sub-iteration. Then, only $O(1)$ different digests are proposed to $\mathcal{SMBA}[k][x]$ by correct processes.*

Proof. If a correct process p_i proposes a digest z to $\mathcal{SMBA}[k][x]$ (line 62), then p_i commits z in iteration k (lines 57, 59 and 61). Hence, $z \in \mathsf{committed}(k)$. As $|\mathsf{committed}(k)| \in O(1)$ (by Proposition 23), the proof is concluded. □

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

Proposition 25. *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 $n - t = 3t + 1$ correct processes, all correct processes eventually receive $n - t = 3t + 1$ STORED messages (line 42) and broadcast a SUGGEST message (line 45). Therefore, all correct processes eventually receive $n - t = 3t + 1$ SUGGEST messages (line 46) and start sub-iteration $(k, 1)$ at line 55. □

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

Proposition 26. *Let $(k \in \mathbb{N}, x \in \{1, 2, 3\})$ 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), all correct processes propose to $\mathcal{SMBA}[k][x]$ (line 62). By Proposition 24, $\mathcal{SMBA}[k][x]$ behaves according to its specification (see Module 3). Thus, the termination property of $\mathcal{SMBA}[k][x]$ ensures that all correct processes eventually decide from $\mathcal{SMBA}[k][x]$. Then, all correct processes broadcast a RECONSTRUCT message (line 65), thus ensuring that every correct process eventually receives $n - t = 3t + 1$ RECONSTRUCT messages (as there are at least $n - t = 3t + 1$ correct processes). Hence, all correct processes eventually propose to $\mathcal{MBA}[k][x]$ (line 72). The termination property of $\mathcal{MBA}[k][x]$ ensures that all correct processes complete sub-iteration (k, x), thus concluding the proof. □

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

Proposition 27. *Every iteration and sub-iteration are eventually started and completed by all correct processes.*

Proof. By Proposition 19, all correct processes start iteration 1. Therefore, Proposition 25 proves that all correct processes start sub-iteration $(1, 1)$. By inductively applying propositions 25 and 26, we prove that every sub-iteration (and, thus, every iteration) is eventually started and completed by all correct processes. □

To not pollute the presentation, we might not explicitly rely on Proposition 27 in the rest of the proof. Recall that, by Def. 1, an iteration $k \in \mathbb{N}$ is good if and only if $\mathsf{leader}(k) \in \mathcal{D}_{\text{first}}$. Recall that $\mathcal{D}_{\text{first}}$ denotes the set of so-far-uncorrupted processes from which p_{first}—the first correct process that broadcasts a FINISH message at line 33—receives DONE messages before broadcasting the aforementioned FINISH message. Note that $|\mathcal{D}_{\text{first}}| \geq n - 2t = 2t + 1$. Moreover, recall that, for every good iteration k, (1) $v^\star(k)$ denotes the valid proposal of $\mathsf{leader}(k)$, and (2) $z^\star(k)$ denotes the digest of $v^\star(k)$. The proposition below proves that every correct process suggests $z^\star(k)$ in any good iteration k.

Proposition 28. *Let $k \in \mathbb{N}$ be any good iteration. Then, every correct process suggests $z^\star(k)$ in iteration k.*

Proof. As $\mathsf{leader}(k) \in \mathcal{D}_{\text{first}}$, $\mathsf{leader}(k)$ stores $z^\star(k)$ at $n - t = 3t + 1$ processes in the dissemination phase, out of which at most t can be faulty. Thus, $\mathsf{leader}(k)$ stores $z^\star(k)$ at $\geq 2t+1$ correct processes. This further implies that each correct process receives $z^\star(k)$ in STORED messages from at least $t+1$ processes, which means that each correct process broadcasts a SUGGEST message with $z^\star(k)$ (due to the check at line 43) and, thus, suggests $z^\star(k)$ in iteration k. □

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

Proposition 29. *Let $k \in \mathbb{N}$ be any good iteration. Then, every correct process commits $z^\star(k)$ in iteration k.*

Proof. By Proposition 28, all correct processes suggest $z^\star(k)$ in iteration k. Hence, for each correct process p_i, $candidates_i[1] = z^\star(k)$ or $candidates_i[2] = z^\star(k)$ when process p_i broadcasts a SUGGEST message (line 45). Furthermore, each correct process p_i receives a SUGGEST message with $z^\star(k)$ from at least $n - t - t = 2t + 1$ processes, which means that p_i does not remove $z^\star(k)$ from $candidates_i$ at line 49. Hence, the proposition holds. □

The following proposition 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.

Proposition 30. *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 Proposition 28, all correct processes suggest $z^\star(k)$ in iteration k. Thus, when process p_i reaches line 50, $candidates_i.\mathsf{size} > 0$ as $candidates_i[1] = z^\star(k)$ or $candidates_i[2] = z^\star(k)$. Hence, process p_i does not execute line 51. 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 48). □

The following proposition proves that $|\mathsf{committed}(k)| \leq 3$ in any good iteration.

