Efficient Signature-Free Validated Agreement

Pierre Civit
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Muhammad Ayaz Dzulfikar
NUS Singapore, Singapore

Seth Gilbert
NUS Singapore, Singapore

Rachid Guerraoui
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Jovan Komatovic
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Manuel Vidigueira
Ecole Polytechnique Fédérale de Lausanne (EPFL), Switzerland

Igor Zablotchi
Mysten Labs, Zürich, Switzerland

Abstract

Byzantine agreement enables n processes to agree on a common L-bit value, despite up to $t > 0$ arbitrary failures. A long line of work has been dedicated to improving the bit complexity of Byzantine agreement in synchrony. This has culminated in COOL, an error-free (deterministically secure against a computationally unbounded adversary) solution that achieves $O(nL + n^2 \log n)$ worst-case bit complexity (which is optimal for $L \geq n \log n$ according to the Dolev-Reischuk lower bound). COOL satisfies strong unanimity: if all correct processes propose the same value, only that value can be decided. Whenever correct processes do not agree *a priori* (there is no unanimity), they may decide a default value $\perp$ from COOL.

Strong unanimity is, however, not sufficient for today's state machine replication (SMR) and blockchain protocols. These systems value progress and require a decided value to always be *valid* (according to a predetermined predicate), excluding default decisions (such as $\perp$) even in cases where there is no unanimity a priori. *Validated Byzantine agreement* satisfies this property (called *external validity*). Yet, the best error-free (or even signature-free) validated agreement solutions achieve only $O(n^2 L)$ bit complexity, a far cry from the $\Omega(nL + n^2)$ Dolev-Reischuk lower bound. Is it possible to bridge this complexity gap?

We answer the question affirmatively. Namely, we present two new synchronous algorithms for validated Byzantine agreement, HashExt and ErrorFreeExt, with different trade-offs. Both algorithms are (1) signature-free, (2) optimally resilient (tolerate up to $t < n/3$ failures), and (3) early-stopping (terminate in $O(f + 1)$ rounds, where $f \leq t$ denotes the actual number of failures). On the one hand, HashExt uses only hashes and achieves $O(nL + n^3\kappa)$ bit complexity, which is optimal for $L \geq n^2\kappa$ (where κ is the size of a hash). On the other hand, ErrorFreeExt is error-free, using no cryptography whatsoever, and achieves $O\big((nL + n^2) \log n\big)$ bit complexity, which is near-optimal for any L.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Validated Byzantine agreement, Bit complexity, Round complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2024.14

Related Version The full version of this paper, which includes detailed pseudocode and proofs, is available online.
Full Version: `https://arxiv.org/abs/2403.08374v3` [20]

Funding *Pierre Civit*: Supported in part by the FNS (#40B2-0_218648).
Seth Gilbert: Supported in part by the Singapore MOE Tier 2 grant MOE-T2EP20122-0014.
Manuel Vidigueira: Supported in part by the FNS (#200021_215383).

1 Introduction

Byzantine agreement [42] is arguably the most important problem of distributed computing. It lies at the heart of state machine replication (SMR) [6, 16, 38, 1, 7, 37, 59, 48, 50] and blockchain systems [46, 13, 4, 32, 3, 25, 24]. Additionally, Byzantine agreement plays an essential role in cryptographic protocols such as multi-party computation [33, 11, 36, 10, 30, 17].

Byzantine agreement operates among n processes, out of which up to $t > 0$ can be corrupted by the adversary. A corrupted process is said to be *faulty* and can behave arbitrarily; a non-faulty process is said to be *correct* and follows the prescribed protocol. Let Value denote the set of L-bit values. (As this paper is concerned with multi-valued Byzantine agreement, we set no restrictions on the cardinality of the Value set.) During the agreement protocol, each process *proposes* exactly one value, and eventually the protocol outputs a single *decision*, as per the following interface:

- **request** $\mathsf{propose}(v \in \mathsf{Value})$: a process proposes an L-bit value v.
- **indication** $\mathsf{decide}(v' \in \mathsf{Value})$: a process decides an L-bit value v'.

Intuitively, Byzantine agreement ensures that all correct processes agree on the same *admissible* value. (We formally define the properties of Byzantine agreement in the later part of this section.)

Practical notion of value-admissibility. A critical question in designing practical Byzantine agreement algorithms is which values should be considered admissible. Traditionally, Byzantine agreement algorithms treated the proposals of correct processes as admissible. Consequently, they have focused on properties like *strong unanimity* [5, 18, 52]: if every correct process proposes the same value v, then v is the only possible decision. Notice that in such cases, if even one correct process proposes a value different from the (same) value held by all other $n-1$ processes, it is perfectly legal to decide some default "null op" value (e.g., $\perp$); it is also perfectly legal to decide a value that is "nonsense" from the perspective of the underlying application. Thus, unless all correct processes agree *a priori*, Byzantine agreement algorithms with strong unanimity are not guaranteed to make any "real" progress.

Many modern applications may require a stronger requirement: even if correct processes propose different values, the resulting decision should still adhere to some *validity* test, ensuring that the decision is not "wasted". Such a condition is usually called *external validity* [14, 41, 45, 56, 5, 61, 31, 44, 55]: any decided value must be valid according to a predetermined logical predicate. We underline that the external validity property is prevalent in today's blockchain systems. Indeed, as long as a produced block is valid (e.g., no double-spending), the block can safely be added to the chain (irrespectively of who produced it).[1]

[1] Let us underline that real-world blockchain systems might be concerned with *fairness*, thus making the question of "who produced a block" important. However, this work does not focus on fairness (or any similar topic [34, 35]).

Synchronous validated agreement. We study *validated agreement*, a variant of the Byzantine agreement problem satisfying the external validity property, in the standard synchronous setting. Formally, let $\mathsf{valid} : \mathsf{Value} \to \{\mathit{true}, \mathit{false}\}$ be any predetermined predicate. Importantly, correct processes propose valid values. The following properties are guaranteed by validated agreement:

- *Agreement:* No two correct processes decide different values.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes eventually decide.
- *Strong unanimity:* If all correct processes propose the same value v, then no correct process decides any value $v' \neq v$.
- *External validity:* If a correct process decides a value v, then $\mathsf{valid}(v) = \mathit{true}$.

We underline that validated agreement algorithms usually do not satisfy strong unanimity (but only external validity). Additionally, we emphasize that obtaining an agreement algorithm $\mathcal{A}^\star$ that satisfies *both* strong unanimity and external validity is straightforward given (1) an agreement algorithm $\mathcal{A}_1$ satisfying only strong unanimity, and (2) an agreement algorithm $\mathcal{A}_2$ satisfying only external validity. Indeed, to obtain $\mathcal{A}^\star$, processes run $\mathcal{A}_1$ and $\mathcal{A}_2$ in parallel. Then, processes decide (1) the value of $\mathcal{A}_1$ if that value is valid, or (2) the value of $\mathcal{A}_2$ otherwise.

Complexity of synchronous validated agreement. There exist two dominant worst-case complexity metrics when analyzing any synchronous validated agreement algorithm: (1) *the bit complexity*, the total number of bits correct processes send, and (2) *the round complexity*, the number of synchronous rounds it takes for all correct processes to decide (and halt). The lower bound on the bit complexity of validated agreement is $\Omega(nL + n^2)$: (1) the "nL" term comes from the fact that each correct process needs to receive the decided value, and (2) the "n^2" term comes from the seminal Dolev-Reischuk bound [27] stating that even agreeing on a single bit requires $\Omega(n^2)$ exchanged bits. We emphasize that the $\Omega(nL + n^2)$ lower bound holds even in *failure-free* executions in the signature-free world (with signatures, the bound does not hold [56]). The lower bound on the round complexity is $\Omega(f + 1)$ [28], where $f \leq t$ denotes the *actual* number of failures. If an algorithm achieves $O(f + 1)$ round complexity, it is said that the algorithm is *early-stopping.*[2]

State-of-the-art. The most efficient known validated agreement algorithm is Ada-Dare [19]. Ada-Dare achieves $O(nL + n^2\kappa)$ bit complexity (optimal for $L > n\kappa$), where κ denotes a security parameter. However, Ada-Dare internally utilizes threshold signatures [54]. (We emphasize that if $t < n/3$, some partially synchronous authenticated algorithms [60, 15] can trivially be adapted to achieve $O(nL + n^2\kappa)$ bit complexity in synchrony; Ada-Dare tolerates up to $t < n/2$ failures.) Perhaps surprisingly, the best *signature-free* validated agreement algorithms [43, 12, 22, 18] still achieve only $O(n^2L)$ bit complexity, a far cry from the $\Omega(nL + n^2)$ lower bound.

The fact that no efficient signature-free validated agreement is known becomes even more surprising when considering that optimal signature-free algorithms exist for the "traditional" Byzantine agreement problem. COOL [18] is a Byzantine agreement algorithm satisfying (only) strong unanimity while exchanging $O(nL+n^2 \log n)$ bits. Although it was not the goal of the COOL algorithm, COOL can trivially achieve early-stopping (by internally utilizing an early-stopping binary agreement such as [43]). In addition, COOL is optimally resilient

[2] We consider only *asymptotic* early-stopping (as in [43]) instead of *strict* early stopping (as in [28]) that requires termination in exactly $f + 2$ rounds.

(tolerates up to $t < n/3$ failures). Importantly, COOL uses no cryptography whatsoever: we say that COOL is *error-free* as it is deterministically secure against a computationally unbounded adversary.

Is there a fundamental complexity gap between external validity and strong unanimity in the signature-free world? Can signature-free validated agreement be solved efficiently in synchrony? These are the questions we study in this paper.

