Deterministic Byzantine Agreement with Adaptive $O(n \cdot f)$ Communication

Fatima Elsheimy*
Yale University

Giorgos Tsimos†
University of Maryland

Charalampos Papamanthou‡
Yale University

Abstract

We present a deterministic synchronous protocol for binary Byzantine Agreement against a corrupt minority with adaptive $O(n \cdot f)$ communication complexity, where f is the exact number of corruptions. Our protocol improves the previous best-known deterministic Byzantine Agreement protocol developed by Momose and Ren (DISC 2021), whose communication complexity is quadratic, independent of the exact number of corruptions. Our approach combines two distinct primitives that we introduce and implement with $O(n \cdot f)$ communication, *Reliable Voting* and *Weak Byzantine Agreement*. In Reliable Voting, all honest parties agree on the same value only if all honest parties start with that value, but there is no agreement guarantee in the general case. In Weak Byzantine Agreement we achieve agreement, but validity requires that the inputs to the protocol satisfy certain properties. Our Weak Byzantine Agreement protocol is an adaptation of the recent Cohen et al. protocol (OPODIS 2022), in which we identify and address various issues.

1 Introduction

Byzantine Agreement (BA) is a fundamental problem in the domain of distributed computing. In the simplest (binary) version of the BA problem, n parties start with some value in $\{0, 1\}$ and wish to jointly agree on one value while tolerating up to $t < n/2$ Byzantine (i.e., malicious) parties (Agreement.) If all honest parties start with the same value, they must output that value (Validity.) See Definition 2.1 for the formal definition of BA. The foundations of this field were laid by the pioneering work of Lamport, Shostak, and Pease [27] in the 1980s. However, in recent years, we have witnessed the practical applications of BA in large-scale systems, mainly due to recent advancements in blockchain protocols.

1.1 Our Result. This paper focuses on the *communication complexity* of deterministic binary BA. In particular, we present the first protocol to achieve deterministic binary BA[1] with adaptive $O(n \cdot f)$ communication complexity[2] and near-optimal resilience $t < (\frac{1}{2} - \epsilon)n$, where $f \leq t$ is the *exact* number of faulty parties. This constitutes a major improvement over the quadratic $O(n^2)$ communication bound achieved by previous work [29] (For example, for executions where f is constant, our work achieves a linear communication bound.) It should be noted that our protocol does not require knowledge of the exact value of f. For a comparison of our protocol with existing work, please see Table 1.

Reliable Voting and Weak BA. To achieve deterministic BA with $O(n \cdot f)$ communication, we reduce deterministic BA into two distinct problems, *Reliable Voting* (RV) and *Weak Byzantine Agreement* (WBA). Then we provide protocols with $O(n \cdot f)$ communication for RV and WBA. Note that both RV and WBA are problem definitions that we introduce. (In particular, our WBA definition is an appropriate extension of a WBA definition appearing in previous work [15].)

Intuitively, in RV, every party starts with some value in $\{0, 1\}$ and upon termination, every party outputs a value v alongside some auxiliary information aux. What we wish RV to achieve is provable validity with respect to a fixed Boolean predicate $\mathsf{validate}$: If all honest parties have the same input value v, then they all output v

*fatima.elsheimy@yale.edu
†tsimos@umd.edu (Work partially done when the author was visiting Yale.)
‡charalampos.papamanthou@yale.edu

[1] Note that for the binary case, BA and strong BA where parties must agree *always* on the initial input of some honest party, are exactly the same. This is not trivially the case in the multivalued setting.

[2] As in related work, communication complexity refers to the number of words, where a word is a constant-size object.

Table 1: Comparison of existing literature with our work. We measure communication of all protocols in *words*, where a word comprises a constant number of signatures and values.

Protocol	Corruptions	Communication	Problem
Dolev and Strong [20]	$t < n$	$O(n^2 \cdot t)$	Broadcast
Momose and Ren [29]	$t < n/2$	$O(n^2)$	Agreement
Cohen et al. [15]	$t < n/2$	$O(n \cdot f)$	Broadcast
Our Π_{BA} (Figure 1)	$t < (1/2 - \epsilon)n$	$O(n \cdot f)$	Agreement

along with aux such that $(\mathsf{v}, \mathsf{aux})$ satisfy $\mathsf{validate}$. At the same time we must ensure that no malicious party can output a different value $\mathsf{v}' \neq \mathsf{v}$ and some auxiliary information aux' that satisfy $\mathsf{validate}$. See Definition 2.2 for the formal definition of RV. In our RV construction, aux represents a proof that there exist $n - t$ distinct digital signatures on v, r (for some $r = 1, \ldots, n$), in which case we call the respective Boolean predicate a "threshold predicate"[3]. A keen reader might have already noticed that RV achieves the validity property of BA, however, it does not achieve agreement (Nothing is guaranteed for the case where honest parties start with different values.)

For this, we make use of WBA. In WBA, each party now starts with some input value v along with some auxiliary information aux and upon termination outputs a decision value. The protocol must satisfy *agreement* and *restricted validity*, again with respect to a fixed Boolean predicate $\mathsf{validate}$. Agreement is the same as for BA with respect to the output values. Restricted validity requires that if all honest parties start with the same input value v and auxiliary information aux such that $(\mathsf{v}, \mathsf{aux})$ satisfy $\mathsf{validate}$, and no party starts with a different value $\mathsf{v}' \neq \mathsf{v}$ and some auxiliary information aux' that satisfy $\mathsf{validate}$, then all honest parties output v. See Definition 2.3 for the formal definition of WBA. Again, the $\mathsf{validate}$ predicate in our WBA construction will be the threshold predicate.

BA from RV and WBA. One can observe the input requirements for WBA's restricted validity are satisfied by the output requirements guaranteed by RV's provable validity, assuming they are both using the same predicate. Clearly, this is not a coincidence. What our BA construction does is taking the input values of BA, runs them through an RV protocol that enhances them with aux that satisfies the threshold predicate, whenever all honest parties start with the same input. Then it runs the output pair (value and aux) through a WBA protocol to guarantee agreement. The precise pseudocode of our proposed BA is in Figure 1.

Algorithm $\Pi_{\mathsf{BA}}(\mathsf{v}_i)$:
1: $\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i \leftarrow \Pi_{\mathsf{RV}}(\mathsf{v}_i)$
2: $\mathsf{y}_{\mathsf{i}} \leftarrow \Pi_{\mathsf{WBA}}(\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i)$
return y_{i}

Figure 1: Code of Π_{BA} for party p_i.

We now state and prove our first theorem.

Theorem 1.1. (Byzantine Agreement from Reliable Voting and Weak Byzantine Agreement) *Let Π_{RV} be an RV protocol, as per Definition 2.2, with respect to a predicate* $\mathsf{validate}$. *Let Π_{WBA} be a WBA protocol, as per Definition 2.3, with respect to the same predicate* $\mathsf{validate}$. *Then Π_{BA} from Figure 1 is a BA protocol, as per Definition 2.1.*

Proof. For validity, assume each honest party p_i starts with value $\mathsf{v}_i = \mathsf{v}$. The execution of $\Pi_{\mathsf{RV}}(\mathsf{v}_i)$ ensures that all honest parties output $(\mathsf{v}, \mathsf{aux}_i)$ such that $\mathsf{validate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$. At the same time, no party generates an output of $(\mathsf{v}' \neq \mathsf{v}, \mathsf{aux}_i)$ where $\mathsf{validate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$, as guaranteed by RV. In Line 2, all honest parties invoke Π_{WBA} with the output $(\mathsf{y}_{\mathsf{rv}}, \mathsf{aux}_i)$ of Π_{RV}. Thus, based on the restricted validity property of WBA, all honest

[3] An implementation of this predicate could simply be a verification of $n - t$ distinct signatures, although in our construction we will be using other implementations of this predicate, e.g., through verification of succinct arguments.

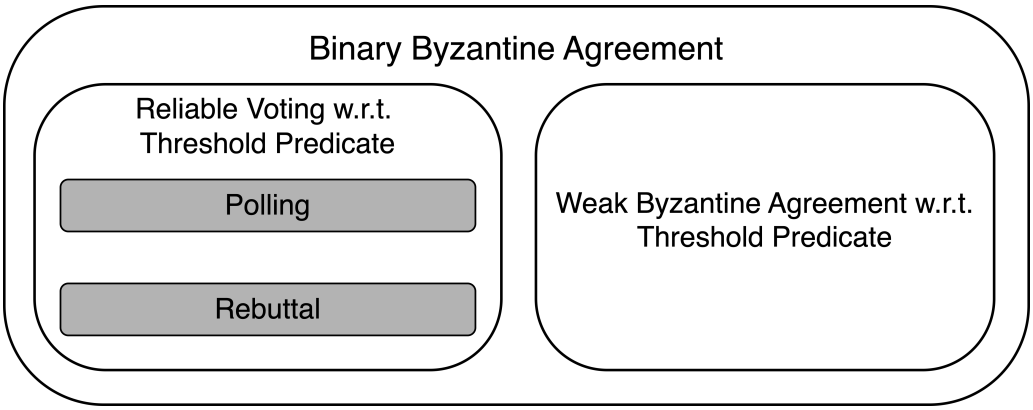

Figure 2: Our implementation of Byzantine Agreement: We derive Byzantine Agreement from Reliable Voting and Weak Byzantine Agreement, both w.r.t. to the Threshold Predicate. Reliable Voting is implemented via two subprotocols: Polling and Rebuttal.

parties decide v. For agreement, Π_{WBA} achieves agreement independently of the input with which it is initialized. Since Π_{WBA} is the last call of every honest party, the agreement property of Π_{BA} is guaranteed from the agreement property of Π_{WBA}. Finally, termination follows trivially from the termination of Π_{RV} and Π_{WBA}. □

Communication complexity and the use of silent phases. What is the communication complexity of the above protocol? For one thing, naive implementations of RV or WBA easily lead to $O(n^2)$ communication complexity, far from our stated goal of $O(n \cdot f)$. For example, one can implement RV by having each party sign its input and send its signature along with its input to all other parties, which will lead to quadratic communication complexity. As mentioned before, to achieve our stated goal of $O(n \cdot f)$, it is essential for both RV and WBA to maintain a communication complexity of $O(n \cdot f)$. This is the focus of our paper.

Towards this goal, both our proposals for RV and WBA follow the paradigm of *silent phases*, introduced by Spiegelman [32]. In this paradigm, protocols execute over n round-robin phases, where each party is the leader of its respective phase. Communication takes place only (1) from leader to parties and (2) from parties to leader. In our work we also require that communication is restricted to a fixed connectivity structure—expander graphs (Malicious parties can communicate with each other as they choose, but this communication is not counted in the total communication.) Throughout each phase, the goal is for the leader to build and send out certificates, which will ensure that the leader made all honest parties agree on the same output, which satisfies some additional, protocol-specific properties. Once an *honest* leader has sent such a certificate, all subsequent honest leaders will not invoke any communication during their respective phases, i.e., they remain silent. Since it typically takes f phases for an honest leader to be picked (and nobody speaks after that) and each phase typically causes the communication of $O(n)$ words (due to expander graphs), protocols in this paradigm achieve $O(n \cdot f)$ communication.