Proposition 31. *Let $k \in \mathbb{N}$ be any good iteration k. Then, $|\mathsf{committed}(k)| \leq 3$.*

Proof. Proposition 29 proves that all correct processes commit $z^\star(k)$ in iteration k. Thus, $z^\star(k) \in \mathsf{committed}(k)$. To prove the proposition, we analyze the cardinality of the $\mathsf{committed}(k) \setminus \{z^\star(k)\}$ set; let $X = |\mathsf{committed}(k) \setminus \{z^\star(k)\}|$.

For every digest $z \in \mathsf{committed}(k) \setminus \{z^\star(k)\}$, Proposition 30 proves that z is suggested by (at least) $2t+1-f$ correct processes in iteration k. By Proposition 28, all correct processes suggest $z^\star(k)$ in iteration k. Given that each correct process suggests at most two digests (by Proposition 20), there are at most $n-f$ "correct suggestions" for non-$z^\star(k)$ 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-$z^\star(k)$ digests. 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. □

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

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

Observe that the following holds: (1) $\mathsf{committed}(k, 1) \subseteq \mathsf{committed}(k)$, and (2) $\mathsf{committed}(k, 2) \subseteq \mathsf{committed}(k)$. The following proposition proves that Eq. (◎) holds in any good iteration.

Proposition 32. *Let $k \in \mathbb{N}$ be any good iteration. Then, the following holds:*

$$\big(z^\star(k) \in \mathsf{committed}(k, 1) \wedge |\mathsf{committed}(k, 1)| \leq 2\big) \vee \big(\{z^\star(k)\} = \mathsf{committed}(k, 2)\big).$$

Proof. Recall that Proposition 29 proves that every correct process commits $z^\star(k)$ in iteration k. (Thus, $z^\star(k) \in \mathsf{committed}(k)$.) To prove the proposition, we consider three possible cases:

- Let $z^\star(k)$ be the lexicographically smallest digest in $\mathsf{committed}(k)$. In this case, every correct process 1-commits $z^\star(k)$ in iteration k. (Otherwise, $z^\star(k)$ could not be the smallest digest among $\mathsf{committed}(k)$.) Thus, $\mathsf{committed}(k, 1) = \{z^\star(k)\}$.
- Let $z^\star(k)$ be the lexicographically greatest digest in $\mathsf{committed}(k)$. In this case, every correct process 2-commits $z^\star(k)$ in iteration k. Hence, $\mathsf{committed}(k, 2) = \{z^\star(k)\}$.
- Let $z^\star(k)$ not be the lexicographically smallest nor the lexicographically greatest digest in $\mathsf{committed}(k)$. Hence, there exist digests z_1 and z_2 in $\mathsf{committed}(k)$ such that (1) $z_1 < z^\star(k)$, and (2) $z^\star(k) < z_2$. (Proposition 31 then shows that $\mathsf{committed}(k) = \{z_1, z^\star(k), z_2\}$.) As (1) $z_2 \in \mathsf{committed}(k)$, (2) every correct process commits $z^\star(k) < z_2$ in iteration k (by Proposition 29), and (3) each correct process commits at most two different digests, there exists a correct process that 1-commits $z^\star(k)$, i.e., $z^\star(k) \in \mathsf{committed}(k, 1)$. Moreover, since (1) every correct process commits $z^\star(k)$ in iteration k (by Proposition 29), and (2) $z^\star(k) < z_2$, no correct process 1-commits z_2 in iteration k. Therefore, $z^\star(k) \in \mathsf{committed}(k, 1)$ and $|\mathsf{committed}(k, 1)| \leq 2$, which concludes the proof in this case.

As the statement of the proposition holds in each scenario, the proof is concluded. □

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

Proposition 33. *Let $k \in \mathbb{N}$ be any good iteration. Then, all correct processes quasi-decide $v^\star(k)$ in iteration k.*

Proof. By Proposition 32, the following holds:

$$\big(z^\star(k) \in \mathsf{committed}(k,1) \wedge |\mathsf{committed}(k,1)| \leq 2\big) \vee \big(\{z^\star(k)\} = \mathsf{committed}(k,2)\big).$$

Moreover, recall that Proposition 29 proves that all correct processes commit $z^\star(k)$ in iteration k.

We distinguish two cases:

- Let $\mathsf{committed}(k,2) = \{z^\star(k)\}$. Hence, all correct processes propose $z^\star(k)$ to $\mathcal{SMBA}[k][2]$. The strong validity and termination properties of $\mathcal{SMBA}[k][2]$ ensure that all correct processes decide $z^\star(k)$ from $\mathcal{SMBA}[k][2]$. As k is a good iteration, correctly encoded RS symbols (that correspond to value $v^\star(k)$) are stored at $\geq n - 2t = 2t + 1$ correct processes. Therefore, each correct process receives RS symbols with correct witnesses for $z^\star(k)$ via RECONSTRUCT messages from at least $t+1$ processes and decodes value $v^\star(k)$ (line 69). Next, all correct processes propose $v^\star(k)$ to $\mathcal{MBA}[k][2]$, which ensures that all correct processes decide $v^\star(k)$ (due to its strong unanimity and termination properties). As $v^\star(k)$ is valid, all correct processes indeed quasi-decide $v^\star(k)$ in iteration k.
- Let $|\mathsf{committed}(k,1)| \leq 2$ and $z^\star(k) \in \mathsf{committed}(k,1)$. In this case, correct processes propose to $\mathcal{SMBA}[k][1]$ at most two different digests (as $|\mathsf{committed}(k,1)| \leq 2$) out of which one is $z^\star(k)$ (as $z^\star(k) \in \mathsf{committed}(k,1)$). The termination and strong validity properties of $\mathcal{SMBA}[k][1]$ prove that all correct processes decide from $\mathcal{SMBA}[k][1]$ a digest that belongs to $\mathsf{committed}(k,1)$. We now distinguish two possibilities:
  - Let the digest decided from $\mathcal{SMBA}[k][1]$ be $z^\star(k)$. As k is a good iteration, correctly encoded RS symbols (that correspond to value $v^\star(k)$) are stored at $\geq n - 2t = 2t + 1$ correct processes. This means that each correct process receives RS symbols with correct witnesses for $z^\star(k)$ via RECONSTRUCT messages from at least $t+1$ processes and decodes value $v^\star(k)$ (line 69). Furthermore, all correct processes propose $v^\star(k)$ to $\mathcal{MBA}[k][1]$, which ensures that all correct processes decide $v^\star(k)$ (due to its strong unanimity and termination properties). As $v^\star(k)$ is valid, all correct processes quasi-decide $v^\star(k)$ in iteration k.
  - Let the digest decided from $\mathcal{SMBA}[k][1]$ be $z \neq z^\star(k)$. Due to the strong validity property of $\mathcal{SMBA}[k][1]$ (recall that only two different digests are proposed to $\mathcal{SMBA}[k][1]$ by correct processes), $z \in \mathsf{committed}(k,1)$. Therefore, the following holds:
    * All correct processes that proposed z to $\mathcal{SMBA}[k][1]$ switch their proposal for $\mathcal{SMBA}[k][3]$: all those correct processes propose $z^\star(k)$ to $\mathcal{SMBA}[k][3]$ (line 59). (Recall that all correct processes commit $z^\star(k)$ in iteration k.)
    * All correct processes that proposed $z^\star(k)$ to $\mathcal{SMBA}[k][1]$ propose $z^\star(k)$ to $\mathcal{SMBA}[k][3]$ (as the check at line 58 does not pass).

    Therefore, all correct processes propose $z^\star(k)$ to $\mathcal{SMBA}[k][3]$. The strong validity and termination properties of $\mathcal{SMBA}[k][3]$ ensure that all correct processes decide $z^\star(k)$ from $\mathcal{SMBA}[k][3]$. As in the previous cases, correctly encoded RS symbols (that correspond to value $v^\star(k)$) are stored at $\geq n - 2t = 2t + 1$ correct processes. Therefore, each correct process receives RS symbols with correct witnesses for $z^\star(k)$ via RECONSTRUCT messages from at least $t+1$ processes and decodes value $v^\star(k)$ (line 69). This means that all correct processes propose $v^\star(k)$ to $\mathcal{MBA}[k][3]$, which ensures that all correct processes decide $v^\star(k)$ (due to its strong unanimity and termination properties). Given that $v^\star(k)$ is a valid value, all correct processes quasi-decide $v^\star(k)$.

As the statement of the proposition holds in both cases, the proof is concluded. □

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

Lemma 11 (Reducer satisfies termination)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies termination in the presence of a computationally bounded adversary. Precisely, every correct process decides within $O(1)$ iterations in expectation.*

Proof. Proposition 33 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 within $O(1)$ iterations in expectation. □

Quality. To conclude Reducer's proof of correctness, we now show that Reducer satisfies the quality property. Recall that the quality property requires that the probability of deciding an adversarial value is strictly less than 1.

Lemma 12 (Reducer satisfies quality)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, Reducer (see Alg. 1) satisfies quality in the presence of a computationally bounded adversary.*

Proof. Proposition 33 proves that all correct processes quasi-decide value $v^\star(k)$ in a good iteration k; recall that $v^\star(k)$ is a non-adversarial value. Hence, if the first iteration of Reducer is good, all correct processes quasi-decide $v^\star(1)$ in iteration 1. In that case, for Reducer to decide $v^\star(1)$, it is required that the Index() request selects $v^\star(1)$. The probability of that happening is at least $\frac{1}{3}$ as there can be at most three quasi-decided values. Thus, the probability that Reducer decides a non-adversarial value is (at least) $\frac{2t+1}{4t+1} \cdot \frac{1}{3} \approx \frac{1}{6}$. Therefore, quality is ensured as the probability that an adversarial value is decided is at most $\frac{5}{6} < 1$. □

C.2 Proof of Complexity

We now formally prove the complexity of Reducer (see Thm. 2).