1.1 Contributions

In this paper, we present the first validated agreement algorithms achieving $o(n^2L)$ bit complexity *without* signatures:

- First, we introduce HashExt, a hash-based algorithm that exchanges $O(nL + n^3\kappa)$ bits (optimal for $L \geq n^2\kappa$), where κ denotes the size of a hash.
- Second, we provide ErrorFreeExt, an error-free (i.e., cryptography-free) solution that achieves $O\big((nL + n^2)\log n\big)$ bit complexity and is thus nearly-optimal.

Importantly, both HashExt and ErrorFreeExt are (1) optimally resilient (tolerate up to $t < n/3$ failures), and (2) early-stopping (terminate in $O(f + 1)$ synchronous rounds). A comparison of our new algorithms with the state-of-the-art can be found in Table 1.

Table 1 Performance of deterministic synchronous agreement algorithms with L-bit values and κ-bit security parameter. S stands for "strong unanimity", E stands for "external validity", and IC stands for "interactive consistency" (where processes agree on the proposals of all processes). (There exists a trivial reduction from IC to S + E, where each correct process decides the most represented valid value in the decided vector. Hence, we write that IC implies S + E.) All considered algorithms are early-stopping, except for Ada-Dare$_{ic}$ and Ada-Dare$_{su}$ (whose goal was not early-stopping).

Protocol	Validity	Bit complexity	Resilience	Cryptography
COOL [18, 43]	S	$O(nL + n^2 \log n)$	$n > 3t$	None
Parallel COOL [18, 43]	IC $\rightarrow$ (S + E)	$O(n^2L + n^3 \log n)$	$n > 3t$	None
Ada-Dare$_{ic}$ [19]	IC $\rightarrow$ (S + E)	$O(n^2L + n^2\kappa)$	$n > 2t$	Threshold Sign.
Ada-Dare$_{su}$ [19]	S + E	$O(nL + n^2\kappa)$	$n > 2t$	Threshold Sign.
HashExt	S + E	$O(nL + n^3\kappa)$	$n > 3t$	Hash
ErrorFreeExt	S + E	$O\big((nL + n^2)\log n\big)$	$n > 3t$	None
Lower bound [27, 21]	Any	$\Omega(nL + n^2)$	$t \in \Omega(n)$	Any

1.2 Overview & Technical Challenges

Why is efficient validated agreement hard? To solve the validated agreement problem (i.e., to satisfy external validity), a decided value must be valid. Therefore, a validated agreement algorithm needs to ensure that it is operating on (or converging to) a valid value. If the value (in its entirety) is attached to every message, satisfying external validity is (relatively) simple: each message can be individually validated and invalid messages can be ignored. Unfortunately, attaching an L-bit value to each message is inherently expensive, yielding a sub-optimal bit complexity of $\Omega(n^2L)$.

To avoid attaching an L-bit value to each message, the most efficient solutions to validated agreement (designed for arbitrary-sized values) involve coding techniques, where an L-bit value is split into n different shares of $O(\frac{L}{n} + \log n)$ size. The goal is to (somehow) reach agreement on a valid value using $O(n^2)$ messages of $O(\frac{L}{n} + \log n)$ bits, for a total of $O(nL + n^2 \log n)$ exchanged bits. However, this "coding-based" design introduces a new challenge. How can a process that only holds one share (or constantly many shares) know that the corresponding value is valid? For example, to check if a split value v is valid, correct processes might attempt to reconstruct it, expending $O(nL + n^2 \log n)$ bits in the process (as reconstruction is expensive). Since there may be (in the worst case) up to $t \in \Omega(n)$ invalid values (from as many faulty processes), this reconstruction process might have to be repeated many times before a valid value is found, resulting in (say) sub-optimal $O(n^2L + n^3 \log n)$ total communication.

Overview of HashExt. To overview HashExt's design, we first revisit how efficient signature-based validated agreement is solved (see, e.g., [19]). In the signature-based paradigm, efficient validated agreement algorithms adopt the following approach: (1) First, each process disseminates its value (using coding techniques) and obtains a *proof of retriveability* (PoR). A PoR is a cryptographic object containing a digest (of a value) and proving that (i) the pre-image of the digest can be retrieved by all correct processes, and (ii) the pre-image of the digest is valid. (2) Second, processes agree on a single PoR. (3) Third, processes retrieve a value corresponding to the agreed-upon PoR. Importantly, each PoR must be "self-certifying": once a correct process obtains an alleged PoR, the process must be able to determine if the PoR is valid to be sure that if this PoR gets decided in the second step, a valid value can be retrieved. That is why PoRs are usually implemented using signatures: if a PoR contains a *signature-based certificate*, processes can be confident in its validity. Due to this "self-certifying" nature of PoRs, it seems challenging to adapt them to the signature-free world.

To design a hash-based validated agreement algorithm HashExt, we (roughly) follow the aforementioned three-step approach with one fundamental difference: HashExt utilizes *implicit* ("non-self-certyfing") PoRs. Given any observed digest d, a correct process executing HashExt can determine if (1) the pre-image v of digest d can be retrieved, and (2) v is valid. There is no *proof* that the valid pre-image can be retrieved – only the protocol design ensures this guarantee.

Overview of ErrorFreeExt. To implement ErrorFreeExt, our error-free (cryptography-free) near-optimal solution, we rely on a recursive structure – carefully adapting to long values the recursive design proposed by [12, 22, 43, 51] that is only concerned with constant-sized values. At each recursive iteration with n processes, processes are statically partitioned into two halves that run the algorithm among $n/2$ processes. Moreover, each recursive iteration exhibits "additional work" through the *graded consensus* [9, 2] primitive. Intuitively, the graded consensus primitive reconciles decisions made by two distinct halves to ensure that all processes agree on a unique valid value. Due to the recursive nature of ErrorFreeExt, its bit complexity depends on the complexity of graded consensus. To obtain ErrorFreeExt's near-optimal $O\big((nL+n^2) \log n\big)$ bit complexity, we observe that a graded consensus algorithm with $O(nL + n^2 \log n)$ bits can be derived from the "reducing" technique introduced by the previously mentioned COOL [18] protocol.[3]

Roadmap. We define the system model and introduce some preliminaries in §2. We present HashExt in §3, whereas ErrorFreeExt is introduced in §4. Finally, we conclude in §5. Omitted pseudocode, detailed related work and proofs are relegated to the full version of the paper.

[3] A similar observation has recently been made for (balanced) synchronous *gradecast*, a sender-oriented counterpart to graded consensus [8].

2 System Model & Preliminaries

2.1 System Model

Processes. We consider a static set $\Pi = \{p_1, p_2, ..., p_n\}$ of n processes, where each process acts as a deterministic state machine. Our HashExt (resp., ErrorFreeExt) algorithm implements validated agreement against a computationally bounded (resp., unbounded) adversary that can corrupt up to $t < n/3$ processes at any time during an execution. (We underline that no signature-free agreement algorithm can tolerate $n/3$ or more failures [40], disregarding the restricted-resource model [29] that allows for a higher corruption threshold.) A corrupted process is said to be *faulty*; a non-faulty process is said to be *correct*. We denote by $f \leq t$ the actual number of faulty processes; we emphasize that f is not known.

Stopping. Each correct process can invoke a special stop request while executing any protocol. Once a correct process stops executing a protocol, it ceases taking any steps (e.g., sending and receiving messages).

Communication network. Processes communicate by exchanging messages over an authenticated point-to-point network. The communication network is reliable: if a correct process sends a message to a correct process, the message is eventually received.

Synchrony. We assume the standard synchronous environment in which the computation unfolds in synchronous δ-long rounds, where δ denotes the known upper bound on message delays. In each round $1, 2, ... \in \mathbb{N}$, each process (1) performs (deterministic) local computations, (2) sends (possibly different) messages to (a subset of) the other processes, and (3) receives the messages sent to it by the end of the round.

2.2 Complexity Measures

Let Agreement be any synchronous validated agreement algorithm, and let $\mathcal{E}(\mathsf{Agreement})$ denote the set of Agreement's executions. Let $\alpha \in \mathcal{E}(\mathsf{Agreement})$ be any execution. The bit complexity of α is the number of bits correct processes collectively send throughout α. The *bit complexity* of Agreement is then defined as

$$\max_{\alpha \in \mathcal{E}(\mathsf{Agreement})} \bigg\{ \text{the bit complexity of } \alpha \bigg\}.$$

Similarly, the latency complexity of α is the time it takes for all correct processes to decide and stop in α. The *latency complexity* of Agreement is then defined as

$$\max_{\alpha \in \mathcal{E}(\mathsf{Agreement})} \bigg\{ \text{the latency complexity of } \alpha \bigg\}.$$

We say that Agreement satisfies *early stopping* if and only if the latency complexity of Agreement belongs to $O\big((f+1)\delta\big)$. Note that the maximum number of rounds Agreement requires to decide – the *round complexity* of Agreement – is equal to the latency complexity of Agreement divided by δ. Throughout the paper, we use the latency and round complexity interchangeably.

2.3 Building Blocks

This subsection overviews building blocks utilized in both HashExt and ErrorFreeExt.

Reed-Solomon codes. HashExt and ErrorFreeExt rely on Reed-Solomon (RS) codes [53]. We use RSEnc and RSDec to denote RS' encoding and decoding algorithms. In brief, $\mathsf{RSEnc}(M, m, k)$ takes as input a message M consisting of k symbols, treats it as a polynomial of degree $k-1$, and outputs m evaluations of the corresponding polynomial. Similarly, $\mathsf{RSDec}(k, r, T)$ takes as input a set of symbols T (some of the symbols might be incorrect) and outputs a degree $k-1$ polynomial (i.e., k symbols) by correcting up to r errors (incorrect symbols) in T. Note that RSDec can correct up to r errors in T and output the original message given that $|T| \geq k + 2r$ [47]. Importantly, the bit-size of an RS symbol obtained by the $\mathsf{RSEnc}(M, m, k)$ algorithm is $O(\frac{|M|}{k} + \log m)$, where $|M|$ denotes the bit-size of the message M.