Constructing RV: Polling and Rebuttal. Our RV protocol with respect to the threshold predicate consists of two subprotocols, *Polling* and *Rebuttal*—see Figure 2.

The goal of the Polling protocol is for honest parties to agree on a first estimation on who has signed which value. In particular, in Polling, each party starts with an input value and must output a *ballot* and potentially an *accusation list*. The accusation list is a subset S of $[n]$ and the ballot contains two counters, count_0 and count_1, a commitment (hash) H_A, and a proof to verify the proper construction of the counters and the hash. Not only does Polling guarantee honest parties agree on the same ballot, but it also ensures (1) count_0 corresponds to the size of some set $S_0 \subseteq [n] - S$ of parties that signed value $0, r$; (2) count_1 corresponds to the size of some set $S_1 \subseteq [n] - S$ of parties that signed value $1, r$; (3) $S_0 \cup S_1 \cup S = [n]$; (4) H_A is a commitment to the output accusation list (The accusation list, therefore, contains parties that are still unaccounted for and Polling ensures at least one honest party outputs it.) We note here that while honest parties can be assured about the validity of ballots by actually verifying the set of all involved signatures and checking all conditions above, this would be very communication inefficient. Instead, Polling uses succinct arguments (SNARKs) [24], which are crucial for achieving our final $O(n \cdot f)$ communication bound. We finally note that our proposed Polling protocol (see Figure 4)

is a leader-based process based on the silent phases paradigm, thus achieving communication complexity $O(n \cdot f)$.

If one were to envision Polling as an untrustworthy voting process, then Rebuttal acts as an opportunity for accused parties to redeem themselves. Because our implementation of Polling is leader-based, a malicious leader might be able to construct verifiable ballots that selectively exclude signatures from honest parties. Thus, a malicious adversary can bias the tallies of a seemingly correct ballot. In our implementation of Rebuttal (see Figure 5), parties who were in the accusation list of the ballot output by Polling, get to send their own values directly to the other parties. This is done efficiently by bounding the size of the accusation list to $O(f)$, ensuring that the communication does not exceed $O(n \cdot f)$. Ultimately, by the conclusion of Rebuttal, all honest parties' initial values will be accounted for, thereby yielding provable validity as required by RV.

WBA Definition by Cohen et al. [15] versus our WBA definition. Our WBA definition resembles the WBA definition of Cohen et al. [15] but differs in the validity property that is provided. In particular, Cohen et al. [15] allow parties to output $\perp$ if more than one valid values (according to a predicate) exist during the execution. In our definition, we never output $\perp$ and parties agree on a value v if there is only a *single* input v (to the protocol) that satisfies the predicate (See Definition 2.3.) For example, in our definition, it might be the case that all honest parties start with the same value v that satisfy the predicate, but another value v' that satisfies the predicate is also input to the protocol (e.g., as input of a malicious party). In this case our protocol will guarantee agreement, but not necessarily on v. We note here that while delving into the details of the Cohen et al. protocol [15], we discovered an attack that breaks the agreement property of their definition. In Appendix B we present the attack and propose a fix. We do not prove the entirety of their protocol correct from scratch but our own WBA protocol uses the idea behind our proposed fix.

Our WBA Protocol. Similarly to [15], our WBA protocol is constructed following the silent phases paradigm. Abstractly, each non-silent phase of WBA executes four rounds of message exchanges between the leader and each party. The goal of an honest leader is to convince the honest parties to agree with the honest leader's proposal. Each party has three states. neutral, committed, and decided. At the start of WBA parties are undecided. States transfer across phases and are one-way: neutral parties can become committed and committed parties can become decided, but no other switch occurs. A decided leader will remain silent.

Intuitively, in each round of a non-silent phase, a leader sends a proposal of some form. Depending on their state and the leader's proposal, parties might either reply to the leader or not. Depending on the number of replies received, the leader might be able to construct a valid certificate to convince parties to change their state. Our construction guarantees that if all honest parties start WBA with the same value and proof that the value satisfies a pre-defined predicate, the leader can create valid certificates only on that value. This ensures restricted validity. During the protocol, we use several primitives. We apply SNARKs, threshold signatures, and collision-resistant hash functions to reduce the communication cost of the required proofs and certificates during each phase. This, in conjunction with the use of expanders, facilitates the efficient dissemination of certificates between parties, achieving the targeted $O(n \cdot f)$ word complexity.

1.2 Related Work.

Byzantine agreement has a long history in the literature, dating back to the seminal work of Shostak, Pease, and Lamport [27]. While their pioneering work [27, 30] introduced initial solutions, both protocols suffered from exponential communication complexity. Subsequently, Dolev and Strong presented a protocol with improved message complexity[4] of $O(n^2 \cdot t)$ [20], which was later proven to be optimal by Dolev and Reishuk [18]. However, the optimal message complexity of $O(n^2)$ in the Dolev-Strong protocol came at the cost of cubic communication complexity due to message length of $O(t)$ signatures. Later, Momose and Ren proposed a protocol with optimal $O(n^2)$ complexity [29].

Improved Communication Complexity. Nevertheless, quadratic complexity remains impractical for large-scale applications. Recently, several lines of work have attempted to circumvent the quadratic complexity either through randomization [2, 9, 25], which weakens the protocol guarantees to probabilistic ones and is inherently insecure against a strong adaptive adversary, or optimistic execution [32, 23, 35, 6], where the expected complexity is sub-quadratic under ideal conditions like synchronous execution and the absence of failures. Another line of work is adaptive communication complexity [15, 32], which is the focus of this work and where the communication complexity is expressed as a function of the exact number of malicious parties during the protocol execution, which

[4] It is noteworthy that the Dolev-Strong protocol directly achieves $O(n^2(f+1))$ adaptive message complexity, by performing the communication analysis for the actual number of corruptions f, on a given run, instead of the upper bound t.

we denote by f. The work by Spiegelman [32] adapts the Dolev-Reishuk bound based on the actual number of malicious parties, proving a more practical lower bound of $\Omega(ft + t)$, and presents a protocol that achieves optimistic synchronous Byzantine agreement with an asynchronous fall-back algorithm that tolerates resilience of $t < \frac{n}{3}$. Cohen et al., [15] a Broadcast protocol with a communication complexity of $O(n \cdot f)$. To achieve Broadcast, they introduce a Weak Byzantine Agreement protocol with $O(n \cdot f)$ communication, which they use as a subroutine to achieve Broadcast. Both protocols tolerate $t < \frac{n}{2}$. Note, this is not the optimal resilience for authenticated broadcast, as it is known it can tolerate up to $t < n$ malicious parties [20]. Another line of work includes early-stopping protocols [19, 17, 3], which aim to achieve consensus in $\min\{f+2, t+1\}$ rounds, resulting in communication complexity dependent on f. Table 1 shows a comparison of previous works with our result.

Concurrent Work by Civit et al. [14]. Concurrently with our work, Civit et al. [14] proposed a BA protocol with $O(n \cdot f)$ communication complexity. This work appeared online after the SODA 2024 submission deadline. Their approach is also based on silent phases and requires running a pre-processing protocol to create a constant-size verifiable certificate (that a value is safe to vote on) so that validity is not violated later when running a WBA. Both our protocols achieve comparable suboptimal resilience—achieving the same results with optimal resilience $t < n/2$ remains an open problem. Note, their protocol is based on Cohen's WBA [15], which we have identified as having a vulnerability due to an attack on agreement.

1.3 Paper Roadmap. Section 2 provides definitions for BA, RV and WBA as well as for the cryptographic primitives we employ such as SNARKs, threshold signatures and collision-resistant hash functions. Section 3 discusses the intuition and construction of the RV protocol. Section 4 presents the WBA protocol. The correctness proofs of the protocols and some key lemmas are presented in Appendix A. Furthermore, the attack in the WBA protocol proposed in [15] as well as our fix are presented in Appendix B.

2 Preliminaries

We consider a set of n parties, namely $\{p_1, \ldots, p_n\}$, of which at most $t < (\frac{1}{2} - \epsilon)n$ parties could behave maliciously where $\epsilon \in (0, 1/2)$. We denote by $f \leq t$ the exact number of malicious parties. We assume that the protocols operate through synchronous rounds, where there is a known upper bound on the message delays. The communication network is deemed point-to-point, reliable, and authenticated. We consider an adaptive Byzantine adversary; it may deviate arbitrarily from the protocol, and it has the freedom to corrupt parties dynamically during the process, up to t parties. We assume the adversary to be computationally bounded that cannot break the cryptographic primitives used with non-negligible probability. It is important to note that once a party deviates from the protocol, it retains this status throughout the execution, thus called "malicious". Conversely, a party that does not become malicious throughout the entire execution is referred to as "honest". Furthermore, we assume a trusted public key infrastructure (PKI), where there exists a trusted party that generates a public/private key pair (pk_i, sk_i) for each party p_i and shares $(pk_1, \ldots, pk_n)$ and sk_i to each p_i such that all parties hold the (same) vector of public keys $(pk_1, \ldots, pk_n)$ but only p_i knows the respective sk_i. Since our later constructions require SNARKs, we further require the trusted party to share a Common Reference String (CRS) with all parties. We follow the widely adopted notion of complexity employed in previous works [2, 29, 15], the so-called word complexity. The communication complexity of a protocol is determined by the total number of words sent by all honest parties throughout the execution of the protocol. Here, a word comprises a constant number of signatures and values, and we assume that each message transmitted during the protocol contains at least one word.

2.1 Consensus Definitions. In this subsection, we define the primitives used to achieve Byzantine agreement. Specifically, our study focuses on the binary Byzantine agreement problem, where possible values are restricted to the set $\{0, 1\}$. Throughout the definitions presented, we use λ to denote the security parameter.

Definition 2.1. (Byzantine Agreement) *Let Π be an n-party protocol, where each honest party p_i starts with initial value $\mathsf{v}_i \in \{0,1\}$, and outputs value $\mathsf{y}_i \in \{0,1\}$. A protocol Π achieves Byzantine Agreement, with up to t malicious parties, if it achieves the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- *Validity: If all honest parties have initial value $\mathsf{v}_i = \mathsf{v}$, all honest parties output $\mathsf{y}_i = \mathsf{v}$;*
- *Agreement: All honest parties output the same value;*
- *Termination: All honest parties terminate.*

As mentioned earlier, our approach involves constructing a Byzantine Agreement protocol through the combination of a Reliable Voting protocol and a Weak Byzantine Agreement protocol. We define them formally.