Message complexity. We start by proving that Reducer sends $O(n^2)$ messages in expectation.

Lemma 13 (Reducer's expected message complexity)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, the expected message complexity of Reducer (see Alg. 1) is $O(n^2)$ in the presence of a computationally bounded adversary.*

Proof. The lemma holds as (1) the dissemination phase exchanges $O(n^2)$ messages in expectation, (2) each iteration exchanges $O(n^2)$ messages in expectation, and (3) there are $O(1)$ iterations in expectation (by Lem. 11). □

Bit complexity. Next, we prove Reducer's expected bit complexity. We start by proving that correct processes send $O(n\ell + n^2\lambda \log n)$ bits in the dissemination phase.

Proposition 34. *Correct processes 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 $n \cdot O(\frac{\ell}{n} + \log n + \lambda + \lambda \log n) \subseteq O(\ell + n\lambda \log n)$ bits via INIT messages. Moreover, process p_i sends $O(n)$ bits via ACK, DONE and FINISH messages. Therefore, process p_i sends

$$\underbrace{O(\ell + n\lambda \log n)}_{\text{INIT}} + \underbrace{O(n)}_{\text{ACK, DONE \& FINISH}}$$

$\subseteq O(\ell + n\lambda \log n)$ bits in the dissemination phase.

This implies that correct processes collectively send $O(n\ell + n^2\lambda \log n)$ bits. □

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

Proposition 35. *Correct processes send $O(n\ell + n^2\lambda \log n + n^2 \log k)$ bits in expectation in any 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 $(k, x \in \{1, 2, 3\})$. Correct processes send $O(n^2\lambda + n^2 \log k)$ bits in expectation while executing $\mathcal{SMBA}[k][x]$. Note that the $O(n^2 \log k)$ term exists as (1) each message of $\mathcal{SMBA}[k][x]$ needs to be tagged with "$k, x \in \{1, 2, 3\}$", and (2) $\mathcal{SMBA}[k][x]$ 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 $\mathcal{MBA}[k][x]$. Again, the $O(n^2 \log k)$ term exists as (1) each message of $\mathcal{MBA}[k][x]$ needs to be tagged with "$k, x \in \{1, 2, 3\}$", and (2) $\mathcal{MBA}[k][x]$ exchanges $O(n^2)$ messages in expectation. Therefore, correct processes collectively send

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

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

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

Thus, the proposition holds. □

We are ready to prove Reducer's expected bit complexity.

Lemma 14 (Reducer's expected bit complexity)**.** *Given $n = 4t + 1$ and the existence of a collision-resistant hash function, the expected bit complexity of Reducer (see Alg. 1) is*

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

in the presence of a computationally bounded adversary.

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 Lem. 11, $\mathbb{E}[K] \in O(1)$. Proposition 34 proves that all correct processes send $O(n\ell + n^2\lambda \log n)$ during the dissemination phase. Similarly, Proposition 35 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:

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

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, $\mathbb{E}[B] \in O(n\ell + n^2\lambda \log n)$. □

Time complexity. We conclude the subsection by proving Reducer's expected time complexity.

Lemma 15 (Reducer's expected time complexity)**.** *Given $n = 4t+1$ and the existence of a collision-resistant hash function, the expected time complexity of Reducer (see Alg. 1) is $O(1)$ in the presence of a computationally bounded adversary.*

Proof. Given that (1) correct processes decide within $O(1)$ iterations (by Lem. 11), (2) each iteration takes $O(1)$ time in expectation, and (3) the dissemination phase takes $O(1)$ time, Reducer indeed has $O(1)$ expected time complexity. □

D Reducer++: Proof

This section formally proves the correctness and complexity of our MVBA algorithm Reducer++. Recall that Reducer++'s pseudocode is given in Alg. 2.

D.1 Proof of Correctness

This subsection proves the correctness of Reducer++, i.e., we formally prove Thm. 3. Recall that $n = (3+\epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and $C = \lceil \frac{12}{\epsilon^2} \rceil + \lceil \frac{7}{\epsilon} \rceil$.

External validity. We start by proving Reducer++'s external validity.

Lemma 16 (Reducer++ satisfies external validity)**.** *Given $n = (3+\epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies external validity in the presence of a computationally bounded adversary.*

Proof. The property is satisfied as every value quasi-decided by a correct process is valid as ensured by the check at line 52. □

Agreement. We proceed by proving Reducer++'s 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 53. The following proposition proves that no two correct processes quasi-decide different vectors in any iteration k.

Proposition 36. *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 proposition follows from the agreement property of the $\mathcal{MBA}[k][x]$ instance (of the MBA primitive), for every $x \in [1, C]$. □

We are ready to prove Reducer++'s agreement.

Lemma 17 (Reducer++ satisfies agreement)**.** *Given $n = (3 + \epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies agreement in the presence of a computationally bounded adversary.*

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 Proposition 36, 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. □

Weak validity. First, we show that if any correct process proposes a value v to the $\mathcal{MBA}[k][x]$ instance, for any sub-iteration (k, x), and all processes are correct, then v is the proposal of a correct process.