Graded consensus. Both HashExt and ErrorFreeExt make extensive use of the graded consensus primitive [9, 2] (also known as Adopt-Commit [26]), whose formal specification is given in Module 1. In brief, graded consensus allows processes to propose their input value from the $\mathsf{GC\_Value}$ set and decide on some value from the $\mathsf{GC\_Value}$ set with some binary grade. The graded consensus primitive ensures agreement among correct processes only if some correct process decides a value with (higher) grade 1. If no correct process decides with grade 1, graded consensus allows correct processes to disagree. (Thus, graded consensus is a weaker problem than validated agreement.) HashExt employs the graded consensus primitive on hash values ($\mathsf{GC\_Value} \equiv$ the set of all hash values). On the other hand, ErrorFreeExt utilizes graded consensus on values proposed to validated agreement ($\mathsf{GC\_Value} \equiv \mathsf{Value}$).

Module 1 Graded consensus.

Events:
- *request* $\mathsf{propose}(v \in \mathsf{GC\_Value})$: a process proposes a value $v \in \mathsf{GC\_Value}$.
- *indication* $\mathsf{decide}(v' \in \mathsf{GC\_Value}, g' \in \{0, 1\})$: a process decides a value $v' \in \mathsf{GC\_Value}$ with a grade g'.

Assumed behavior:
- Every correct process proposes exactly once.
- All correct processes propose simultaneously (i.e., in the same round). (We revisit this assumption for the graded consensus primitive employed in ErrorFreeExt; see §4.3.)

Properties:
- *Strong unanimity:* If all correct processes propose the same value v and a correct process decides a pair (v', g'), then $v' = v$ and $g' = 1$.
- *Justification:* If a correct process decides a pair $(v', \cdot)$, then v' was proposed by a correct process.
- *Consistency:* If any correct process decides a pair $(v, 1)$, then no correct process decides any pair $(v' \neq v, \cdot)$.
- *Integrity:* No correct process decides more than once.
- *Termination:* All correct processes decide simultaneously (i.e., in the same round). (The "simultaneous" termination is revisited in the graded consensus primitive employed in ErrorFreeExt; see §4.3.)

3 HashExt: Optimal Early-Stopping Hash-Based Solution

In this section, we present HashExt, our hash-based validated Byzantine agreement solution that achieves $O(nL + n^3\kappa)$ bit complexity, which is optimal for $L \geq n^2\kappa$ (κ denotes the size of a hash value). Additionally, HashExt is (1) optimally resilient as it tolerates up to $t < n/3$ faults, and (2) early-stopping as it terminates in $O\big((f+1)\delta\big)$ time (i.e., $O(f+1)$ synchronous rounds).

We start by introducing the building blocks of HashExt (§3.1). Then, we present HashExt's pseudocode (§3.2). Finally, we present a proof sketch of HashExt's correctness and complexity (§3.3). We relegate a proof of HashExt's correctness and complexity to the full version of the paper.

3.1 Building Blocks

Digests. We assume a collision-resistant function $\mathsf{digest} : \mathsf{Value} \rightarrow \mathsf{Digest} \equiv \{0,1\}^\kappa$, where κ is a security parameter. Concretely, the $\mathsf{digest}(v \in \mathsf{Value})$ function performs the following steps: (1) it encodes value v into n RS symbols $[m_1, m_2, ..., m_n] \leftarrow \mathsf{RSEnc}(v, n, t+1)$; (2) it aggregates $[m_1, m_2, ..., m_n]$ into an accumulation value z_v using the Merkle-tree-based (i.e., hash-based) cryptographic accumulator [49]; (3) it returns z_v. Note that, as we employ hash-based Merkle trees, an accumulation value z_v is a hash. The formal definition of the $\mathsf{digest}(\cdot)$ function can be found in the full version of the paper.

Data dissemination. The formal specification of the data dissemination primitive is given in Module 2. Intuitively, the data dissemination primitive allows all correct processes to obtain the same value $v^\star$ assuming that (1) all correct processes a priori agree on the digest $d^\star$ of value $v^\star$ (even if processes do not know the pre-image $v^\star$ of $d^\star$ a priori), and (2) at least one correct process initially holds the pre-image $v^\star$. We relegate the implementation of the data dissemination primitive to the full version of the paper. In brief, the implementation heavily relies on Merkle-tree-based accumulators and it exchanges $O(nL + n^2\kappa \log n)$ bits while terminating in 2δ time.

Module 2 Data dissemination.

Events:
- *request* $\mathsf{input}(v \in \mathsf{Value} \cup \{\bot\}, d \in \mathsf{Digest})$: a process inputs a value v (or $\bot$) and a digest d.
- *request* $\mathsf{output}(v' \in \mathsf{Value})$: a process outputs a value v'.

Assumed behavior:
- All correct processes input a pair. We underline that correct processes might not input their values simultaneously (i.e., at the exact same round).
- No correct process stops unless it has previously output a value.
- There exists a value $v^\star \in \mathsf{Value}$ ($v^\star \neq \bot$) and a digest $d^\star = \mathsf{digest}(v^\star)$ such that:
  - If any correct process inputs a pair $(v \in \mathsf{Value}, \cdot)$, then $v = v^\star$.
  - If any correct process inputs a pair $(\cdot, d \in \mathsf{Digest})$, then $d = d^\star$.
  - At least one correct process inputs a pair $(v^\star, d^\star)$.

Properties:
- *Safety:* If any correct process outputs a value v, then $v = v^\star$.
- *Liveness:* Let τ be the first time by which all correct processes have input a pair. Then, every correct process outputs a value by time $\tau + 2\delta$.
- *Integrity:* No correct process outputs a value unless it has previously input a pair.

3.2 Pseudocode

The pseudocode of HashExt is given in Algorithm 1.

Key idea. The crucial idea behind HashExt is to ensure that all correct processes agree on a digest $d^\star$ of a *valid* value $v^\star$ such that at least *one* correct process knows the pre-image $v^\star$ of $d^\star$. To solve validated agreement, it then suffices to utilize the data dissemination primitive (see Module 2): if (1) all correct processes input the same digest $d^\star$, and (2) at least one correct process inputs the pre-image $v^\star$ of $d^\star$, then all correct processes agree on the (valid) value $v^\star$. Given that the data dissemination primitive exchanges $O(nL + n^2\kappa \log n)$ bits and terminates in 2 rounds, HashExt dedicates $O(nL + n^3\kappa)$ bits and $O(f+1)$ rounds to agreeing on digest $d^\star$.

Protocol description. HashExt internally utilizes an instance $\mathcal{DD}$ of the data dissemination primitive. We design HashExt in a *view-based* manner: HashExt operates in (at most) $f+1$ views, where each view V has its leader $\mathsf{leader}(V) = p_V$.[4] Each view V internally uses two instances $\mathcal{GC}_1[V]$ and $\mathcal{GC}_2[V]$ of the graded consensus primitive (see Module 1) that operates on digests.

We say that a correct process p_i *commits* a digest d in view V if and only if p_i invokes $\mathcal{DD}.\mathsf{input}(\cdot, d)$ in view V (line 44). Each correct process p_i maintains four important local variables:

- $locked_i$ (line 6): holds a digest (or $\bot$) on which p_i is currently "locked on".
- $vote_i$ (line 7): holds a digest (or $\bot$) currently supported by p_i.
- $known\_values_i[D]$, for every digest D (line 9): holds the pre-image of digest D observed by p_i.
- $accepted_i[V]$, for every view V (line 10): holds the set of digest that are "accepted" in view V.

Let p_i be any correct process. Each view V operates in four steps:

1. Process p_i proposes $locked_i$ to $\mathcal{GC}_1[V]$ and decides a pair (d_1, g_1) (line 16). Intuitively, if $d_1 \neq \bot$ and $g_1 = 1$, p_i sticks with digest d_1 throughout the view as it is possible that some other correct process has previously committed digest d_1. (Hence, not sticking with digest d_1 in view V might be dangerous as it could lead to a disagreement on committed digests.)
2. Here, the leader of view V (if correct) aims to enable all correct processes to commit a digest in view V. Specifically, the leader behaves in the following manner:
   - If it decided a non-$\bot$ digest from $\mathcal{GC}_1[V]$, then the leader disseminates the digest (line 20).
   - Otherwise, the leader disseminates its proposal (line 22).

   Process p_i behaves according to the following logic:
   - If p_i decided a non-$\bot$ digest d with grade 1 from $\mathcal{GC}_1[V]$ ($d_1 = d \neq \bot$ and $g_1 = 1$; see the rule at line 23), then p_i supports digest d by broadcasting a support message for d (line 24).
   - If p_i decided $\bot$ from $\mathcal{GC}_1[V]$, then p_i supports a digest d by broadcasting a support message for d (line 27 or line 30) if (1) it receives digest d from the leader and p_i accepted d in any previous view (line 26), or (2) it receives a valid value v from the leader such that $\mathsf{digest}(v) = d$ (line 28). If the latter case applies, process p_i "observes" the pre-image v of digest d (line 29).
3. Process p_i accepts a digest d in view V if it receives a support message for d from $t+1$ processes (line 33). Moreover, process p_i updates its $vote_i$ variable to a digest d if it receives a support message for d from $2t+1$ processes (line 35). Otherwise, process p_i sets its $vote_i$ variable to $\bot$ (line 37). Observe that if any correct process p_j updates its $vote_j$ variable to a digest d, then every correct process p_k accepts d in view V. Indeed, as p_j receives a support message for digest d from at least $2t+1$ processes out of which at least $t+1$ are correct, it is guaranteed that p_k receives a support message for d from at least $t+1$ processes.