DEFINITION 2.2. (RELIABLE VOTING) *Let Π be an n-party protocol, where each honest party p_i starts with initial value $\mathsf{v}_i \in \{0,1\}$ and outputs value $\mathsf{y}_i \in \{0,1\}$ and auxiliary information aux_i. Let $\mathsf{validate}(.,.)$ be a Boolean predicate on a party's output. A protocol Π achieves Reliable Voting with respect to $\mathsf{validate}(.,.)$, with up to t malicious parties, if it achieves the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- *Provable Validity: If all honest parties have initial value $\mathsf{v}_i = \mathsf{v}$, then (1) all honest parties output $\mathsf{y}_i = \mathsf{v}$ and aux_i such that $\mathsf{validate}(\mathsf{y}_i, \mathsf{aux}_i) = \mathsf{true}$ and (2) no party outputs $\mathsf{y}_j \neq \mathsf{v}$ and auxiliary information aux_j such that $\mathsf{validate}(\mathsf{y}_j, \mathsf{aux}_j) = \mathsf{true}$;*
- *Termination: All honest parties terminate.*

DEFINITION 2.3. (WEAK BYZANTINE AGREEMENT) *Let Π be an n-party protocol, where each honest party p_i starts with input $(\mathsf{v}_i \in \{0,1\}, \mathsf{aux}_i)$, and outputs value $\mathsf{y}_i \in \{0,1\}$. Let $\mathsf{validate}(.,.)$ be a Boolean predicate on a party's input. A protocol Π achieves Weak Byzantine Agreement with respect to $\mathsf{validate}(.,.)$, with up to t malicious parties, if it achieves the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- *Restricted Validity: If all honest parties have input $(\mathsf{v}_i = \mathsf{v}, \mathsf{aux}_i)$ such that $\mathsf{validate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$, and no party inputs $(\mathsf{v}' \neq \mathsf{v}, \mathsf{aux}_i)$ such that $\mathsf{validate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$, all honest parties output $\mathsf{y}_i = \mathsf{v}$;*
- *Agreement: All honest parties output the same value;*
- *Termination: All honest parties terminate.*

2.2 Cryptographic Primitives. In this section we introduce definitions for the various cryptograhpic primitives that our protocols will be using.

Succinct Non-Interactive Arguments of Knowledge (SNARKs) [24]. Let $\mathcal{R}$ be an efficiently computable binary relation that consists of pairs of the form (x, w) where x is a statement and w is a witness.

DEFINITION 2.4. (SNARKS) *A SNARK is a triple of PPT algorithms $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ defined as follows:*

- $\mathsf{Setup}(1^\lambda, \mathcal{R}) \rightarrow \mathsf{crs}$*: takes a security parameter λ and a binary relation $\mathcal{R}$ and outputs a common (structured) reference string* crs.
- $\mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}) \rightarrow \pi$ *: on input* crs*, a statement x and the witness* w*, outputs an argument π.*
- $\mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi) \rightarrow 1/0$*: on input* crs*, a statement* x*, and a proof π, it outputs either 1 indicating accepting the argument or 0 for rejecting it.*

It also satisfies the following properties:

- *Completeness: For all $(\mathsf{x}, \mathsf{w}) \in \mathcal{R}$, the following holds:*

$$\Pr\left[\mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi) = 1 \,\middle|\, \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ \pi \leftarrow \mathsf{Prove}(\mathsf{crs}, \mathsf{x}, \mathsf{w}) \end{array}\right] = 1$$

- *Knowledge Soundness: For any PPT adversary $\mathcal{A}$, there exists a PPT extractor $Ext_{\mathcal{A}}$ such that the following probability is negligible in λ:*

$$\Pr\left[\begin{array}{l} \mathsf{Verify}(\mathsf{crs}, \mathsf{x}, \pi) \\ \wedge \mathcal{R}(\mathsf{x}, \mathsf{w}) = 0 \end{array} \,\middle|\, \begin{array}{r} \mathsf{crs} \leftarrow \mathsf{Setup}(1^\lambda, \mathcal{R}) \\ ((\mathsf{x}, \pi); \mathsf{w}) \leftarrow \mathcal{A} \parallel_{\mathcal{X}_{\mathcal{A}}} (\mathsf{crs}) \end{array}\right]$$

- *Succinctness: For any x and w, the length of the proof π is given by $|\pi| = \mathsf{poly}(\lambda, \log|\mathsf{w}|, \log|\mathsf{x}|)$.*

Signature Schemes. We assume a signature scheme that supports threshold signatures [12, 31].

DEFINITION 2.5. (THRESHOLD SIGNATURES) *An $(n-t)$-out-of-n threshold signature scheme is a tuple of efficient algorithms* (KeyGen, SignShare, VerShare, Combine, Verify) *defined as follows.*

- $\mathsf{SS.KeyGen}(\lambda, n, n-t)$*: The randomized key generation algorithm takes as input the security parameter λ and the number of parties n in the protocol, it outputs a public and secret key share $(\mathsf{sk}_i, \mathsf{pk}_i)$ for each party, and a public key pk_{ss};*
- $\mathsf{SS.SignShare}(x, \mathsf{sk}_i)$*: On input a message x and a secret key share sk_i, it outputs a signature share σ_i;*
- $\mathsf{SS.VerShare}(\mathsf{pk}_i, x, \sigma_i)$*: On input a public key share pk_i, a message x, and a signature share σ_i, it outputs 1 if the signature is correct or 0 otherwise;*
- $\mathsf{SS.Combine}((\mathsf{pk}_1, \ldots, \mathsf{pk}_n), x, (\sigma_1, \ldots, \sigma_{n-t}))$*: On input a vector of public key shares $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, a message x, and a set S of $n-t$ signature shares (σ_i, i) (with corresponding indices), it outputs a signature σ or $\perp$;*
- $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, x, \sigma)$*: On input a public key pk_{ss}, a message x, and a signature σ. It outputs 1 if the signature is correct or 0 otherwise.*

The signature scheme should satisfy Unforgeability defined as follows:

Unforgeability: For any PPT adversary $\mathcal{A}$ that can corrupt up to t parties, given a share signing oracle $\mathcal{S}$, the following probability is negligible in λ:

$$\Pr\left[\begin{array}{r|l} \sigma' \leftarrow \mathcal{A} & (\mathsf{pk}_{ss}, (\mathsf{pk_i}, \mathsf{sk_i})_{\mathsf{i}\in\mathsf{n}}) \leftarrow \mathsf{SS.KeyGen}(\lambda, n, n-t) \\ \mathsf{SS.Verify}(\mathsf{pk}_{ss}, m', \sigma') = 1 \wedge m' \neq m & (m, \sigma_i) \leftarrow \mathcal{A}^{\mathcal{S}}(pk_{i\in n}) \end{array}\right]$$

Collision-Resistant Hash Functions. We use a cryptographic collision-resistant hash function Hash, which maps an arbitrary length message to a fixed length output.

DEFINITION 2.6. (COLLISION-RESISTANT HASH FUNCTIONS) *We say a family of hash functions $\mathcal{H}$, is collision-resistant if for a random $\mathsf{Hash} \in_R \mathcal{H}$ it is hard for a polynomially bounded adversary to come up with two preimages $x \neq y$ that collide $(\mathsf{Hash}(x) = \mathsf{Hash}(y))$.*

Signatures and Hashing Notation. Throughout the paper, we use the abbreviated notation $\langle m \rangle_{\sigma_i}$ to refer to tuples $(m, \mathsf{sig}_i(m))$, where m is some message and $\mathsf{sig}_i(m)$ is the valid signature on that message for party p_i with public key pk_i. We utilize the function Hash to compute the hash value of a list of public keys $\mathsf{pk}_1, \ldots, \mathsf{pk}_r$. To achieve this, we sort the list in ascending order and then hash the concatenation of the sorted list.

In Table 2 we provide an overview of the variables and predicates used across our protocols.

2.3 Expander Graphs.

Our protocol in Section 3 and Section 4 uses expanders. Expanders are sparse graphs with $O(1)$ degree and have good connectivity.

DEFINITION 2.7. (EXPANDERS) *Let α and β be constants satisfying $0 < \alpha < \beta < 1$. We define an (n, α, β)-expander to be a graph of n vertices such that, for any set S of αn vertices, the number of neighbors of S is greater than βn.*

We use expanders as defined in [29], where $\alpha = 2\epsilon, \beta = 1 - \alpha$, for constant $\epsilon \in (0, 1/2)$. In [29] they prove that such expanders exist with non-negligible probability, and can be obtained with overwhelming probability via a probabilistic algorithm within at most polynomially many attempts. We follow their notation and define a $(n, 2\epsilon, 1-2\epsilon)$ expander graph as $\mathcal{G}_{n,\epsilon}$.

Note that the use of expanders does not directly render a protocol randomized. Our protocol shown later is deterministic in the sense that every operation after initialization (e.g., cryptographic-keys generation) is deterministic (e.g., there is no random committee election). The expander can be computed during initialization (and provided by the trusted party) or be pre-calculated. In the latter case, the expander can serve multiple executions of the protocol for the same number of parties.

Table 2: Variables and predicates used throughout all our protocols used by party p_i.

Variable	Definition
N	Set of public keys belonging to the n parties.
$\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$	r: phase number in which ballot was created; count_0: number of partis that have signed 0; count_1: number of parties that have signed 1; H_A: a hash of the list of public keys of accused parties; π_{ballot}: a proof to verify consistency of $(\mathsf{count}_0, \mathsf{count}_1, H_A)$.
accList	list of accused parties committed to by H_A.
Ψ_i	Data structure that holds all accList that p_i has received with respect to ballot. $\Psi_i[\mathsf{ballot}]$ retrieves accList if p_i has received it, or $\perp$ otherwise.
QC_{commit}	Aggregate signature of $n-t$ distinct signatures on $\langle \mathsf{ballot}, r \rangle$ for Π_{Polling}. Aggregate signature of $n-t$ distinct signatures on $\langle \mathsf{v}, r \rangle$ for Π_{WBA}.
Commit Certificate for $\Pi_{\mathsf{Polling}}/\Pi_{\mathsf{WBA}}$	$\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle / \langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle$
QC_{decide}	Aggregate signature of $n-t$ distinct signatures on $\langle decide, \mathsf{ballot}, r \rangle$ for Π_{Polling}. Aggregate signature of $n-t$ distinct signatures on $\langle decide, \mathsf{v}, r \rangle$ for Π_{WBA}.
Decide Certificate for $\Pi_{\mathsf{Polling}}/\Pi_{\mathsf{WBA}}$	$\langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle / \langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle$.
$\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux})$	$\mathsf{SNRK}_3.\mathsf{Verify}(\mathsf{crs}_3, (\mathsf{v}), \mathsf{aux})$ (SNRK_3 is described in Section 3.2).
$\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$	true if $\mathsf{v}_{\mathsf{commit}} = \perp$ **and** $\big(\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ **or** $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}\big)$.