Proposition 37. *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 $\mathcal{MBA}[k][x]$, then v is the proposal of a correct process.*

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

- Let p_i execute line 49. In this case, process p_i has received (at least) $\epsilon t + 1$ 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. Therefore, v is the proposal of leader(k), which proves the statement of the proposition in this case.
- Let p_i execute line 50. The statement of the proposition trivially holds in this case as v is p_i's proposal to Reducer++.

As the statement of the proposition holds in both cases, the proof is concluded. □

We are now ready to prove Reducer++'s weak validity.

Lemma 18 (Reducer++ satisfies weak validity)**.** *Given $n = (3 + \epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies weak validity in the presence of a computationally bounded adversary.*

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 Lem. 16. 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 $\perp_{\mathsf{MBA}}$ is invalid, $v \neq \perp_{\mathsf{MBA}}$. Thus, the justification property of $\mathcal{MBA}[k][x]$ guarantees that v was proposed to $\mathcal{MBA}[k][x]$ by a correct process. Proposition 37 then shows that v is the proposal of a correct process, which concludes the proof. □

Integrity. Next, we prove Reducer++'s integrity.

Lemma 19 (Reducer++ satisfies integrity)**.** *Given $n = (3 + \epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies integrity in the presence of a computationally bounded adversary.*

Proof. The lemma trivially holds due to the check at line 53. □

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 21. Recall that the dissemination phase of Reducer++ follows the same structure as the dissemination phase of Reducer, with the only difference being that, during encoding, values are considered as polynomials of degree ϵt. For completeness, the following proposition proves that at least one correct process completes the dissemination phase.

Proposition 38. *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 upon receiving INIT messages. As there are (at least) $n-t$ correct processes, every correct process eventually broadcasts a DONE message. Similarly, every correct process eventually receives $n - t$ DONE messages and broadcasts a FINISH message. Thus, every correct process eventually receives a FINISH message from (at least) $n-t$ processes and completes the dissemination phase, thus contradicting the fact that no correct process completes the dissemination phase. □

Next, we prove that if any correct process completes the dissemination phase, every correct process completes the dissemination phase.

Proposition 39. *If any 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 $n - t = (2 + \epsilon)t + 1$ FINISH messages, out of which (at least) $(1 + \epsilon)t + 1$ messages are sent by correct processes. Therefore, every correct process eventually receives $(1 + \epsilon)t + 1 \geq t + 1$ FINISH messages and broadcasts its FINISH message. Given that there are (at least) $n - t$ correct processes, every correct process eventually receives $n - t$ FINISH messages and completes the dissemination phase. □

We now prove that all correct processes eventually complete the dissemination phase of Reducer++.

Proposition 40. *Every correct process eventually completes the dissemination phase.*

Proof. The proposition holds due to propositions 38 and 39. □

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

Proposition 41. *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 26) and broadcast a SUGGEST message (line 29). Hence, all correct processes eventually receive $n - t = (2 + \epsilon)t + 1$ SUGGEST messages (line 30) and start sub-iteration $(k, 1)$ at line 34. □

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

Proposition 42. *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 46), thus ensuring that every correct process eventually receives $n - t = (2 + \epsilon)t + 1$ RECONSTRUCT messages (line 47). Hence, all correct processes eventually propose to $\mathcal{MBA}[k][x]$ (line 51). The termination property of $\mathcal{MBA}[k][x]$ then ensures that all correct processes complete sub-iteration (k, x). □

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

Proposition 43. *Every sub-iteration is eventually started and completed by all correct processes.*

Proof. By Proposition 40, all correct processes start iteration 1. Therefore, Proposition 41 proves that all correct processes start sub-iteration $(1, 1)$. By inductively applying propositions 41 and 42, we prove that every sub-iteration is eventually started and completed by all correct processes. □

To not pollute the presentation, we might not explicitly rely on Proposition 43 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 29). Let $\mathsf{suggested}_i(k)$ denote the set of digests suggested by any correct process p_i in any iteration $k \in \mathbb{N}$. The following proposition proves that each correct process suggests at most $\lceil \frac{3}{\epsilon} \rceil$ digests in any iteration.

Proposition 44. *For every correct process p_i and every iteration k, $|\mathsf{suggested}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$.*

Proof. For every digest $z \in \mathsf{suggested}_i(k)$, process p_i receives (at least) $n - 3t = \epsilon t + 1$ STORED messages in iteration k (line 27). As p_i receives $n - t = (2 + \epsilon)t + 1$ STORED messages (line 26) 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:

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

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

We say that a correct process *commits* a digest z in an iteration $k \in \mathbb{N}$ if and only if $z \in candidates_i$ when process p_i reaches line 34 in iteration k. Let $\mathsf{committed}_i(k)$ 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 $k \in \mathbb{N}$, $\mathsf{committed}_i(k) \subseteq \mathsf{suggested}_i(k)$.