[4] HashExt elects leaders in a round-robin fashion.

Algorithm 1 HashExt: Pseudocode (for process p_i).

Uses:
  Graded consensus, **instances** $\mathcal{GC}_1[V]$, $\mathcal{GC}_2[V]$, for each view $V \in [1, t+1]$ ▷ bits: $O(n^2\kappa)$; rounds: 2
  Data dissemination, **instance** $\mathcal{DD}$ ▷ bits: $O(nL + n^2\kappa \log n)$; rounds: 2

Local variables:
  Value $v_i \leftarrow p_i$'s proposal
  Digest $locked_i \leftarrow \bot$ ▷ locked digest
  Digest $vote_i \leftarrow \bot$ ▷ digest to be voted for
  View $committed\_view_i \leftarrow \bot$
  Map(Digest → Value) $known\_values_i \leftarrow \{\bot, \bot, ..., \bot\}$ ▷ values corresponding to digests
  Map(View → Set(Digest)) $accepted_i \leftarrow \{\emptyset, \emptyset, ..., \emptyset\}$ ▷ accepted digests per view

– **Task 1** –
for each view $V \in [1, t+1]$:
  if $committed\_view_i \neq \bot$ and $commited\_view_i + 1 = V$: complete the view after 6 synchronous rounds
  if $commited\_view_i \neq \bot$ and $V > committed\_view_i + 1$: do not execute the view
  Step 1 of view V: ▷ 2 synchronous rounds
    Let $(d_1 \in \mathsf{Digest} \cup \{\bot\}, g_1 \in \{0, 1\}) \leftarrow \mathcal{GC}_1[V].\mathsf{propose}(locked_i)$
  Step 2 of view V: ▷ 2 synchronous round
    if $p_i = \mathsf{leader}(V)$:
      if $d_1 \neq \bot$: ▷ check if a non-$\bot$ digest is decided from $\mathcal{GC}_1[V]$
        broadcast d_1 ▷ broadcast a non-$\bot$ digest decided from $\mathcal{GC}_1[V]$
      else:
        broadcast v_i ▷ broadcast the proposed value
    if $d_1 \neq \bot$ and $g_1 = 1$:
      broadcast $\langle \text{SUPPORT}, d_1 \rangle$
    else:
      if $d_l \in \mathsf{Digest}$ is received from $\mathsf{leader}(V)$ and a view $V' < V$ exists with $d_l \in accepted[V']$:
        broadcast $\langle \text{SUPPORT}, d_l \rangle$
      else if $v_l \in \mathsf{Value}$ is received from $\mathsf{leader}(V)$ such that $\mathsf{valid}(v_l) = true$:
        $known\_values[\mathsf{digest}(v_l)] \leftarrow v_l$
        broadcast $\langle \text{SUPPORT}, \mathsf{digest}(v_l) \rangle$
  Step 3 of view V: ▷ 0 synchronous round (only local computation)
    if exists $d \in \mathsf{Digest}$ such that a $\langle \text{SUPPORT}, d \rangle$ message is received from $t+1$ processes:
      $accepted_i[V] \leftarrow accepted_i[V] \cup \{d\}$
    if exists $d \in \mathsf{Digest}$ such that a $\langle \text{SUPPORT}, d \rangle$ message is received from $2t+1$ processes:
      $vote_i \leftarrow d$
    else:
      $vote_i \leftarrow \bot$
  Step 4 of view V: ▷ 2 synchronous rounds
    Let $(d_2 \in \mathsf{Digest} \cup \{\bot\}, g_2 \in \{0, 1\}) \leftarrow \mathcal{GC}_2[V].\mathsf{propose}(vote_i)$
    if $d_2 \neq \bot$: ▷ check if a non-$\bot$ digest is decided from $\mathcal{GC}_2[V]$
      $locked_i \leftarrow d_2$ ▷ digest d_2 is locked as some correct process might commit it
      if $g_2 = 1$ and $committed\_view_i = \bot$: ▷ check if digest d_2 is decided with grade 1
        $committed\_view_i \leftarrow V$
        invoke $\mathcal{DD}.\mathsf{input}(known\_values[d_2], d_2)$ ▷ **commit digest** d_2

– **Task 2** – ▷ executed in a separate thread
upon $\mathcal{DD}.\mathsf{output}(v' \in \mathsf{Value})$:
  trigger $\mathsf{decide}(v')$
  wait for view $committed\_view_i + 1$ to be completed (if not yet and if $committed\_view_i + 1 \leq t+1$)

  trigger stop ▷ process p_i stops HashExt

4. Process p_i proposes $vote_i$ to $\mathcal{GC}_2[V]$ and decides a pair (d_2, g_2). If $d_2 \neq \bot$, process p_i updates its $locked_i$ variable to d_2 (line 41). Additionally, if $g_2 = 1$, then p_i commits d_2 (line 44). Importantly, if any correct process p_j commits a digest $d \neq \bot$ in view V, *every* correct process p_k updates its $locked_k$ variable to d. Indeed, as p_j commits d, it decides $(d \neq \bot, 1)$ from $\mathcal{GC}_2[V]$. The consistency property of $\mathcal{GC}_2[V]$ ensures that each correct process p_k decides d from $\mathcal{GC}_2[V]$.

We emphasize that if process p_i commits a digest in some view V, process p_i does not execute any view greater than $V + 1$ (line 14). Moreover, if p_i commits in view $V < t + 1$, then process p_i necessarily completes view $V + 1$ before stopping (line 48). Importantly, process p_i completes view $V + 1$ after *exactly* 6 rounds have elapsed. Let us elaborate. As some correct process $p_j \neq p_i$ might never enter view $V + 1$ (since it has committed in a view smaller than view V), it is possible that *not all* correct processes participate in view $V + 1$. This implies that utilized graded consensus instances might never complete, which further means that process p_i can forever be stuck executing a graded consensus instance in view $V + 1$. To avoid this scenario, process p_i completes view $V + 1$ after 6 rounds irrespectively of which step of view $V + 1$ p_i is in after 6 rounds. Finally, once p_i outputs a value v' from $\mathcal{DD}$ (and completes the aforementioned "next view"), p_i decides v' (line 47) and stops executing HashExt (line 49).

3.3 Proof Sketch

This subsection provides a proof sketch of the following theorem:

▶ **Theorem 1.** *HashExt (Algorithm 1) is a hash-based early-stopping validated agreement algorithm with $O(nL + n^3\kappa)$ bit complexity.*

Our proof sketch focuses on the crucial intermediate guarantees ensured by HashExt.

Preventing disagreement on committed digests. First, we show that correct processes do not disagree on committed digests. Let $\mathcal{V}$ denote the first view in which a correct process commits; let $d^\star$ be the committed digest. No correct process commits any non-$d^\star$ digest in view $\mathcal{V}$ due to the consistency property of $\mathcal{GC}_2[\mathcal{V}]$: it is impossible for correct processes to decide different digests from $\mathcal{GC}_2[\mathcal{V}]$ with grade 1.

If $\mathcal{V} < t + 1$, HashExt prevents any non-$d^\star$ digest to be committed in any view greater than $\mathcal{V}$. Specifically, HashExt guarantees that all correct processes commit $d^\star$ (and no other digest) by the end of view $\mathcal{V} + 1$. The consistency property of $\mathcal{GC}_2[\mathcal{V}]$ ensures that every correct process p_i updates its $locked_i$ variable to $d^\star$ at the end of view $\mathcal{V}$. Therefore, all correct processes propose $d^\star$ to $\mathcal{GC}_1[\mathcal{V} + 1]$, which implies that all correct processes decide $(d^\star, 1)$ from $\mathcal{GC}_1[\mathcal{V} + 1]$ (due to the strong unanimity property of $\mathcal{GC}_1[\mathcal{V} + 1]$). Hence, all correct processes broadcast a SUPPORT message for digest $d^\star$ (line 24), which further implies that all correct processes propose $d^\star$ to $\mathcal{GC}_2[\mathcal{V} + 1]$. Finally, the strong unanimity property of $\mathcal{GC}_2[\mathcal{V} + 1]$ ensures that all correct processes decide $(d^\star, 1)$ from $\mathcal{GC}_2[\mathcal{V} + 1]$ and thus commit $d^\star$ by the end of view $\mathcal{V} + 1$.

Ensuring eventual agreement on the committed digest. Second, we prove that an agreement on the committed digest eventually occurs. Concretely, we now show that *all* correct processes commit a digest by the end of the first view whose leader is correct. Let that view be denoted by $\mathcal{V}_l \in [1, f + 1]$ and let $p_{\mathcal{V}_l}$ be the leader of $\mathcal{V}_l$. If any correct process commits a digest in any view smaller than $\mathcal{V}_l$, then all correct processes commit the same digest by the end of view $\mathcal{V}_l$ due to the argument from the previous paragraph. Hence, suppose no correct process commits any digest in any view preceding view $\mathcal{V}_l$. We distinguish two scenarios:

- Let $p_{\mathcal{V}_l}$ decide a digest $d \neq \bot$ from $\mathcal{GC}_1[\mathcal{V}_l]$. Crucially, the justification property of $\mathcal{GC}_1[\mathcal{V}_l]$ ensures that $d \neq \bot$ is proposed by some correct process p_j. Hence, the value of the $locked_j$ variable is d at the beginning of view $\mathcal{V}_l$. Let $V' < \mathcal{V}_l$ denote the view in which p_j updates $locked_j$ to d upon deciding $d \neq \bot$ from $\mathcal{GC}_2[V']$. Again, the justification property of $\mathcal{GC}_2[V']$ guarantees that a correct process proposed d to $\mathcal{GC}_2[V']$ upon receiving $2t+1$ SUPPORT messages for d. As at least $t+1$ such messages are received from correct processes, *every* correct process accepts digest d in view V'.