3 Reliable Voting (RV)

In this section we propose a protocol Π_{RV} for RV—see Figure 3. Recall that in RV, each party starts with initial value $\mathsf{v}_i \in \mathsf{V}$, where $\mathsf{V} = \{0, 1\}$, and generates output value $\mathsf{y}_i \in \mathsf{V}$, along with auxiliary information aux_i. As mentioned earlier, RV plays a crucial role in ensuring validity for BA through its provable validity property. Our Π_{RV} consists of two subroutines: the Polling protocol (Section 3.1) and the Rebuttal protocol (Section 3.2), as depicted in Figure 3. Our Π_{RV} protocol is instantiated with respect to the Boolean predicate ThresholdPredicate. Specifically, $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i)$ is set to true if aux_i contains proof of the existence of $n-t$ distinct signatures (votes) on v, r for some $r \in [n]$, where the proof here is constructed using SNARKs.

Algorithm $\Pi_{\mathsf{RV}}(\mathsf{v}_i)$:

1: $\mathsf{ballot}_i, \mathsf{accList}_i \leftarrow \Pi_{\mathsf{Polling}}(\mathsf{v}_i)$
2: $\mathsf{y}_i, \mathsf{aux}_i \leftarrow \Pi_{\mathsf{Rebuttal}}(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$

return $(\mathsf{y}_i, \mathsf{aux}_i)$

Figure 3: Code of Π_{RV} for party p_i.

3.1 Polling. The Polling protocol (Figure 4) comprises two crucial steps: the leader-based step, which is built upon the silent phases framework, and the processing step. In essence, the Polling protocol aims to tally parties' initial values of the form $\langle \mathsf{v}_i, r \rangle$, where r is the current phase number, create a list of parties whose votes weren't

counted, accList, provide proof of correct tallying, and share the result with everyone in the form of a ballot. By the end of Polling, it is guaranteed that all honest parties agree on the same ballot, however, parties reach weak agreement on accList; An honest party either outputs $\mathsf{accList}_i = \mathsf{accused}$ or $\perp$, but no other honest party output $\mathsf{accList}_j \neq \mathsf{accused}$.

During the leader-based step, each leader, if undecided, requests parties to send their initial values. A leader p_i is considered undecided if $\mathsf{ballot}_i = \perp$. For a leader to create a ballot, the leader counts the initial values (votes) received for each value $\in \{0, 1\}$, count_0 and count_1, and compiles a list of accused parties (accList) from whom the leader allegedly did not receive votes, along with a hash of that list (H_A). It is crucial for honest parties to ensure that the initial values of accused parties are not included in the ballot's count to avoid double counting in the Rebuttal step. Therefore, the leader creates a proof π_{ballot}, which is included in ballot, that verifies three essential properties: firstly, the tally ($\mathsf{count}_\mathsf{v}$) for a specific value v is computed from the signed initial values sent by a set of parties in $P_\mathsf{v} \subset N$; secondly, the aforementioned set of parties, along with the set of accused parties, are mutually exclusive, and when combined, they form the complete list of all parties involved in the protocol (correctness property in Theorem 3.1). Finally, the vote of party p_j is included in the count of the respective value; p_j is not in accList. The ballot takes the form of $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$, and the leader sends ballot to each party p_j. An honest party first verifies the correctness of the proof and then signs the ballot to the leader. In the following rounds, the leader attempts to get parties to decide on this ballot, which we discuss in detail in the next section.

Flexible Threshold. Constructing a ballot involves collecting initial values from parties and tallying them, where the number of initial values included directly influences the size of the list of accused parties who are claimed to have not provided values. Consequently, it becomes crucial to determine an appropriate threshold for the minimum number of votes required to form a valid ballot. Failure to set a suitable threshold could result in a substantial increase in communication complexity during the Rebuttal protocol, since the accused parties would receive accusing messages from all parties during it, resulting in a communication complexity of $O(n \cdot |list|)$. Naively, one might set the threshold to $n - t$. However, setting a fixed minimum threshold of $n - t$ is not feasible, as it would allow a malicious leader to manipulate the system by submitting a list of $O(t)$ accused parties whose values were not included in the tally, even if the leader received more than $n - t$ initial values. This would result in a quadratic complexity of $O(n \cdot t)$ in the Rebuttal protocol instead of $O(n \cdot f)$ even if $f < O(t)$. To mitigate this problem, we propose a novel idea of a flexible threshold.

In the first round, the leader attempts to construct an n-ballot consisting of initial values received from n parties. If unsuccessful, it indicates the presence of at least one malicious party $f \geq 1$, who is either the leader or one of the $n - 1$ parties that did not send their initial value to the leader. In the second phase, the second leader decreases the threshold by 1 and attempts to create at least an $(n - 1)$-ballot. If successful, it ensures the existence of at least one malicious party from the previous phase. Otherwise, the protocol moves on to the third phase, where the leader needs to collect $n - 2$ initial values. This process continues until phase $n - t + 1$, where a created valid ballot should contain at least $(n - t)$ votes. In subsequent rounds, the leader can only create a ballot of the minimum threshold $n - t$. This flexible threshold concept ensures that the accused list can be constructed without causing communication complexity to exceed $O(n \cdot f)$. Precisely, we prove in Lemma 4 that the size of the accused list is at most $2f - 1$. Furthermore, while this leader-based step is built upon the silent phases framework, it is not guaranteed that it will only incur $f + 1$ non-silent phases. In other words, if there is a phase $r < f$ where the leader is honest and undecided, it is not certain that the leader will be able to create a valid ballot and convince all honest parties to decide on it by the end of that phase. This uncertainty is due to the flexible threshold, as a leader might not be able to collect sufficient initial values to meet the threshold required for creating the ballot. Nevertheless, in Lemmas 3,2, we demonstrate that the run consists of at most $2f + 1 = O(f)$ non-silent phases, and at most f honest parties remain undecided.

Thus, the leader-based step is followed by a processing step, during which undecided parties send a help request to all parties, seeking assistance from parties that have already made their decision to obtain the agreed-upon ballot. This ensures that all parties reach a decision before Rebuttal. As the number of undecided honest parties is bounded by f, the processing step incurs at most $n(2f)$ communication, where the additional f is due to the malicious parties.

Protocol Overview. The protocol consists of a leader-based part and a processing part. The former runs for n phases, each phase involving four steps; prepare, pre-commit, commit and decide.

(prepare) In each phase, the leader remains silent if decided ($\mathsf{ballot}_i \neq \perp$). Otherwise, the leader seeks help by sending $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ to all parties. Upon receiving $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ from the leader, if a party p_j is committed to

a certain ballot, $\mathsf{ballot}_{\mathsf{commit}} = \mathsf{ballot}$, in a previous phase, p_j forwards ballot to the leader along with $\mathsf{rank}_{\mathsf{commit}}$ (the phase in which commit certificate was created), and π_{commit} (proof of the validity of the commit certificate). The π_{commit} is $n-t$ signatures on $\langle \mathsf{ballot}, \mathsf{rank}_{\mathsf{commit}} \rangle$ (QC_{commit}). Otherwise, p_j sends to the leader the initial value $\langle \mathsf{v}_j, r \rangle_\sigma$, where r is the current phase number. Note, that including the phase number in the vote is important to prevent malicious leaders from using votes from older phases to create a valid ballot.

(pre-commit) In case the leader receives multiple valid commit certificates $\langle \mathsf{ballot}, \mathsf{rank}_{\mathsf{commit}}, QC_{\mathsf{commit}} \rangle$ from parties, where a commit certificate is considered valid if $\mathsf{SS.Verify}(pk_{ss}, \langle \mathsf{ballot}, \mathsf{rank}_{\mathsf{commit}} \rangle, QC_{\mathsf{commit}}) = 1$, the leader selects the one with the highest $\mathsf{rank}_{\mathsf{commit}}$. Subsequently, the leader proposes the corresponding ballot in the commit certificate with the highest rank to the parties, accompanied by π_{commit} and $\mathsf{rank}_{\mathsf{commit}}$. Otherwise, the leader uses the initial values received to form ballot. To create a ballot, the leader uses a collision-resistant hash function (Definition 2.6) to form a hash H_A on the list of parties that the leader accuses of not submitting any votes. The leader also includes the count of the set of parties voted on $0, r$ and the set of parties voted on $1, r$. Furthermore, the leader provides evidence to validate the legitimacy of the generated $\langle \mathsf{count}_0, \mathsf{count}_1, H_A \rangle$ by including a proof π_{ballot}. Let $\mathsf{SNRK_1}$ be a SNARK system per Definition 2.4. The leader p_r constructs π_{ballot} for the public statement:

$$\mathsf{x} = (\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A \rangle) ,$$

The above public NP statement is verified using the following witness:

$$\mathsf{w} = (P_0, P_1, S_0, S_1) ,$$

where P_0 is a vector of public keys, P_1 is another vector of public keys, S_0 is a set of signatures and S_1 is another set of signatures. The verification computation $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, \mathsf{x}, \pi_{\mathsf{ballot}})$ outputs "1" if and only if the following checks are true:

1. $\mathsf{count}_0 = |P_0|$ and $\mathsf{count}_1 = |P_1|$;
2. $P_0 \subseteq N$, $P_1 \subseteq N$ and $P_0 \cap P_1 = \emptyset$;
3. $H_A = \mathsf{Hash}(N - P_0 - P_1)$;
4. S_0 are valid signatures on $\langle 0, r \rangle$ by public keys in P_0;
5. S_1 are valid signatures on $\langle 1, r \rangle$ by public keys in P_1;

Finally, the leader sends $(\langle \mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r})$ to each party.

If a party receives a ballot proposal from the leader associated with a commit certificate with rank greater than $\mathsf{rank}_{\mathsf{commit}}$, the party propagates the proposal ballot though the expander. Otherwise, if a party receives ballot, the party verifies whether $\mathsf{Hash}(\mathsf{accList}) = H_A$, $|\mathsf{accList}| \leq \min\{r-1, t\}$, $p_i \notin \mathsf{accList}$, and $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, (r, \mathsf{count}_0, \mathsf{count}_1, H_A), \pi_{\mathsf{ballot}}) = 1$. If the verification is successful, the party propagates ballot through the expander. Since the leader might send a conflicting ballot', we use an $(n, 2\epsilon, 1-2\epsilon)$-expander with constant degree. After propagation, each party checks if received a conflicting ballot' before sending vote $\langle \mathsf{ballot}, r \rangle_{\sigma_i}$ to the leader.