Proposition 45. *For every correct process p_i and every iteration $k \in \mathbb{N}$, the following holds: (1) $\mathsf{committed}_i(k) \subseteq \mathsf{suggested}_i(k)$, and (2) $|\mathsf{committed}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$.*

Proof. We have that $\mathsf{committed}_i(k) \subseteq \mathsf{suggested}_i(k)$ directly from the fact that correct process p_i only removes digests from its $candidates_i$ list after broadcasting its SUGGEST message in iteration k (line 33). Moreover, as $|\mathsf{suggested}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$ (by Proposition 44), $|\mathsf{committed}_i(k)| \leq \lceil \frac{3}{\epsilon} \rceil$. □

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 $\mathsf{leader}(k) \in \mathcal{D}_{\text{first}}$. Moreover, recall that, for every good iteration k, (1) $v^\star(k)$ denotes the valid proposal of $\mathsf{leader}(k)$, and (2) $z^\star(k)$ denotes the digest of $v^\star(k)$. The following proposition proves that every correct process suggests $z^\star(k)$ in any good iteration $k \in \mathbb{N}$.

Proposition 46. *Let $k \in \mathbb{N}$ be any good iteration. Then, every correct process suggests $z^\star(k)$ in iteration k.*

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

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

Proposition 47. *Let $k \in \mathbb{N}$ be any good iteration. Then, every correct process commits $z^\star(k)$ in iteration k.*

Proof. By Proposition 46, all correct processes suggest $z^\star(k)$ in iteration k. Therefore, each correct process receives a SUGGEST message with $z^\star(k)$ from at least $n - 2t = (1+\epsilon)t+1$ processes, which means that each correct process commits $z^\star(k)$. □

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

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

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

Proposition 48. *For every good iteration $k \in \mathbb{N}$, $|\mathsf{committed}(k)| \leq C$.*

Proof. For every digest $z \in \mathsf{committed}(k)$, at least $(1+\epsilon)t+1-t = \epsilon t+1$ correct processes suggest z in iteration k. Moreover, by Proposition 44, 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 $|\mathsf{committed}(k)|$ set:

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

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, $|\mathsf{committed}(k)| < \frac{(4+\epsilon)\cdot\lceil\frac{3}{\epsilon}\rceil}{\epsilon}$. 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:

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

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

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

Proposition 49. *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 negligible probability, which concludes the proof of the proposition. □

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 $\mathsf{Noise}()$ request in sub-iteration (k, x) (line 35). 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 51). For any sub-iteration $(k \in \mathbb{N}, x \in [1, C])$, we define the $\mathsf{start}(k, x)$ set:

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

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

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

Note that the following holds:

- For every iteration $k \in \mathbb{N}$ and every $x \in [1, C]$, $\mathsf{start}(k, x) \subseteq \mathsf{committed}(k)$.
- For every iteration $k \in \mathbb{N}$ and every $x \in [1, C]$, $\mathsf{end}(k, x) \subseteq \mathsf{committed}(k)$.
- For every iteration $k \in \mathbb{N}$, $\mathsf{start}(k, 1) \subseteq \mathsf{end}(k, 1) \subseteq \mathsf{start}(k, 2) \subseteq \mathsf{end}(k, 2) \subseteq ... \subseteq \mathsf{start}(k, C) \subseteq \mathsf{end}(k, C) \subseteq \mathsf{committed}(k)$. 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 proposition proves that $z^\star(k) \in \mathsf{start}(k, x)$, for any sub-iteration (k, x) of a good iteration k.

Proposition 50. *Let $k \in \mathbb{N}$ be any good iteration. Then, for every sub-iteration $(k, x \in [1, C])$, $z^\star(k) \in \mathsf{start}(k, x)$.*

Proof. By Proposition 47, all correct processes commit $z^\star(k)$ in iteration k. Hence, the correct process that "starts" sub-iteration $(k, 1)$ (i.e., invokes the first $\mathsf{Noise}()$ request) commits $z^\star(k)$ in sub-iteration k. Thus, $z^\star(k) \in \mathsf{start}(k, 1)$. As $\mathsf{start}(k, 1) \subseteq \mathsf{start}(k, x)$, for every $x \in [2, C]$, the proposition holds. □

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 $adopted\_digest_i = z$ when process p_i reaches line 46 in sub-iteration (k, x). Next, we prove that there exists a constant probability that all correct processes that propose to $\mathcal{MBA}[k][x]$ before sub-iteration (k, x) ends propose $v^\star(k)$ given that (1) k is a good iteration, and (2) $\mathsf{start}(k, x) = \mathsf{end}(k, x)$.

Proposition 51. *Let $k \in \mathbb{N}$ be any good iteration. Let $(k, x \in [1, C])$ be any sub-iteration such that $\mathsf{start}(k, x) = \mathsf{end}(k, x)$. Then, 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 propose $v^\star(k)$.*

Proof. We prove the proposition through the following steps.