  In this case, process $p_{\mathcal{V}_l}$ broadcasts digest d in Step 2. We show that all correct processes broadcast a SUPPORT message for digest d. Consider any correct process p_i. We study two possible cases:

  - Let p_i decide a non-$\bot$ digest d' with grade 1 from $\mathcal{GC}_1[\mathcal{V}_l]$. In this case, the consistency property of $\mathcal{GC}_1[\mathcal{V}_l]$ ensures that $d = d'$. Thus, process p_i sends a SUPPORT message for digest d (line 24).
  - Let p_i decide $\bot$ or with grade 0 from $\mathcal{GC}_1[\mathcal{V}_l]$. In this case, process p_i sends a SUPPORT message for digest d (line 27) as (1) it receives d from $p_{\mathcal{V}_l}$, and (2) it accepts d in view $V' < \mathcal{V}_l$.

- Let $p_{\mathcal{V}_l}$ decide $\bot$ from $\mathcal{GC}_1[\mathcal{V}_l]$. Note that this implies that no correct process decides a non-$\bot$ digest with grade 1 from $\mathcal{GC}_1[\mathcal{V}_l]$ (due to the consistency property of $\mathcal{GC}_1[\mathcal{V}_l]$). Hence, process $p_{\mathcal{V}_l}$ broadcasts its valid value v, which then implies that all correct processes send a SUPPORT message for digest $d = \mathsf{digest}(v)$ (line 30).

Hence, there exists a digest d for which all correct processes express their support in both cases. Therefore, all correct processes propose d to $\mathcal{GC}_2[\mathcal{V}_l]$. Finally, the strong unanimity property ensures that all correct processes decide $(d, 1)$ from $\mathcal{GC}_2[\mathcal{V}_l]$ and thus commit digest d in view $\mathcal{V}_l$.

Ensuring that some correct process knows the valid pre-image of the committed digest. We show how HASHEXT enables processes to "obtain" implicit PoRs (see §1). Let $d^\star$ denote the (unique) committed digest. For $d^\star$ to be committed, there exists a correct process that sends a SUPPORT message for $d^\star$ in a view in which $d^\star$ is committed (due to the justification property of $\mathcal{GC}_2[V]$, for every view V). Therefore, it suffices to show that the first correct process to ever send a SUPPORT message for $d^\star$ (or any other digest) does so at line 30 upon receiving valid value $v^\star$ with $\mathsf{digest}(v^\star) = d^\star$. Let p_i denote the first process to send a SUPPORT message for digest $d^\star$ and let it do so in some view V. We study if p_i could have sent the message at lines 24 and 27:

- Process p_i could not have sent the SUPPORT message at line 24 as this would imply that p_i is not the first correct process to send the message for $d^\star$. The justification property of $\mathcal{GC}_1[V]$ ensures that some correct process p_j has its $locked_j$ variable set to $d^\star$ at the beginning of view V. For process p_j to update its $locked_j$ variable to $d^\star$ in some view $V' < V$, there must exist a correct process that sends a SUPPORT message for $d^\star$ in view V' (due to the justification property of $\mathcal{GC}_2[V']$). Therefore, p_i cannot be the first correct process to send a SUPPORT message for $d^\star$.
- Process p_i could not have sent the SUPPORT message at line 27 as this would also imply that p_i is not the first correct process to send the message for $d^\star$. Indeed, for the message to be sent at line 27, process p_i accepts $d^\star$ in some view $V' < V$, which implies that at least one correct process sends a SUPPORT message for $d^\star$ in view V'.

Hence, p_i must have sent the message at line 30, which implies that p_i knows the pre-image $v^\star$ of digest $d^\star$ and that $v^\star$ is valid (due to the check at line 28).

Correctness. The previous three intermediate results show that the preconditions of $\mathcal{DD}$ (see Module 2) are satisfied, which implies that $\mathcal{DD}$ behaves according to its specification. Hence, all correct processes decide the same valid value from HashExt due to the properties of $\mathcal{DD}$.

Complexity. Each view with a non-correct leader exchanges $O(n^2\kappa)$ bits. Moreover, each view with a correct leader exchanges $O(nL + n^2\kappa)$ bits. As $\mathcal{DD}$ exchanges $O(nL + n^2\kappa \log n)$ bits and it is ensured that only $O(1)$ views with correct leaders are executed, HashExt exchanges $O(nL + n^2\kappa) + n \cdot O(n^2\kappa) + O(nL + n^2\kappa \log n) = O(nL + n^3\kappa)$ bits.

As all correct processes start $\mathcal{DD}$ at the end of the first view with a correct leader (at the latest), all correct processes input to $\mathcal{DD}$ in $O(f + 1)$ rounds (recall that each view has 6 rounds). Since $\mathcal{DD}$ guarantees agreement in 2 rounds, all correct decide and stop in $O(f + 1)$ rounds.

On the lack of strong unanimity. Note that HashExt as presented in Algorithm 1 does not satisfy strong unanimity. Indeed, even if all correct processes propose the same value v, it is possible that correct processes agree on a value v' proposed by a faulty leader. However, as specified in §1, it is trivial to modify HashExt to obtain an early-stopping algorithm with both strong unanimity and external validity that exchanges $O(nL + n^3\kappa)$ bits. Indeed, this can be done by running in parallel (1) the current (without strong unanimity) implementation of HashExt, and (2) the error-free early-stopping COOL [18, 43] protocol with only strong unanimity.

4 ErrorFreeExt: Near-Optimal Early-Stopping Error-Free Solution

This section presents ErrorFreeExt, an error-free validated Byzantine agreement algorithm that achieves (1) $O\big((nL + n^2) \log n\big)$ bit complexity, and (2) early stopping. Recall that ErrorFreeExt is also optimally resilient (tolerates up to $t < n/3$ Byzantine processes).

We start by introducing ErrorFreeExt's building blocks (§4.1). To introduce ErrorFreeExt's recursive structure, we first show how (a simplified version of) the recursive structure yields a near-optimal validated agreement without early-stopping – SlowExt (§4.2). Then, we overview ErrorFreeExt (§4.3) and give a proof sketch of its correctness and complexity (§4.4). We relegate ErrorFreeExt's full pseudocode and a formal proof to the full version of the paper.

4.1 Building Blocks

We now overview the building blocks of ErrorFreeExt. Given ErrorFreeExt's recursive structure, the specification of each building block explicitly states its participants (to increase the clarity). Moreover, given that building blocks might be executed among an overly corrupted set of participants (due to the recursion), each building block explicitly states what properties are ensured given the level of corruption among its participants.

Committee broadcast. The formal specification of the committee broadcast primitive is given in Module 3. Committee broadcast is concerned with two sets of processes: (1) $\mathsf{Entire} \subseteq \Pi$, and (2) $\mathsf{Committee} \subseteq \mathsf{Entire}$. Moreover, the primitive is associated with a validated Byzantine agreement algorithm $\mathcal{VA}$ to be executed among processes in $\mathsf{Committee}$. Intuitively, the committee broadcast primitive ensures the following: (1) correct processes in $\mathsf{Committee}$ agree on the same value using the $\mathcal{VA}$ algorithm (given that $\mathsf{Committee}$ is not overly corrupted), and (2) correct processes in Committee disseminate the previously agreed-upon value to all processes in Entire. We underline that the totality property of committee broadcast (deliberately written in orange in Module 3) is important only for ErrorFreeExt's early-stopping, i.e., it can be ignored for SlowExt (in §4.2).

Module 3 Committee broadcast $\langle \mathsf{Entire}, \mathsf{Committee}, \mathcal{VA} \rangle$.

Participants:
- $\mathsf{Entire} \subseteq \Pi$; let $x = |\mathsf{Entire}|$ and let x' be the greatest integer smaller than $x/3$.
- $\mathsf{Committee} \subseteq \mathsf{Entire}$; let $y = |\mathsf{Committee}|$, let y' be the greatest integer smaller than $y/3$ and let f' be the actual number of faulty processes in Committee.

Utilized validated agreement among Committee:
- $\mathcal{VA}$; let $\mathcal{L}_{\mathcal{VA}}(y, f')$ denote the worst-case latency complexity of $\mathcal{VA}$ with up to f' faulty processes and let $\mathcal{B}_{\mathcal{VA}}(y)$ denote the maximum number of bits any correct process sends while executing $\mathcal{VA}$ with up to y' faulty processes. (We underline that $\mathcal{L}_{\mathcal{VA}}(y, f')$ is based on the non-known *actual* number of failures, whereas $\mathcal{B}_{\mathcal{VA}}(y)$ is based on the known *upper bound* on the number of failures.)

Events:
- *request* $\mathsf{input}(v \in \mathsf{Value}, g \in \{0, 1\})$: a process inputs a pair (v, g).
- *indication* $\mathsf{output}(v' \in \mathsf{Value})$: a process outputs a value v'.

Assumed behavior:
- Every correct process inputs a pair.
- If a correct process inputs a pair $(v, \cdot)$, then $\mathsf{valid}(v) = \mathit{true}$.
- No correct process stops unless it has previously output a value.
- If any correct process inputs a pair $(v, 1)$, for any value v, then no correct process inputs a pair $(v' \neq v, \cdot)$.