(commit) If the leader receives at least $n-t$ precommit messages of the form $\langle \mathsf{ballot}, r \rangle_{\sigma_j}$, the leader creates a valid commit certificate $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$ by aggregating the messages received forming QC_{commit} and sends the commit certificate to all parties. If a commit certificate exists, then $n - t > t + 2\epsilon$ parties, of which at least 2ϵ honest parties, must have propagated the leader's ballot proposal to their neighbors. The expansion property implies more than $(1-2\epsilon)n > 2t$, out of which at least $t+1$ honest parties would receive ballot. They would never vote for $\mathsf{ballot}' \neq \mathsf{ballot}$, so a conflicting commit certificate cannot exist (Lemma 6). Note that $\mathsf{accList}$ is not propagated or signed by the parties, nor is it included in commit certificate as it is of size $O(f)$. Had it been included in commit certificate, f the malicious leader could influence honest parties to send the commit certificate to which they are committed in *(pre-commit)*, incurring $O(nf^2)$ communication complexity for $O(f)$ phases. However, the parties store $(\mathsf{ballot}, \mathsf{accList})$ for future purposes. Upon receiving a valid commit certificate from the leader, the party updates commit variables; $\mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}}$, and propagates the commit certificate received through the expander. Consequently, the party sends a matching decide message

of $\langle decide, \mathsf{ballot}, r\rangle_{\sigma_i}$ to the leader, where $\mathsf{ballot} \in$ commit certificate received from the leader in the previous round. As the adversary can potentially generate two conflicting commitment certificates in the run using two consecutive malicious leaders, it is not safe to decide on the ballot that the party is committed to. Thus, we introduce an additional layer of protection in the form of another round of voting, *(decide)* round.

(decide) If the leader receives at least $n-t$ decide messages on the same ballot, the leader creates a valid decide certificate by aggregating the messages received to form QC_{decide}, and multicasts $\langle \mathsf{ballot}, r, QC_{\mathsf{decide}}\rangle_{\sigma_r}$ to parties. Once received a valid decide certificate, the party sets $\mathsf{ballot}_i = \mathsf{ballot}$. If stored $\mathsf{accList}$ for ballot_i in *(pre-commit)*, $\Psi_i[\mathsf{ballot}_i] \neq \perp$, the party sets $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i]$. Consequently, the party is decided and remains silent.

(processing.) By the end of the n phases, it is not guaranteed that all parties are decided due to the decreasing threshold. For example, in a scenario where $f = O(t)$ and the initial leaders are honest, it is possible that they cannot gather sufficient initial values to meet the threshold required for the creation of the ballot. Consequently, the following malicious leader will cause the remaining honest parties, who are designated as leaders for the remaining n phases, to be decided, resulting in their silence during their respective phases. Thus, after the n phases, if an honest party is still undecided, the party sends a help message $\langle help\rangle_{\sigma_j}$ to all parties. Upon receiving a help message, an honest party sends back the ballot decided during Polling, and proof of decision π_{decide} to all parties who sent the help message. As a result, we guarantee that all honest parties are decided before starting the rebuttal protocol. In Lemma 2, we prove that the number of undecided honest parties is upper-bounded by $O(f)$. Thus, the overall communication complexity of this step is $O(n \cdot f)$.

Protocol Π_{Polling} achieves the properties of Theorem 3.1, which we prove in Appendix A.

THEOREM 3.1. (POLLING) *Assume an execution of protocol Π_{Polling} (Figure 4), where each party p_i inputs value $\mathsf{v}_i \in \{0,1\}$, and outputs values $\mathsf{ballot}_i = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}}\rangle$ and $\mathsf{accList}_i$, where $\mathsf{accList}_i$ is either a subset of $[n]$ or $\perp$. Then, Π_{Polling} satisfies the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- *Agreement: If an honest party outputs ballot_i, no honest party outputs $\mathsf{ballot}_j \neq \mathsf{ballot}_i$;*
- *Correctness: (i) There exists a set of parties P_0, with $|P_0| = \mathsf{count}_0$, that have signed $\langle 0, r\rangle$; (ii) There exists a set of parties P_1, with $|P_1| = \mathsf{count}_1$, that have signed $\langle 1, r\rangle$; (iii) P_0 and P_1 are disjoint; (iv) H_A is the collision-resistant hash of the set $[n] - P_0 - P_1$; (v) π_{ballot} is a SNARK proof that $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A\rangle$ satisfy properties (i) to (iv);*
- *Count validity: Let h be the number of honest parties in the set of parties committed to by H_A. If honest parties start with the same value $\mathsf{v}_i = \mathsf{v}$, then $\mathsf{count}_{\mathsf{v}} + h \geq n - t$;*
- *Integrity: If an honest party outputs $\mathsf{accList}_i \neq \perp$, then $\mathsf{accList}_i = [n] - P_0 - P_1$;*
- *Reliability: At least one honest party outputs $\mathsf{accList}_i \neq \perp$;*
- *Termination: All honest parties terminate.*

Subsequently, we prove that Π_{Polling} incurs overall $O(n \cdot f)$ communication complexity. To do so, we begin by demonstrating that if a leader is honest and undecided in phase $r > f$, all honest parties will reach a decision by the end of this phase. This observation leads us to the conclusion that there are at most $2f + 1$ non-silent phases throughout the run.

LEMMA 1. *If a leader is honest and undecided in phase $r > f$, all honest parties are decided by phase r.*

Proof. Assume some phase $r > f$ and let leader p_r for that phase be honest. In Round 1, p_r ,if undecided, sends $\langle \mathsf{value\_req}\rangle$ to all parties. To propose a ballot in *(precommit)*, p_r can either propose a ballot included in a commit message received from some party at the beginning of the round or create a new ballot and π_{ballot} from the initial values received from parties. If the leader receives some commit message, p_r will propose the respective valid ballot with the greatest respective phase number.

Conversely, by construction, if the leader does not receive any commit message, then no honest party is committed. Therefore, the leader needs to create a new ballot by collecting enough initial values to meet the threshold for the creation of the ballot. Since $r > f$, the leader needs to collect at least $n-r+1 \leq n-(f+1)+1 \leq$

$n - f$ initial values from parties. Since no honest party is committed, then every honest party sends the initial value to the leader. So, the leader will collect at least $n - f$ initial values and will meet the threshold for the creation of the ballot. Consequently, the leader will send a valid ballot proposal to all parties.

Once the honest leader proposes a valid ballot to the parties, all honest parties will vote for it in *(round 3)*. Thus, the leader will collect at least $n-t$ votes to create QC_{commit} and a valid commit certificate $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$, which the leader will broadcast to all parties. Upon receiving a valid commit certificate, every honest party will send to the leader a matching decide message $\langle decide, \mathsf{ballot}, r \rangle$, allowing the leader to collect at least $n-t$ decide messages to form QC_{decide} and a valid decide certificate that the leader broadcasts to all parties. Finally, once an honest party receives a valid decide certificate, sets ballot_i to the received ballot, which is the one the honest leader proposed for the respective commit and the decide certificate. Thus, all honest parties are decided at the end of phase r. □

From Lemma 1, we prove that the number of undecided honest parties before the processing step, and the number of non-silent phases are bounded by $O(f)$.

LEMMA 2. *Before processing (Line 46 of Π_{Polling}), at least $n-2f$ honest parties are decided, and at most f honest parties are undecided.*

Proof. Given that the leading-based step spans n phases, every honest party has the opportunity to act as a leader during its respective phase, provided that it has not already decided. From Lemma 1, if a leader is honest and undecided in phase $r > f$, every honest party is decided at the end of their phase. Thus, if not all honest parties are decided at the end of Polling, the adversary must have made every honest party that is supposed to be a leader in phases $r > f$ decided by phase f. As there are n phases in the run and the number of malicious parties that could potentially be leaders after phase f is f, at least $n-f-f$ honest parties are decided at the end of Polling. Since there are f malicious parties, at most $n-(n-2f)-f = f$ honest parties are undecided. □

LEMMA 3. *There are at most $2f+1$ non-silent phases in Π_{Polling}.*

Proof. In Polling, each honest party p_i is the leader during phase i. From Lemma 1, if an honest leader p_r is undecided at the beginning of phase $r > f$, then after phase r all honest parties are decided and every honest party $p_{r'}$ remains silent for phases $r' > r$. Since there are n phases and at most f malicious parties that could be leaders after phase f, there are at most $f + f + 1$ non-silent phases during Π_{Polling}. □

The minimum threshold for the number of initial values required to create a ballot is determined by the phase number, which, in turn, affects the maximum size of the accused list for that phase. This relationship is derived from the fact that the size of the accused list is calculated as the difference between the total number of parties n and the minimum threshold for that phase. By previously proving that there are at most $2f + 1$ non-silent phases, we proceed to prove Lemma 4, which bounds the size of the accused list to $O(f)$.

LEMMA 4. *There are at most $2f - 1 = O(f)$ accused parties in $\mathsf{accList}$.*

Proof. The size of $\mathsf{accList}$ depends on the phase at which the ballot ballot, which all parties decide, is formed. Let r be the first phase in which ballot is considered valid. Then, by construction, ballot cannot already be accompanied by a commit certificate. Therefore, it must include a valid SNRK_1 proof and must be accompanied by an accused list $\mathsf{accList}$. According to Line 23, an honest party accepts ballot only if the size of $\mathsf{accList}$ is at most $n - \max\{n-r+1, n-t\} = \min\{r-1, t\}$. If $r \leq f$, $|\mathsf{accList}| < f = O(f)$. Otherwise, if $r > f$, we distinguish two cases based on whether ballot is created by an honest or malicious leader. In phase $r > f$, an honest leader collects at least $n - f \geq \max\{n - r + 1, n - t\}$ initial values, resulting in $|\mathsf{accList}| \leq f$. On the other hand, a malicious leader can have ballot accompanied by $\mathsf{accList}$ which includes parties who actually sent their values to the leader. Despite this, due to the constraint of at most $2f+1$ speaking phases from Lemma 3, a malicious leader must construct the ballot agreed by the parties at most by phase $2f$. Thus, the maximum size of the $\mathsf{accList}$ a malicious leader of phase $r \in \{f+1, \ldots, 2f\}$ can send is limited to at most $\min\{2f-1, t\}$, else honest parties do not accept $\mathsf{accList}$ in Line 23. □

Finally, we use Lemmas 2,3 and 4 to prove that Π_{Polling} incurs $O(n \cdot f)$ communication complexity.

THEOREM 3.2. Π_{Polling} *has a communication complexity of* $O(n \cdot f)$.

Proof. To assess the communication complexity of the polling consensus protocol, we analyze the communication for each phase. Each phase comprises four rounds, namely prepare, precommit, commit, and decide.