Step 1: *If a correct process p_i adopts $z^\star(k)$ in sub-iteration (k, x) and proposes v to $\mathcal{MBA}[k][x]$, then $v = v^\star(k)$.*
As k is a good iteration, correctly encoded RS symbols (that correspond to value $v^\star(k)$) 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 $z^\star(k)$ via RECONSTRUCT messages from at least $\epsilon t + 1$ processes and decodes $v^\star(k)$ (line 49). This means that process p_i indeed proposes $v^\star(k)$ to $\mathcal{MBA}[k][x]$.

Step 2: *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 adopt $z^\star(k)$ in sub-iteration (k, x).*
Recall that Proposition 47 proves that every correct process commits $z^\star(k)$ in iteration k. Hence, no correct process p_i updates its $adopted\_digest_i$ variable at line 36. Therefore, every correct process p_i updates its $adopted\_digest_i$ variable at line 44.

Let ϕ denote the output of the $\mathsf{Noise}()$ request in sub-iteration (k, x). Moreover, for every $z \in \mathsf{start}(k, x)$, let $h(z) = \mathsf{hash}(z, \phi)$. By Proposition 50, $z^\star(k) \in \mathsf{start}(k, x)$. Let $H = \{h(z) \mid z \in \mathsf{start}(k, x)\}$. To prove the statement, we show that the probability $h\big(z^\star(k)\big)$ is the lexicographically smallest hash value among H is at least $\frac{1}{C}$. Indeed, if $h\big(z^\star(k)\big)$ is the lexicographically smallest hash value among H, then all correct processes that propose to $\mathcal{MBA}[k][x]$ before sub-iteration (k, x) ends do adopt $z^\star(k)$ at line 44.

Since processes (including the faulty ones) make only polynomially many random oracle queries, the probability that the random oracle model is queried on some $(z \in \mathsf{start}(k, x), \phi)$ 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 $\frac{1}{|\mathsf{start}(k,x)|} \geq \frac{1}{C}$ as $|\mathsf{start}(k, x)| \leq |\mathsf{committed}(k)| \leq C$ (by Proposition 48). Hence, there is (at least) $\frac{1}{C}$ probability that $h\big(z^\star(k)\big)$ 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 $z^\star(k)$ 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 $v^\star(k)$. □

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

Proposition 52. *Let $k \in \mathbb{N}$ be any good iteration. Then, there exists a sub-iteration $(k, x \in [1, C])$ such that $\mathsf{start}(k, x) = \mathsf{end}(k, x)$.*

Proof. Recall that $|\mathsf{committed}(k)| \leq C$ (by Proposition 48). Moreover, Proposition 50 proves that $z^\star(k) \in \mathsf{start}(k, x)$, 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 $\mathsf{start}(k, x) \neq \mathsf{end}(k, x)$, for every sub-iteration $(k, x \in [1, C])$. As we have that $\mathsf{start}(k, x) \subseteq \mathsf{end}(k, x)$, for every sub-iteration $(k, x \in [1, C])$, we have that $\mathsf{start}(k, x) \subset \mathsf{end}(k, x)$. Moreover, $\mathsf{start}(k, x) \subset \mathsf{start}(k, x + 1)$, for every $x \in [1, C - 1]$. Thus, $|\mathsf{start}(k, C)| \geq C$, which then implies that $|\mathsf{end}(k, C)| > C$. This contradicts $|\mathsf{end}(k, C)| \leq C$, which completes the proof. □

The following proposition 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 $v^\star(k)$, then all correct processes quasi-decide $v^\star(k)$ in sub-iteration (k, x).

Proposition 53. *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 $v^\star(k)$. Then, all correct processes quasi-decide $v^\star(k)$ in sub-iteration (k, x).*

Proof. By Proposition 49, the first correct process that decides from $\mathcal{MBA}[k][x]$ does decide $v^\star(k)$. As $\mathcal{MBA}[k][x]$ ensures agreement and termination, all correct processes eventually decide $v^\star(k)$ from $\mathcal{MBA}[k][x]$, which proves the proposition. □

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

Lemma 20 (Reducer++ satisfies termination)**.** *Given $n = (3+\epsilon)t+1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies termination in the presence of a computationally bounded adversary. Precisely, every correct process decides within $O(C)$ iterations in expectation.*

Proof. Proposition 53 proves that all correct processes quasi-decide the valid value $v^\star(k)$ 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 $v^\star(k)$. Given a good iteration k and its sub-iteration $(k, x \in [1, C])$ with $\mathsf{start}(k, x) = \mathsf{end}(k, x)$, Proposition 51 proves that all correct processes that propose to $\mathcal{MBA}[k][x]$ before sub-iteration (k, x) ends do propose $v^\star(k)$ with probability (at least) $\frac{1}{C}$. Moreover, Proposition 52 proves that there exists a sub-iteration (k, x) within any good iteration k such that $\mathsf{start}(k, x) = \mathsf{end}(k, x)$. 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, $\Pr[E_k] \leq (1 - P)^k \to 0$ as $k \to \infty$. 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 $O(C)$ iterations in expectation. □