Properties ensured only if up to x' processes in Entire are faulty:
- *Totality:* Let τ denote the first time at which a correct process outputs a value. Then, every correct process outputs a value by time $\tau + 2\delta$.
- *Stability:* If a correct process inputs a pair $(v, 1)$ and outputs a value v', then $v' = v$.
- *External validity:* If a correct process outputs a value v, then $\mathsf{valid}(v) = \mathit{true}$.
- *Optimistic consensus:* If (1) there are up to y' faulty processes in Committee, and (2) all correct processes in Entire start within 2δ time of each other, the following properties are satisfied:
  - *Liveness:* Let τ be the first time by which all correct processes in Committee have input a pair. Then, every correct process outputs a value by time $\tau + 7\delta + \mathcal{L}_{\mathcal{VA}}(y, f')$.
  - *Agreement:* No two correct processes output different values.
  - *Strong unanimity:* If every correct process proposes a pair $(v, \cdot)$, for any value v, then no correct process outputs a value different from v.

Properties ensured even if more than x' processes in Entire are faulty:
- *Complexity:* Each correct process sends $O(L + x \log x) + \mathcal{B}_{\mathcal{VA}}$ bits.

Finisher. The formal specification of the finisher primitive is given in Module 4. Finisher is executed among a set $\mathsf{Entire} \subseteq \Pi$ of processes. Each process inputs a pair $(v \in \mathsf{Value}, g \in \{0, 1\})$, where v is a value and g is a binary grade. In brief, finisher ensures that all correct processes output the same value if all correct processes input the same value with grade 1 (the liveness property). Moreover, finisher ensures totality: if any correct process outputs a value, then all correct processes output the same value. We emphasize that the finisher primitive is introduced *only* for achieving early-stopping in ErrorFreeExt, i.e., it plays no role in SlowExt.

4.2 SlowExt: Achieving Near-Optimality Without Early-Stopping

Wisdom of the ancients. As mentioned in §1.2, the problem with the sequential reconstructive approach is that, by allowing each Byzantine process to impose its own value, we can end up with $f = t \in O(n)$ (wasted) reconstructions of invalid values (with $O(n^2)$ messages each), for a total of $O(n^3)$ messages. Making an analogy to a parliamentary system (e.g., of

Module 4 Finisher $\langle\mathsf{Entire}\rangle$.

Participants:
- $\mathsf{Entire} \subseteq \Pi$; let $x = |\mathsf{Entire}|$ and let x' be the greatest integer smaller than $x/3$.

Events:
- *request* $\mathsf{input}(v \in \mathsf{Value}, g \in \{0, 1\})$: a process inputs a pair (v, g).
- *indication* $\mathsf{output}(v' \in \mathsf{Value})$: a process outputs a value v'.

Assumed behavior:
- All correct processes input a pair and they do so within 2δ time of each other.
- No correct process stops unless it has previously output a value.
- If any correct process inputs a pair $(v, 1)$, for any value v, then no correct process inputs a pair $(v' \neq v, \cdot)$.

Properties ensured only if up to x' processes in Entire are faulty:
- *Preservation:* If a correct process p_i outputs a value v', then p_i has previously input a pair $(v', \cdot)$.
- *Agreement:* No two correct processes output different values.
- *Justification:* If a correct process outputs a value, then a pair $(\cdot, 1)$ was input by a correct process.
- *Liveness:* Let all correct processes input a pair $(v, 1)$, for any value v. Let τ be the first time by which all correct processes have input. Then, all correct processes output value v by time $\tau + \delta$.
- *Totality:* Let τ be the first time at which a correct process outputs a value. Then, all correct processes output a value by time $\tau + 2\delta$.

Properties ensured even if more than x' processes in Entire are faulty:
- *Complexity:* Each correct process sends $O(x)$ bits.

some island in ancient Greece [39]), this is the equivalent of allowing every single member of parliament to present their proposal to all others. This is somewhat wasteful. In many modern parliamentary systems, since time is limited, proposals are first filtered *internally* within each party before each party presents *one* proposal to the whole assembly. Hence, no matter how many bad proposals a party might have internally, the whole assembly only discusses one per party. The cost of dealing with bad actors (and proposals) is shifted to the parties, which are individually smaller than the whole assembly. This is (essentially) the crucial realization of [12, 22]. By adopting a recursive framework with two "parties" at each level, [12, 22] obtain non-early-stopping solutions with optimal $O(n^2)$ exchanged messages (albeit still $O(n^2 L)$ exchanged bits).

SlowExt in a nutshell. To design SlowExt, we adapt the recursive framework of [12, 22] to long values. More precisely, we follow the recent variant of the framework proposed by [51, 43] that utilizes (1) the graded consensus [9, 2] primitive (instead of the "universal exchange" primitive of [12]; see Module 1), and (2) the committee broadcast primitive (see Module 3). At each recursive iteration, processes are statically partitioned into two halves (according to their identifiers) that run the algorithm among $n/2$ processes (inside that half's committee broadcast primitive) in sequential order. The recursion stops once a validated agreement instance with only a single process is reached; at this point, the process decides its proposal. A graphical depiction of SlowExt is given in the gray part of Figure 1.

Crucially, as $t < n/3$, at least one half contains less than one-third of faulty processes. Therefore, there exists a "healthy" (non-corrupted) half that successfully executes the recursive call (i.e., successfully executes the committee broadcast primitive). However, agreement achieved among a healthy half must be preserved, i.e., preventing an unhealthy half from ruining the "healthy decision" is imperative. To this end, the recursive framework utilizes the graded consensus primitive that allows the correct processes to stick with their previously made (if any) decision. For example, suppose that the first half of processes is healthy. Hence, after executing SlowExt among the first half of processes (i.e., in the first committee broadcast primitive), all correct processes obtain the same value (due to the optimistic

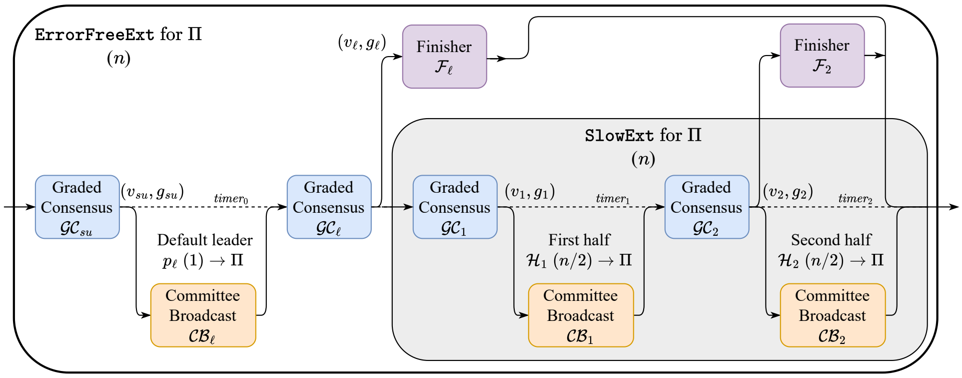

Figure 1 The recursive structure of ERRORFREEEXT (and SLOWEXT).

consensus property of committee broadcast). In this case, the graded consensus primitive $\mathcal{GC}_2$ ensures that correct processes cannot change their values due to the actions of the second half, thus preserving the previously achieved agreement. By implementing both the graded consensus and committee broadcast primitives with only $O(nL + n^2 \log n)$ bits (see the full version of the paper), SLOWEXT achieves near-optimal asymptotic bit complexity:

$$\sum_{i=0}^{\log n} 2^i \cdot \left(\frac{n}{2^i}L + \left(\frac{n}{2^i}\right)^2 \log\left(\frac{n}{2^i}\right)\right) \leq \sum_{i=0}^{\log n} \left(nL + \frac{n^2}{2^i}\log n\right) \in O\big((nL + n^2)\log n\big).$$

4.3 ErrorFreeExt: Overview

The pseudocode for ERRORFREEEXT is provided in the paper's full version and its graphical presentation can be found in Figure 1. Below, we give key insights for obtaining ERRORFREEEXT.

Why is SlowExt not early-stopping? SLOWEXT does not achieve early stopping as SLOWEXT allocates a predetermined number of rounds for each recursive call: processes cannot *prematurely* terminate a recursive call even if they have already decided. In particular, each recursive call consumes the *maximum* number of rounds necessary for its completion. This maximum number of rounds is proportional to the upper bound t on the number of Byzantine processes rather than the actual number $f \leq t$ of Byzantine processes. As a result, SLOWEXT incurs round complexity dependent on t rather than f.

From SlowExt to ErrorFreeExt. To achieve early stopping from SLOWEXT, ERRORFREEEXT mirrors the binary approach of [43] and carefully adapts it to long L-bit values. The first key ingredient is the introduction of the finisher instance $\mathcal{F}_2$ that we position (1) before the committee broadcast instance $\mathcal{CB}_2$ led by the second half of processes, and (2) after the graded consensus instance $\mathcal{GC}_2$. In brief, $\mathcal{F}_2$ leverages the presence of the graded consensus instance $\mathcal{GC}_2$ to check if $\mathcal{GC}_2$ ensured agreement among correct processes. If that is the case, then $\mathcal{F}_2$ allows correct processes to terminate immediately (i.e., in $O(\delta)$ time) after the termination of the committee broadcast instance $\mathcal{CB}_1$ led by the first half of processes.

However, the introduction of $\mathcal{F}_2$ to tackle early-stopping brings its share of technical difficulties. Indeed, since the actual number of failures f is unknown, processes cannot remain perfectly synchronized: a correct process p_i might decide (and terminate) at some time τ thinking this is the maximum time before all correct processes decide given the failures p_i observed, whereas another correct process p_j might still be running after time τ as it has observed more failures than p_i. To handle the aforementioned desynchronization, ErrorFreeExt relies on *weak synchronization* ensuring that correct processes execute different sub-modules with at most 2δ desynchronization time: if the first correct process starts executing a sub-module at time τ, then all correct processes start executing the same sub-module by time $\tau+2\delta$. To achieve this weak synchronization, we follow the standard approach of [57, 58]. Furthermore, to handle the 2δ desynchronization in ErrorFreeExt's sub-modules, we extend the round duration of graded consensus instances from the original δ time to 3δ time. (The specification of the other sub-modules directly tackles the aforementioned desynchronization.) We emphasize that at some point τ, correct processes might be in different rounds: e.g., a correct process p_i can be in round 4, whereas another correct process p_j is in round 5. However, the round duration of 3δ ensures that all correct processes *overlap* in each round for (at least) δ time. As message delays are bounded by δ, the δ-time-overlap is enough to ensure that each correct process hears all r-round-messages from all correct processes before leaving round r. (We emphasize that this is a well-known simulation technique; see, e.g., [43, 23].)