(prepare) In this round, the leader sends a message of size $O(1)$, denoted as $\langle \mathsf{value\_req} \rangle$, to all parties if it is undecided. Honest parties reply by sending either a commit certificate consisting of their committed ballot $\mathsf{ballot}_{\mathsf{commit}}$ and π_{commit}, both of size $O(1)$, or their initial variable $O(1)$. The total communication complexity for (prepare) is $O(n)$.

(precommit) If the leader receives a valid commit certificate from a party, it selects the highest-ranked one and proposes the ballot contained in it, which is of size $O(1)$, to all parties. Otherwise, if the leader gathers enough initial values from the parties to reach the threshold for the creation of a ballot, it creates a ballot (that includes π_{ballot}) and an accussed list and sends them to all parties. The accused list contains its members, which are $(O(f))$ from Lemma 4, resulting to $O(n \cdot f)$ communication for this round. If the leader does not receive a commit certificate or enough initial values to create a new ballot, it remains silent for the rest of the phase. If a party receives a valid ballot proposal, it propagates it to its $O(1)$ neighbors using the expander, without transmitting the accused list to avoid increasing the communication complexity. Finally, the honest party sends its vote on the ballot proposed to the leader. In total, the communication complexity is $O(n \cdot f)$ if the leader creates a ballot, and $O(n)$ if the leader proposes a ballot in a received commit certificate.

(commit) In this round, after receiving enough votes to create a valid commit certificate, the leader sends it to all parties, incurring a total communication complexity of $O(n)$. If a party receives a valid commit certificate, it sends a matching decide message $\langle \mathit{decide}, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle$ to the leader. Thus, the total communication complexity of this round is $O(n)$.

(decide) Similarly, upon receiving enough votes to create a valid decide certificate, the leader sends the certificate to all parties, incurring a total communication complexity of $O(n)$. Once a leader sends a valid decide certificate, all honest parties are decided, and they will be silent as leaders.

It should be noted that $O(n \cdot f)$ communication complexity in the precommit round is incurred by at most one honest leader in the run. This is because once a valid proposal (ballot) is sent, the leader can collect enough $n - t$ votes to create valid QC_{commit} and QC_{decide}, given that $t < (\frac{1}{2} - \epsilon)n$. Therefore, all honest parties will decide by the end of this honest leader's phase and will not speak in their phases. From Lemma 3, there are at most $2f + 1$ non-silent phases. Thus, the overall communication complexity of the first n phases is $O(n \cdot f) + O(n) \cdot (2f) = O(n(2f + 1)) = O(n \cdot f)$.

(processing) In the three rounds of processing, undecided parties send help messages to all parties. Since the number of undecided honest parties is upper-bounded by f (Lemma 2), at most $2f$ parties send help messages to everyone, whereas the other f could come from malicious parties. In the next round, the decided honest parties send ballot_i and π_{decide} to the parties who sent the help messages. Since both messages are of size $O(1)$, the communication complexity of these three rounds is $O(n \cdot f)$.

Thus, the overall communication complexity of Polling is $O(n \cdot f)$. □

Algorithm $\Pi_{\mathsf{Polling}}(\mathsf{v}_i)$:

$\mathsf{ballot}_i, \mathsf{accList}_i, \mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}}, \pi_{\mathsf{decide}} = \perp$, $\Psi_i = \emptyset$

1: **for** phase $r = 1$ to n **do**
2: leader $\leftarrow p_r \mod n$

(prepare)
Round 1:
leader:
3: **if** $\mathsf{ballot}_i = \perp$ **then** SEND$([n], \langle \mathsf{value_req} \rangle_{\sigma_r})$
replica:
4: **if** received $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{ballot}_{\mathsf{commit}} = \perp$ **then** SEND$(p_r, \langle \mathsf{v}_i, r \rangle_{\sigma_i})$
5: **else if** received $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{ballot}_{\mathsf{commit}} \neq \perp$ **then**
6: SEND$(p_r, \langle \mathsf{ballot}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \rangle_{\sigma_i})$

(pre-commit)
Round 2:

leader:

if received messages $\langle c_j \rangle_{\sigma_j}$, s.t $\left\{ \begin{array}{c} c_j = \langle \mathsf{ballot}_j, k, QC_{\mathsf{commit}} \rangle \textbf{ and} \\ \mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}_j, k \rangle, QC_{\mathsf{commit}}) = 1 \end{array} \right\}$ **then**

  $\mathbf{c} = \arg\max_{c_j} \{k\}$

  $\textsc{send}([N], (\langle \mathbf{c}.\mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathbf{c}.k, \mathbf{c}.QC_{\mathsf{commit}} \rangle_{\sigma_r}))$

else if received at least $\max\{n-r+1, n-t\}$ messages $\langle m_j, r \rangle_{\sigma_j}$, $m_j \in \{0,1\}$ **then**

  $P_0 = \{\mathsf{pk}_j\}_{j:m_j=0}$, $P_1 = \{\mathsf{pk}_j\}_{j:m_j=1}$, $\mathsf{accList} = N - (P_0 \cup P_1))$

  $\mathsf{w} = (P_0, P_1, \{\langle 0, r \rangle_{\sigma_j}\}_{j \in P_0}, \{\langle 1, r \rangle_{\sigma_j}\}_{j \in P_1})$

  $\mathsf{x} = (r, |P_0|, |P_1|, \mathsf{Hash}(\mathsf{accList})))$

  $\pi_{\mathsf{ballot}} = \mathsf{SNRK}_1.\mathsf{Prove}(\mathsf{crs}_1, \mathsf{x}, \mathsf{w})$

  $\mathsf{ballot} = \langle r, |P_0|, |P_1|, \mathsf{Hash}(\mathsf{accList}), \pi_{\mathsf{ballot}} \rangle$

  $\textsc{send}([N], (\langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r}))$

replica:

if received exactly one message M from the leader **then**

  if $M = \langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle k, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **then**

    if $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, k \rangle, QC_{\mathsf{commit}}) = 1$ **and** $k \geq \mathsf{rank}_{\mathsf{commit}}$ **then**

      $\textsc{send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{ballot}, r \rangle_{\sigma_r})$

  else if $M = \langle \mathsf{ballot}, r \rangle_{\sigma_r}, \langle \mathsf{accList} \rangle_{\sigma_r} \rangle$, where $\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$ **then**

    if $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, (r, \mathsf{count}_0, \mathsf{count}_1, H_A), \pi_{\mathsf{ballot}}) = 1$ **and** $pk_i \notin \mathsf{accList}$ **and** $\mathsf{Hash}(\mathsf{accList}) = H_A$ **and** $|\mathsf{accList}| \leq \min\{r-1, t\}$ **then**

      $\Psi_i[\mathsf{ballot}] = \langle \mathsf{accList} \rangle_{\sigma_r}$

      $\textsc{send}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{ballot}, r \rangle_{\sigma_r})$

Round 3:

replica:

if sent in (round 2) **and** not received $\langle \mathsf{ballot}', r \rangle_{\sigma_r}$, s.t. $\mathsf{ballot}' \neq \mathsf{ballot}$ **then**

  $\textsc{send}(p_r, \langle \mathsf{ballot}, r \rangle_{\sigma_i})$

(commit)

Round 4:

leader:

if received set S of size $s \geq n - t$ many signatures on $\langle \mathsf{ballot}, r \rangle$ **then**

  $QC_{\mathsf{commit}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle \mathsf{ballot}, r \rangle, S)$

  $\textsc{send}([N], \langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r})$

replica:

if received $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**

  $\mathtt{temp\_com\_cert} = \langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$

  $\textsc{send}(\mathcal{G}_{n,\epsilon}, \langle \mathtt{temp\_com\_cert} \rangle_{\sigma_r})$

Round 5:

replica:

if $\mathtt{temp\_com\_cert} \neq \perp$ **then**

  $\mathsf{ballot}_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.\mathsf{ballot}$, $\mathsf{rank}_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.r$,

  $\pi_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.QC_{\mathsf{commit}}$

  $\textsc{send}(p_r, \langle decide, \mathsf{ballot}_{\mathsf{commit}}, r \rangle_{\sigma_i})$

else if received $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**

  $\mathsf{ballot}_{\mathsf{commit}} = \mathsf{ballot}$, $\mathsf{rank}_{\mathsf{commit}} = r$, $\pi_{\mathsf{commit}} = QC_{\mathsf{commit}}$

(decide)

Round 6:

leader:

if received set DS of size $s \geq n - t$ many signatures on $\langle decide, \mathsf{ballot}, r \rangle$ **then**

  $QC_{\mathsf{decide}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle decide, \mathsf{ballot}, r \rangle, DS)$

  $\textsc{send}([N], \langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r})$

replica:

if received $\langle decide, \mathsf{ballot}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle decide, \mathsf{ballot}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**

  $\mathsf{ballot}_i = \mathsf{ballot}, \pi_{\mathsf{decide}} = r, QC_{\mathsf{decide}}$

  if $\Psi_i[\mathsf{ballot}_i] \neq \emptyset$ **then** $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i].\mathsf{accList}$

(Processing)
Round 1:
46: **if** $\mathsf{ballot}_i = \perp$ **then**
47: SEND$([N], \langle help \rangle_{\sigma_i})$
Round 2:
48: **for** all $\langle help \rangle_{\sigma_j}$ messages received **do**
49: SEND$(p_j, \langle \mathsf{ballot}_i, \pi_{\mathsf{decide}} \rangle_{\sigma_i})$
Round 3:
50: **if** $\mathsf{ballot}_i = \perp$ and received $\langle decide, \mathsf{ballot}_j, r, QC_{\mathsf{decide}} \rangle_{\sigma_j}$ s.t $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{ballot}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**
51: $\mathsf{ballot}_i = \mathsf{ballot}_j$
52: **if** $\Psi_i[\mathsf{ballot}_i] \neq \emptyset$ **then** $\mathsf{accList}_i = \Psi_i[\mathsf{ballot}_i]$
return $(\mathsf{ballot}_i, \mathsf{accList}_i)$

Figure 4: Code of Π_{Polling} for party p_i.