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

Lemma 21 (Reducer++ satisfies quality)**.** *Given $n = (3+\epsilon)t+1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, Reducer++ (see Alg. 2) satisfies quality in the presence of a computationally bounded adversary.*

Proof. Propositions 51 to 53 prove that all correct processes quasi-decide value $v^\star(k)$ in a good iteration k with (at least) $\frac{1}{C}$ probability; recall that $v^\star(k)$ is a non-adversarial value. Hence, if the first iteration of Reducer++ is good, all correct processes quasi-decide $v^\star(1)$ in iteration 1 with probability $\frac{1}{C}$. In that case, for Reducer++ to decide $v^\star(1)$, it is required that the Index() request selects $v^\star(1)$. 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 $1 - \frac{1}{3C^2} < 1$. □

D.2 Proof of Complexity

This subsection proves the complexity of Reducer++, i.e., it formally proves Thm. 4.

Message complexity. We start by proving Reducer++'s message complexity.

Lemma 22 (Reducer++'s expected message complexity)**.** *Given $n = (3+\epsilon)t+1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, the expected message complexity of Reducer++ (see Alg. 2) is $O(n^2C^2)$ in the presence of a computationally bounded adversary.*

Proof. The lemma holds as (1) the dissemination phase exchanges $O(n^2)$ messages in expectation, (2) each iteration exchanges $O(n^2C)$ messages in expectation, and (3) there are $O(C)$ iterations in expectation (by Lem. 20). □

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

Proposition 54. *Correct processes 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 $n \cdot O(\frac{\ell}{n} + \log n + \lambda + \lambda\log n) \subseteq O(\ell + n\lambda\log n)$ bits via INIT messages. Moreover, process p_i sends $O(n)$ bits via ACK, DONE and FINISH messages. Therefore, process p_i sends

$$\underbrace{O(\ell + n\lambda\log n)}_{\text{INIT}} + \underbrace{O(n)}_{\text{ACK, DONE \& FINISH}}$$

$\subseteq O(\ell + n\lambda\log n)$ bits in the dissemination phase.

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

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

Proposition 55. *Correct processes send $O\big(C(n\ell + n^2\lambda\log n + n^2\log k + n^2\log C)\big)$ bits in any 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 Proposition 44. 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

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

Thus, the proposition holds. □

Finally, we are ready to prove Reducer++'s bit complexity.

Lemma 23 (Reducer++'s expected bit complexity)**.** *Given $n = (3 + \epsilon)t + 1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, the expected bit complexity of Reducer++ (see Alg. 2) is*

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

in the presence of a computationally bounded adversary.

Proof. Let K be the random variable that denotes the number of iterations Reducer++ takes to terminate. By Lem. 20, $\mathbb{E}[K] \in O(C)$. Let B be the random variable that denotes the number of bits sent in Reducer++. By Proposition 54, 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 $O\big(C(n\ell + n^2\lambda \log n + n^2 \log k + n^2 \log C)\big) \subseteq O\big(C(n\ell + n^2\lambda \log n + n^2 k + n^2 \log C)\big)$ bits (by Proposition 55). Thus, we have the following:

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

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

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

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:

$$\begin{aligned}\mathbb{E}[B] &\in O\big(C^2(n\ell + n^2\lambda \log n + n^2 \log C)\big) + O(Cn^2) \cdot (O(C^2) + O(C)) \\ &\subseteq O\big(C^2(n\ell + n^2\lambda \log n + n^2 \log C)\big) + O(Cn^2) \cdot O(C^2) \\ &\subseteq O\big(C^2(n\ell + n^2\lambda \log n + n^2 \log C)\big) + O(C^3n^2) \\ &\subseteq O\big(C^2(n\ell + n^2\lambda \log n + n^2 \log C + n^2C)\big) \\ &\subseteq O\big(C^2(n\ell + n^2\lambda \log n + n^2C)\big).\end{aligned}$$

Thus, the lemma holds. □

Time complexity. Lastly, we prove Reducer++'s time complexity.

Lemma 24 (Reducer++'s expected time complexity)**.** *Given $n = (3+\epsilon)t+1$, for any fixed constant $\epsilon > 0$, and the existence of a hash function modeled as a random oracle, the expected time complexity of Reducer++ (see Alg. 2) is $O(C^2)$ in the presence of a computationally bounded adversary.*

Proof. Let K be the random variable that denotes the number of iterations Reducer++ takes to terminate. By Lem. 20, $\mathbb{E}[K] \in O(C)$. Moreover, 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:

$$\mathbb{E}[T] = \mathbb{E}\Big[\sum_{k=1}^{K} T(k)\Big] = \mathbb{E}\Big[\mathbb{E}\big[\sum_{k=1}^{K} T(k)|K\big]\Big].$$

Furthermore, we have:

$$\mathbb{E}\big[\sum_{k=1}^{K} T(k)|K\big] = \mathbb{E}[T(1)] + ... + \mathbb{E}[T(K)] = K \cdot O(C).$$

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. □