It is important to mention that ErrorFreeExt starts with a single standard "Phase King" iteration: (1) the committee broadcast instance $\mathcal{CB}_l$ with a predetermined leader p_ℓ, (2) the graded consensus instance $\mathcal{GC}_\ell$, and (3) the finisher instance $\mathcal{F}_\ell$. This iteration is added to prevent ErrorFreeExt from running for $\Theta(\log n)$ time when there are only $O(1)$ faults. Indeed, if the predetermined leader p_ℓ is correct, the committee broadcast instance $\mathcal{CB}_\ell$ ensures that all correct processes propose the same valid value v to $\mathcal{GC}_\ell$ in $O(1)$ time after starting ErrorFreeExt. Then, the strong unanimity property of $\mathcal{GC}_\ell$ ensures that all correct processes decide $(v,1)$ from $\mathcal{GC}_\ell$ and input $(v,1)$ to $\mathcal{F}_\ell$. This enables $\mathcal{F}_\ell$ to make all correct processes decide v immediately (i.e., in $O(\delta)$ time) after starting.

Finally, the graded consensus instance $\mathcal{GC}_{su}$ (together with $\mathcal{GC}_\ell$) ensures the strong unanimity property. If all correct processes propose the same value v to ErrorFreeExt, then (1) all correct processes decide $(v,1)$ from $\mathcal{GC}_{su}$ and propose v to $\mathcal{GC}_\ell$, (2) all correct processes decide $(v,1)$ from $\mathcal{GC}_\ell$ and input $(v,1)$ to $\mathcal{F}_\ell$, and (3) output v from $\mathcal{F}_\ell$ and decide v from ErrorFreeExt.

4.4 Proof Sketch

This subsection provides a proof sketch of the following theorem:

▶ **Theorem 2.** *ErrorFreeExt is an error-free early-stopping validated agreement algorithm with $O\big((nL+n^2)\log n\big)$ bit complexity.*

We underline that ErrorFreeExt achieves *balanced* bit complexity as its *per-process* complexity is $O\big((L+n)\log n\big)$. This subsection discusses the key intermediate results ensured by ErrorFreeExt.

Gluing all sub-modules together. Processes execute each sub-module within 2δ time of each other, thus enabling the associated implementations to realize the corresponding specifications. The consistency property of the graded consensus primitive ensures a similar consistency for the inputs to the following committee broadcast primitive. Under this condition, the strong unanimity property of the underlying validated agreement protocol ensures agreement if the recursive call is executed with a healthy (non-corrupted) committee.

Ensuring strong unanimity. Strong unanimity is implied by (1) the strong unanimity properties of $\mathcal{GC}_{su}$ and $\mathcal{GC}_\ell$, (2) the stability property of $\mathcal{CB}_\ell$, and (3) liveness and agreement of $\mathcal{F}_\ell$.

Finisher's "lock". If a process decides a value v via a finisher $\mathcal{F} \in \{\mathcal{F}_\ell, \mathcal{F}_2\}$, the justification property of $\mathcal{F}$, combined with the consistency property of the graded consensus $\mathcal{GC} \in \{\mathcal{GC}_\ell, \mathcal{GC}_2\}$ positioned immediately before, ensures that every correct process outputs $(v, \cdot)$ from $\mathcal{GC}$.

From a correct leader or the first healthy committee to a common valid decision. If the predetermined leader p_ℓ is correct, all correct processes agree on a common value after $\mathcal{F}_\ell$: this holds due to (1) the optimistic consensus property of $\mathcal{CB}_\ell$, (2) the strong unanimity property of $\mathcal{GC}_\ell$, and (3) the liveness and agreement properties of $\mathcal{F}_\ell$. Similarly, if p_ℓ is faulty, but the first half of processes is healthy, all correct processes agree on a common value after $\mathcal{F}_2$. Importantly, if some correct process decides via $\mathcal{F}_\ell$, the finisher's lock (see the paragraph above), combined with strong unanimity of $\mathcal{GC}_1$ and $\mathcal{GC}_2$ and the stability property of $\mathcal{CB}_1$, guarantees agreement.

From the second healthy committee to a common valid decision. If a correct process does not decide via $\mathcal{F}_\ell$ or $\mathcal{F}_2$, it means that both the predetermined leader p_ℓ and the first half of processes are unhealthy, which implies that the second half is healthy. If some correct process decides via $\mathcal{F}_2$, the finisher's lock, combined with $\mathcal{CB}_2$'s strong unanimity, preserves agreement. Let us emphasize that if some correct process decides via $\mathcal{F}_\ell$, the agreement is ensured due to (1) the finisher's lock, (2) the strong unanimity properties of $\mathcal{GC}_1$ and $\mathcal{GC}_2$, and (3) the stability property of $\mathcal{CB}_1$.

Complexity. The per-process bit complexity $\mathcal{B}(n)$ of ErrorFreeExt follows from the equation $\mathcal{B}(n) \leq O(L + n \log n) + \max\big(\mathcal{B}(\lfloor \frac{n}{2} \rfloor), \mathcal{B}(\lceil \frac{n}{2} \rceil)\big)$. Similarly, the early stopping property holds due to the following equations: (1) $\mathcal{L}(n, f) \in O(\delta)$ if the predetermined leader p_ℓ is correct, and (2) $\mathcal{L}(n, f) \leq O(\delta) + \mathcal{L}(|\mathcal{H}_1|, f_1) + \mathcal{L}(|\mathcal{H}_2|, f_2)$ otherwise, where f_1 (resp., f_2) denotes the actual number of faulty processes among the first (resp., second) half of processes $\mathcal{H}_1$ (resp., $\mathcal{H}_2$).

5 Concluding Remarks

This paper introduces HashExt and ErrorFreeExt, two synchronous signature-free algorithms for validated Byzantine agreement. Both algorithms are (1) optimally resilient, and (2) early stopping. On one side, HashExt utilizes only collision-resistant hashes, achieving a bit complexity of $O(nL + n^3\kappa)$, which is optimal when $L \geq n^2\kappa$ (with κ being the size of a hash value). Conversely, ErrorFreeExt is error-free, avoids cryptography entirely, and achieves a bit complexity of $O\big((nL + n^2)\log n\big)$, which is nearly optimal for any L. In the future, we plan to focus on the following open questions:

- Is it possible to design an error-free validated agreement algorithm with a bit complexity of $O(nL)$? Our ErrorFreeExt algorithm achieves only $O(nL \log n)$ bit complexity.
- Can HashExt be optimized to achieve $O(nL)$ bit complexity for a wider range of proposal sizes L? Currently, HashExt allows for optimal $O(nL)$ bit complexity only when $L \geq n^2\kappa$.

References

1 Michael Abd-El-Malek, Gregory R Ganger, Garth R Goodson, Michael K Reiter, and Jay J Wylie. Fault-Scalable Byzantine Fault-Tolerant Services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74, 2005. doi:10.1145/1095810.1095817.

2 Ittai Abraham and Gilad Asharov. Gradecast in synchrony and reliable broadcast in asynchrony with optimal resilience, efficiency, and unconditional security. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 392–398. ACM, 2022. doi:10.1145/3519270.3538451.

3 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solidus: An Incentive-compatible Cryptocurrency Based on Permissionless Byzantine Consensus. *CoRR, abs/1612.02916*, 2016.

4 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, *21st International Conference on Principles of Distributed Systems, OPODIS 2017, Lisbon, Portugal, December 18-20, 2017*, volume 95 of *LIPIcs*, pages 25:1–25:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.OPODIS.2017.25.

5 Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 337–346. ACM, 2019. doi:10.1145/3293611.3331612.

6 Atul Adya, William J. Bolosky, Miguel Castro, Gerald Cermak, Ronnie Chaiken, John R. Douceur, Jon Howell, Jacob R. Lorch, Marvin Theimer, and Roger Wattenhofer. FARSITE: federated, available, and reliable storage for an incompletely trusted environment. In David E. Culler and Peter Druschel, editors, *5th Symposium on Operating System Design and Implementation (OSDI 2002), Boston, Massachusetts, USA, December 9-11, 2002*. USENIX Association, 2002. URL: http://www.usenix.org/events/osdi02/tech/adya.html.

7 Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Steward: Scaling byzantine fault-tolerant replication to wide area networks. *IEEE Trans. Dependable Secur. Comput.*, 7(1):80–93, 2010. doi:10.1109/TDSC.2008.53.

8 Gilad Asharov and Anirudh Chandramouli. Perfect (parallel) broadcast in constant expected rounds via statistical VSS. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology - EUROCRYPT 2024 - 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26-30, 2024, Proceedings, Part V*, volume 14655 of *Lecture Notes in Computer Science*, pages 310–339. Springer, 2024. doi:10.1007/978-3-031-58740-5_11.

9 Hagit Attiya and Jennifer L. Welch. Multi-valued connected consensus: A new perspective on crusader agreement and adopt-commit. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, *27th International Conference on Principles of Distributed Systems, OPODIS 2023, December 6-8, 2023, Tokyo, Japan*, volume 286 of *LIPIcs*, pages 6:1–6:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.OPODIS.2023.6.