3.2 Rebuttal. The objective of Π_{Rebuttal} (Figure 5) is to validate the correctness of the initial tallying process and that no initial values belonging to some parties were unjustly omitted. In Π_{Rebuttal}, parties invoke it with input $(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$, where ballot_i and $\mathsf{accList}_i$ represent the output of Π_{Polling}. If, in the Polling protocol, the malicious ballot creator fails to include the initial values of all honest parties, the honest parties possessing non-empty accusation lists ($\mathsf{accList} \neq \perp$) will send accusing messages to each party within $\mathsf{accList}$. In response, upon receiving an accusing message, the honest parties send their initial values to all the other parties. Consequently, honest parties can observe whether the total number of initial values $\mathsf{v}_i = (\mathsf{v}, r)$, where $r \in \mathsf{ballot}$, received during the Rebuttal phase, in addition to $\mathsf{count}_{\mathsf{v}}$ in the agreed ballot, adds to $n-t$ for a $\mathsf{v} \in \{0, 1\}$. If this condition is met, the parties output $\mathsf{y_i} = \mathsf{v}$, and aux_i, which is a SNARK proof π_{v} confirming the existence of $n-t$ votes on v, r during the run. The creation of aux_i involves the use of π_{ballot}, which confirms the existence of $\mathsf{count}_{\mathsf{v}}$ valid signatures, and the additional signatures received during rebuttal. Consequently, we define $\mathsf{ThresholdPredicate}(y_i, \mathsf{aux}_i) = \mathsf{true}$, if aux_i is a valid proof of existence of $n-t$ signatures on v, r. By ensuring the prevention of double counting, we can guarantee that even if accused malicious parties act maliciously and send contradicting initial values to parties, it will not compromise validity. If an honest party receives $n-t$ distinct signatures on the same value, then the signature of at least one honest party is included. If all honest parties start with the same value v, the adversary cannot generate $n-t > t$ signatures on value v', r except with negligible probability for forging signatures. This achievement aligns with the primary goal of Reliable voting, ensuring validity for BA.

Protocol Overview. The algorithm is comprised of three rounds. During the first round, all honest parties possessing a non-null $\mathsf{accList}$ send an accusing message $\langle pk_j, \pi_{j,A} \rangle_{\sigma_r}$ to the accused parties $\in \mathsf{accList}$. Here, $\pi_{j,A}$ denotes a SNARK proof constructed by the accusing parties, proving that $pk_j \in \mathsf{accList}$ to parties that might not have knowledge of $\mathsf{accList}$. Let SNRK_2 be a SNARK system as defined in Definition 2.4. An accusing party constructs $\pi_{j,A}$ for the public statement:

$$\mathsf{x} = (pk_j, H_A)\,,$$

where pk_j is the public key for some party p_j and H_A is the hash of some set. The above public NP statement is verified using the following witness:

$$\mathsf{w} = (\mathsf{accList})\,,$$

where $\mathsf{accList}$ is a set of public keys. The verification computation $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, \mathsf{x}, \pi_{j,A})$ outputs "1" if and only if $H_A = \mathsf{Hash}(\mathsf{accList})$ and $pk_j \in \mathsf{accList}$.
In round 2, parties who receive an accusing message that contains a valid $\pi_{i,A}$, forward their initial value $\langle \mathsf{v}_i, r \rangle_{\sigma_j}$ and the accusing message to all parties. Finally, during round 3, each honest party verifies that each message $\langle \mathsf{v}, r \rangle_{\sigma_j}$, where $r \in \mathsf{ballot}$, it received is accompanied by a valid SNARK proof $\pi_{j,A}$, proving that $p_j \in \mathsf{accList}$. If the messages are deemed valid, the honest party tallies the votes of all accused parties. Specifically, if the sum of (1) the total count of received signatures on v, r and (2) the $\mathsf{count}_{\mathsf{v}} \in \mathsf{ballot}$ is at least $n-t$ for $\mathsf{v} \in \{0, 1\}$, then the party constructs a proof π_{v} using SNRK_3, a SNARK system as per Definition 2.4, indicating the existence of

at least $n-t$ votes on v, r. The party constructs the proof π_{v} for the public statement:

$$\mathsf{x} = (\mathsf{v})\,,$$

This public NP statement is verified using the following witness:

$$\mathsf{w} = (\mathsf{votes}_{\mathsf{v}}, \mathsf{ballot})\,,$$

where $\mathsf{votes}_{\mathsf{v}}$ is a set of messages. The verification computation $\mathsf{SNRK}_3.\mathsf{Verify}(\mathsf{crs}_3, \mathsf{x}, \pi_{\mathsf{v}})$ outputs "1" if and only if the following checks are true:

- $\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$, such that $\mathsf{SNRK}_1.\mathsf{Verify}(\mathsf{crs}_1, \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A \rangle, \pi_{\mathsf{ballot}}) = 1$,
- each $m \in \mathsf{votes}_{\mathsf{v}}$ is of the form $\langle \langle \mathsf{v}, r \rangle_{\sigma_j}, \langle pk_j, \pi_{j,A} \rangle \rangle$,
- for each $m \in \mathsf{votes}_{\mathsf{v}}$, the signature for $\langle \mathsf{v}, r \rangle$ is valid with respect to pk_j,
- for each $m \in \mathsf{votes}_{\mathsf{v}}$: $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, \langle pk_j, H_A \rangle, \pi_{j,A}) = 1$
- $\mathsf{count}_{\mathsf{v}} + |\mathsf{votes}_{\mathsf{v}}| \geq n - t$

Finally, the honest party sets $\mathsf{y_i} = \mathsf{v}$ and $\mathsf{aux}_i = \pi_{\mathsf{v}}$. By the end of Rebuttal, all honest parties are assured of receiving the votes of every honest party in $\mathsf{accList}$ that were not included in ballot.

Protocol Π_{Rebuttal} achieves the properties stated in the following theorem that we prove in Appendix A.

THEOREM 3.3. (REBUTTAL) *Assume an execution of protocol Π_{Rebuttal} (Figure 5). Each party p_i starts with input $(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$, where v_i is the input to Π_{Polling} and $\mathsf{ballot}_i = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$ and $\mathsf{accList}_i$ are the outputs of Π_{Polling}. Each party outputs value $\mathsf{y}_i \in \{0,1\}$ and auxiliary information aux_i. Then, Π_{Rebuttal} satisfies the following properties, except with negligible probability $\mathsf{negl}(\lambda)$:*

- *Provable validity: If all honest parties start with $\mathsf{v}_i = \mathsf{v}$, then, all honest parties output $\mathsf{y}_i = \mathsf{v}$, and aux_i such that $\mathsf{ThresholdPredicate}(y_i, \mathsf{aux}_i)$ outputs true. Moreover, no party outputs $\mathsf{y}_j \neq \mathsf{y}_i$ and auxiliary information aux_j such that $\mathsf{ThresholdPredicate}(\mathsf{y}_j, \mathsf{aux}_j)$ outputs true.*
- *Termination: All honest parties terminate.*

Further, we prove that Π_{Rebuttal} has $O(n \cdot f)$ communication complexity.

THEOREM 3.4. *Π_{Rebuttal} has $O(n \cdot f)$ communication complexity.*

Proof. The protocol involves three communication rounds, with messages exchanged between all parties and the accused parties. By Lemma 4, the number of accused parties is less than $2f$. Consequently, the communication complexity of the rebuttal stage is $O(n \cdot f)$. □

Finally, we demonstrate that the security properties of Π_{Polling} and Π_{Rebuttal} when combined, result in a secure Reliable Voting protocol, Π_{RV}.

THEOREM 3.5. (RELIABLE VOTING FROM POLLING AND REBUTTAL) *Let Π_{Polling} be the protocol of Figure 4 and Π_{Rebuttal} be the protocol of Figure 5. Π_{RV} of Figure 3 achieves Reliable Voting per Definition 2.2, for $t < (1/2 - \epsilon)n$ with respect to ThresholdPredicate, incurring a communication complexity of $O(n \cdot f)$.*

Proof. (Termination) From assumption, Π_{Polling} and Π_{Rebuttal} terminate as per Theorem 3.1 and Theorem 3.3, thereby ensuring the termination of Π_{RV}.
(Provable Validity) Assume that all honest parties start with the same value v. Then, Π_{Rebuttal} achieves provable validity on v and aux_i per Theorem 3.3. Since these are the outputs of Π_{RV}, the property is achieved trivially.
(Communication) Finally, our proposed approach involves invoking each protocol, namely Π_{Polling} and Π_{Rebuttal}, only once in Π_{RV}. Given our assumptions, Π_{RV} incurs a communication complexity of $O(n \cdot f)$. □

Algorithm $\Pi_{\mathsf{Rebuttal}}(\mathsf{v}_i, \mathsf{ballot}_i = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}}\rangle, \mathsf{accList}_i)$:

Round 1:

$\mathsf{y_i} \leftarrow \mathsf{v}_i, \mathsf{aux}_i \leftarrow \perp$

1: **for** each $\mathsf{pk}_j \in \mathsf{accList}_i$ **do**
2: $\quad \pi_{j,A} = \mathsf{SNRK}_2.\mathsf{Prove}(\mathsf{crs}_2, (\mathsf{pk}_j, H_A), \mathsf{accList}_i))$
3: $\quad \textsc{send}(p_j, \langle \pi_{j,A} \rangle_{\sigma_i})$

Round 2:

4: **if** received message $\langle \pi_{i,A} \rangle_{\sigma_j}$ **and** $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, (\mathsf{pk}_i, H_A), \pi_{i,A}) = 1$ **then**
5: $\quad \textsc{send}([N], (\langle \mathsf{v}_i, r \rangle_{\sigma_i}, \langle \pi_{i,A} \rangle_{\sigma_i}))$

Round 3:

6: **for** each $\mathsf{v} \in \{0, 1\}$ **do**
7: $\quad$ **for** all $(\langle \mathsf{v}, r \rangle_{\sigma_j}, \langle \pi_{j,A} \rangle_{\sigma_j})$ received s.t. $\mathsf{SNRK}_2.\mathsf{Verify}(\mathsf{crs}_2, (\mathsf{pk}_j, H_A), \pi_{j,A}) = 1$ **do**
8: $\quad\quad \mathsf{votes_v} = \mathsf{votes_v} \cup \{(\langle \mathsf{v}, r \rangle_{\sigma_j}, \langle \pi_{j,A} \rangle_{\sigma_j})\}$
9: $\quad$ **if** $|\mathsf{votes_v}| + \mathsf{count_v} \geq n - t$ **then**
10: $\quad\quad \pi_{\mathsf{v}} = \mathsf{SNRK}_3.\mathsf{Prove}(\mathsf{crs}_3, (\mathsf{v}), (\mathsf{votes_v}, \mathsf{ballot}_i))$
11: $\quad\quad \mathsf{y_i} = \mathsf{v}, \mathsf{aux}_i = \pi_{\mathsf{v}}$

return $\mathsf{y_i}, \mathsf{aux}_i$

Figure 5: Code of Π_{Rebuttal} for party p_i.