10 Zuzana Beerliova-Trubiniova and Martin Hirt. Simple and efficient perfectly-secure asynchronous MPC. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4833 LNCS:376–392, 2007. doi:10.1007/978-3-540-76900-2_23.

11 Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988. doi:10.1145/62212.62213.

12 Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit Optimal Distributed Consensus. In *Computer science: research and applications*, pages 313–321. Springer, 1992.

13 Ethan Buchman. *Tendermint: Byzantine Fault Tolerance in the Age of Blockchains*. PhD thesis, University of Guelph, 2016. URL: https://atrium.lib.uoguelph.ca/server/api/core/bitstreams/0816af2c-5fd4-4d99-86d6-ced4eef2fb52/content.

14 Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and Efficient Asynchronous Broadcast Protocols. In Joe Kilian, editor, *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, volume 2139 of *Lecture Notes in Computer Science*, pages 524–541. Springer, 2001. doi:10.1007/3-540-44647-8_31.

15 Jan Camenisch, Manu Drijvers, Timo Hanke, Yvonne-Anne Pignolet, Victor Shoup, and Dominic Williams. Internet computer consensus. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 81–91. ACM, 2022. doi:10.1145/3519270.3538430.

16 Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems*, 20(4), 2002. doi:10.1145/571637.571640.

17 Nishanth Chandran, Wutichai Chongchitmate, Juan A. Garay, Shafi Goldwasser, Rafail Ostrovsky, and Vassilis Zikas. The hidden graph model: Communication locality and optimal resiliency with adaptive faults. In *ITCS 2015 - Proceedings of the 6th Innovations in Theoretical Computer Science*, pages 153–162, 2015. doi:10.1145/2688073.2688102.

18 Jinyuan Chen. Optimal error-free multi-valued byzantine agreement. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 17:1–17:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.17.

19 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. DARE to agree: Byzantine agreement with optimal resilience and adaptive communication. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 145–156. ACM, 2024. doi:10.1145/3662158.3662792.

20 Pierre Civit, Muhammad Ayaz Dzulfikar, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Manuel Vidigueira, and Igor Zablotchi. Error-free near-optimal validated agreement. *CoRR*, abs/2403.08374, 2024. doi:10.48550/arXiv.2403.08374.

21 Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, Anton Paramonov, and Manuel Vidigueira. All byzantine agreement problems are expensive. In Ran Gelles, Dennis Olivetti, and Petr Kuznetsov, editors, *Proceedings of the 43rd ACM Symposium on Principles of Distributed Computing, PODC 2024, Nantes, France, June 17-21, 2024*, pages 157–169. ACM, 2024. doi:10.1145/3662158.3662780.

22 Brian A. Coan and Jennifer L. Welch. Modular Construction of a Byzantine Agreement Protocol with Optimal Message Bit Complexity. *Inf. Comput.*, 97(1):61–85, 1992. doi:10.1016/0890-5401(92)90004-Y.

23 Shir Cohen, Idit Keidar, and Alexander Spiegelman. Make every word count: Adaptive byzantine agreement with fewer words. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium*, volume 253 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.OPODIS.2022.18.

24 Miguel Correia. From Byzantine Consensus to Blockchain Consensus. In *Essentials of Blockchain Technology*, pages 41–80. Chapman and Hall/CRC, 2019.

25 Tyler Crain, Vincent Gramoli, Mikel Larrea, and Michel Raynal. DBFT: efficient leaderless byzantine consensus and its application to blockchains. In *17th IEEE International Symposium on Network Computing and Applications, NCA 2018, Cambridge, MA, USA, November 1-3, 2018*, pages 1–8. IEEE, 2018. doi:10.1109/NCA.2018.8548057.

26 Carole Delporte-Gallet, Hugues Fauconnier, and Michel Raynal. On the weakest information on failures to solve mutual exclusion and consensus in asynchronous crash-prone read/write systems. *J. Parallel Distributed Comput.*, 153:110–118, 2021. doi:10.1016/J.JPDC.2021.03.015.

27 Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985. doi:10.1145/2455.214112.

28 Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990. doi:10.1145/96559.96565.

29 Juan Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-Restricted Cryptography: Revisiting MPC Bounds in the Proof-of-Work Era. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 12106 LNCS:129–158, 2020. doi:10.1007/978-3-030-45724-2_5.

30 Sanjam Garg, Aarushi Goel, and Abhishek Jain. The broadcast message complexity of secure multiparty computation. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology - ASIACRYPT 2019 - 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8-12, 2019, Proceedings, Part I*, volume 11921 of *Lecture Notes in Computer Science*, pages 426–455. Springer, 2019. doi:10.1007/978-3-030-34578-5_16.

31 Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022. doi:10.1007/978-3-031-18283-9_14.

32 Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 51–68, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3132747.3132757.

33 Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred V. Aho, editor, *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 218–229. ACM, 1987. doi:10.1145/28395.28420.

34 Vincent Gramoli, Zhenliang Lu, Qiang Tang, and Pouriya Zarbafian. Optimal asynchronous byzantine consensus with fair separability. *IACR Cryptol. ePrint Arch.*, page 545, 2024. URL: https://eprint.iacr.org/2024/545.

35 Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 451–480. Springer, 2020. doi:10.1007/978-3-030-56877-1_16.

36 Hannah Keller, Claudio Orlandi, Anat Paskin-Cherniavsky, and Divya Ravi. MPC with Low Bottleneck-Complexity: Information-Theoretic Security and More. In *4th Conference on Information-Theoretic Cryptography (ITC)*, volume 267, pages 1–21, Aarhus, Denmark, 2023. doi:10.4230/LIPIcs.ITC.2023.11.

37 Ramakrishna Kotla, Lorenzo Alvisi, Michael Dahlin, Allen Clement, and Edmund L. Wong. Zyzzyva: speculative byzantine fault tolerance. In Thomas C. Bressoud and M. Frans Kaashoek, editors, *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 45–58. ACM, 2007. doi:10.1145/1294261.1294267.

38 Ramakrishna Kotla and Michael Dahlin. High throughput byzantine fault tolerance. In *2004 International Conference on Dependable Systems and Networks (DSN 2004), 28 June*

- *1 July 2004, Florence, Italy, Proceedings*, pages 575–584. IEEE Computer Society, 2004. doi:10.1109/DSN.2004.1311928.

39 Leslie Lamport. Paxos Made Simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

40 Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

41 Leslie Lamport, Robert Shostak, and Marshall Pease. Concurrency: The works of leslie lamport. *Association for Computing Machinery*, pages 203–226, 2019.

42 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

43 Christoph Lenzen and Sahar Sheikholeslami. A recursive early-stopping phase king protocol. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 60–69. ACM, 2022. doi:10.1145/3519270.3538425.

44 Yuan Lu, Zhenliang Lu, and Qiang Tang. Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined BFT. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 2159–2173. ACM, 2022. doi:10.1145/3548606.3559346.

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

46 Loi Luu, Viswesh Narayanan, Kunal Baweja, Chaodong Zheng, Seth Gilbert, and Prateek Saxena. SCP: A Computationally-Scalable Byzantine Consensus Protocol For Blockchains. *Cryptology ePrint Archive*, 2015.

47 Florence Jessie MacWilliams and Neil James Alexander Sloane. *The Theory of Error-Correcting Codes*, volume 16. Elsevier, 1977.

48 Dahlia Malkhi, Kartik Nayak, and Ling Ren. Flexible byzantine fault tolerance. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 1041–1053. ACM, 2019. doi:10.1145/3319535.3354225.

49 Ralph C. Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology - CRYPTO '87, A Conference on the Theory and Applications of Cryptographic Techniques, Santa Barbara, California, USA, August 16-20, 1987, Proceedings*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer, 1987. doi:10.1007/3-540-48184-2_32.

50 Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In Yongdae Kim, Jong Kim, Giovanni Vigna, and Elaine Shi, editors, *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 1686–1699. ACM, 2021. doi:10.1145/3460120.3484554.

51 Atsuki Momose and Ling Ren. Optimal Communication Complexity of Authenticated Byzantine Agreement. In Seth Gilbert, editor, *35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference)*, volume 209 of *LIPIcs*, pages 32:1–32:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPIcs.DISC.2021.32.

52 Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. In Hagit Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference*,

volume 179 of *LIPIcs*, pages 28:1–28:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.DISC.2020.28.

53 Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

54 Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000. doi:10.1007/3-540-45539-6_15.

55 Anping Song and Cenhao Zhou. Flexbft: A flexible and effective optimistic asynchronous bft protocol. *Applied Sciences*, 14(4):1461, 2024.

56 Alexander Spiegelman. In search for an optimal authenticated byzantine agreement. *arXiv preprint arXiv:2002.06993*, 2020.

57 T. K. Srikanth and Sam Toueg. Optimal clock synchronization. In Michael A. Malcolm and H. Raymond Strong, editors, *Proceedings of the Fourth Annual ACM Symposium on Principles of Distributed Computing, Minaki, Ontario, Canada, August 5-7, 1985*, pages 71–86. ACM, 1985. doi:10.1145/323596.323603.

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

59 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.

60 Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. Dispersedledger: High-throughput byzantine consensus on variable bandwidth networks. In Amar Phanishayee and Vyas Sekar, editors, *19th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2022, Renton, WA, USA, April 4-6, 2022*, pages 493–512. USENIX Association, 2022. URL: https://www.usenix.org/conference/nsdi22/presentation/yang.

61 You Zhou, Zongyang Zhang, Haibin Zhang, Sisi Duan, Bin Hu, Licheng Wang, and Jianwei Liu. Dory: Asynchronous BFT with reduced communication and improved efficiency. *IACR Cryptol. ePrint Arch.*, page 1709, 2022. URL: https://eprint.iacr.org/2022/1709.