4 Weak Byzantine Agreement (WBA)

Our WBA protocol, Π_{WBA} (Figure 6), is leader-based and built upon the silent phases paradigm. Our starting point is the WBA protocol in [15], which achieves agreement, termination, and unique validity. However, we achieve a different validity property, the so-called restricted validity. In Π_{WBA}, each party begins with input $(\mathsf{v}_i, \mathsf{aux}_i)$, where the auxiliary information aux_i assists the party in deciding whether to support a leader's proposed value or not. Similar to RV, Our Π_{WBA} protocol is instantiated with respect to the Boolean predicate $\mathsf{ThresholdPredicate}$. Thus, if an honest party starts with $(\mathsf{v}_i, \mathsf{aux}_i)$ where $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$, they are more likely to refrain from voting if the leader proposes a conflicting value v_r unless the leader can prove that $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$. For modularity, we define a predicate $\mathsf{SafeVal}$ that checks if the value proposed by the leader p_r is safe to accept with respect to $\mathsf{v}_i, \mathsf{aux}_i$ with which a party p_i starts. In other words, $\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ is true if $\mathsf{v}_{\mathsf{commit}} = \perp$, where $\mathsf{v}_{\mathsf{commit}}$ is a value that the party is committed to, and either $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ or, $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$.

$\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ is true if $\mathsf{v}_{\mathsf{commit}} = \perp$ **and**

- $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{false}$ **or**,
- $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{true}$

It should be noted that our protocol employs the use of expanders, which results in suboptimal resilience of $t < (\frac{1}{2} - \epsilon)n$, where ϵ is a positive constant. Expanders are utilized to ensure that the leader does not generate conflicting commit certificates within a given phase. In other words, parties only vote for a value after confirming that they have not received any contradictory messages from their neighbors in the expander. Although expanders do lower the resilience of the protocol, it is guaranteed that all parties are decided by the end of the n phases, eliminating the need for fallback algorithms, which is utilized in [15], as it is not guaranteed that all parties are decided by the end of the n phases. This is because, in the latter protocol, the leader has to collect $\lceil \frac{n+t+1}{2} \rceil$ signatures to form valid commit and decide certificates. Therefore, if $f > \frac{t}{2}$, it is possible that all honest leaders fail to collect the required $\lceil \frac{n+t+1}{2} \rceil$ signatures. Using expanders as in Π_{Polling}, we can lower the signature threshold to $n - t$, which guarantees that an honest leader can obtain enough votes to form valid commit and decide certificates, ensuring that all honest parties are decided by the end of the leader's phase. This process guarantees safety equivalent to that provided by the $\lceil \frac{n+t+1}{2} \rceil$ threshold, whereby a leader cannot generate two conflicting commit certificates without an honest party voting for both. Notably, since only $n - t$ signatures are required, an honest leader is guaranteed to collect enough votes to form valid commit and decide certificates, leading to all

honest parties being decided by the end of the leader's phase. Therefore, we achieve a better round complexity than in [15]. In Lemma 5, we prove that all honest parties will be decided within $f+1$ non-silent phases.

Protocol Overview. Π_{WBA} runs for n phases. Each phase comprises four rounds; prepare, pre-commit, commit, and decide.

(prepare) If a leader is undecided, they ask for all parties' values by multicasting a request message $\langle \mathsf{value\_req} \rangle$. If a committed honest party receives the request message, they forward their commit value $\mathsf{v}_{\mathsf{commit}}$ along with a proof π_{commit} and a commit rank to the leader. If they are not committed, they forward their input $(\mathsf{v}_i, \mathsf{aux}_i)$ to the leader, but only if $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$.

(precommit) The leader surveys the messages sent during the *prepare* round and selects an optimal message that all honest parties will vote for. If the leader receives commit messages, the leader selects the one with the highest commit rank and proposes it along with the commit proof and rank to the parties. If the leader does not receive any commit message, the leader chooses a value that satisfies the $\mathsf{SafeVal}$ conditions. If none of these conditions is met, the leader proposes initial value $(\mathsf{v}_i, \mathsf{aux}_i)$. Honest parties sign the leader's proposal if they receive a commit message with a higher rank than the one to which they committed or if the value proposed is safe, as determined by the $\mathsf{SafeVal}$ function. Before sending the vote to the leader, the party propagates the leader's proposal through the expander $\mathcal{G}_{n,\epsilon}$ to ensure that the leader did not send a conflicting value.

(commit) If the leader receives $n-t$ votes (signatures) on the leader's proposal, the leader combines them to form a valid commit proof QC_{commit}. The leader then forwards the commit certificate $\langle \mathsf{commit\_value}, r, QC_{\mathsf{commit}} \rangle$ to all parties. Upon receiving a valid commit certificate, a party sends a matching decide message to the leader and propagates the commit message through the expander.

(decide) Once the leader collects enough $(\geq n-t)$ decide messages, the leader forms a valid decide proof QC_{decide} and decide certificate for v_r, which is multicast to all parties to decide on.

LEMMA 5. *All honest parties are decided within $f+1$ phases.*

Proof. Each undecided honest leader p_r initiates communication during designated phase r by sending $\langle \mathsf{value\_req} \rangle$ to all parties. If a party has $\mathsf{v}_{\mathsf{commit}} \neq \perp$, they send their $\mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}$, and π_{commit} to the leader. Otherwise, if an honest party p_j has $\mathsf{v}_{\mathsf{commit}} = \perp$ and $\mathsf{ThresholdPredicate}(\mathsf{v}_j, \mathsf{aux}_j) = \mathsf{true}$, it sends $(\mathsf{v}_j, \mathsf{aux}_j)$ to p_r. As a result, p_r proposes the highest rank commit certificate if they receive one, ensuring that all committed parties vote for it (Line 16), or a value v, where $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$, ensuring that all honest parties vote for it if they are not committed. Otherwise, if the leader does not receive either, they propose their v_i. After receiving a proposed value, each party propagates it to their neighbors to confirm that there are no conflicting values proposed. Since the leader is honest and does not send conflicting value, all honest parties send a matching vote message $\langle \mathsf{v}, r \rangle_{\sigma_i}$, allowing p_r to collect enough $n-t$ votes to form a valid commit certificate, which they forward to all parties. Subsequently, the parties send a matching decide message once they receive a valid commit certificate. This allows p_r to collect enough $n-t$ decide messages to form a valid decide certificate, which they again forward to all parties. Once an honest party sees a valid decide certificate on v, they set $\mathsf{y_i} = \mathsf{v}$ and is decided henceforth. In subsequent phases, honest leaders remain silent and no communication occurs. Since there are f malicious leaders on the run, all honest parties are decided within $f+1$ phases. □

Algorithm $\Pi_{\mathsf{WBA}}(\mathsf{v}_i, \mathsf{aux}_i)$:

$\mathsf{y}_i, \mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \leftarrow \perp$

1: **for** phase $r = 1$ to n **do**
2: leader $\leftarrow p_r \mod n$
(prepare)
Round 1:
leader:
3: **if** $\mathsf{y}_i = \perp$ **then**
4: SEND$([n], \langle \mathsf{value_req} \rangle_{\sigma_r})$
replica:
5: **if** received $\langle \mathsf{value\_req} \rangle_{\sigma_r}$ **and** $\mathsf{v}_{\mathsf{commit}} \neq \perp$ **then**
6: SEND$(p_r, \langle \mathsf{v}_{\mathsf{commit}}, \mathsf{rank}_{\mathsf{commit}}, \pi_{\mathsf{commit}} \rangle_{\sigma_i})$

7: **else if** received $\langle \mathsf{value\_req} \rangle_{\sigma_i}$ **and** $\mathsf{v}_{\mathsf{commit}} = \perp$ **and** $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$ **then**
8: $\text{SEND}(p_r, \langle \mathsf{v}_i, \mathsf{aux}_i \rangle)$

(pre-commit)
Round 2:
leader:
9: **if** received messages $\langle c_j \rangle_{\sigma_j}$, s.t $c_j = \langle \mathsf{v}_j, k, QC_{\mathsf{commit}} \rangle$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}_j, k \rangle, QC_{\mathsf{commit}}) = 1$ **then**
10: $\mathbf{c} = \arg\max_{c_j} \{k\}$
11: $\text{SEND}([n], \langle \mathbf{c}.\mathsf{v}, r \rangle_{\sigma_r}, \langle \mathbf{c}.k, \mathbf{c}.QC_{\mathsf{commit}} \rangle_{\sigma_r}))$
12: **else if** $\mathsf{ThresholdPredicate}(\mathsf{v}_i, \mathsf{aux}_i) = \mathsf{true}$ **or** received $\langle \mathsf{v}_j, \mathsf{aux}_j \rangle_{\sigma_j}$, where $\mathsf{ThresholdPredicate}(\mathsf{v}_j, \mathsf{aux}_j) = \mathsf{true}$ **then**
13: pick any and $\text{SEND}([n], (\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle \mathsf{aux}, r \rangle_{\sigma_r}))$
14: **else** $\text{SEND}([n], (\langle \mathsf{v}_i, r \rangle_{\sigma_r}), \langle \mathsf{aux}_i, r \rangle_{\sigma_r})$

replica:
15: **if** received exactly one message M **then**
16: **if** M is of the form $\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle k, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **then**
17: **if** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, k \rangle, QC_{\mathsf{commit}}) = 1$ **and** $k \geq \mathsf{rank}_{\mathsf{commit}}$ **then**
18: $\text{SEND}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{v}, r \rangle_{\sigma_r})$
19: **else if** M is of the form $\langle \mathsf{v}, r \rangle_{\sigma_r}, \langle \mathsf{aux}, r \rangle_{\sigma_r}$ **and** $\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}_i, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}})$ **then**
20: $\text{SEND}(\mathcal{G}_{n,\epsilon}, \langle \mathsf{v}, r \rangle_{\sigma_r})$

(Round 3):
replica:
21: **if** sent in (round 2) and not received $\langle \mathsf{v}', r \rangle_{\sigma_r}$ s.t. $\mathsf{v}' \neq \mathsf{v}$ **then**
22: $\text{SEND}(p_r, \langle \mathsf{v}, r \rangle_{\sigma_i})$

(commit)
Round 4:
leader:
23: **if** received set S of size $s \geq n - t$ many signatures on $\langle \mathsf{v}, r \rangle$ **then**
24: $QC_{\mathsf{commit}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle \mathsf{v}, r \rangle, S)$
25: $\text{SEND}([n], \langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_i})$

replica:
26: **if** received $\langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
27: $\mathtt{temp\_com\_cert} = \langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle$
28: $\text{SEND}(\mathcal{G}_{n,\epsilon}, \langle \mathtt{temp\_com\_cert} \rangle_{\sigma_r})$

Round 5:
replica:
29: **if** $\mathtt{temp\_com\_cert} \neq \perp$ **then**
30: $\mathsf{v}_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.\mathsf{v}, \mathsf{rank}_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.r$, $\pi_{\mathsf{commit}} = \mathtt{temp\_com\_cert}.QC_{\mathsf{commit}}$
31: $\text{SEND}(p_r, \langle decide, \mathsf{v}, r \rangle_{\sigma_i})$
32: **if** received $\langle \mathsf{v}, r, QC_{\mathsf{commit}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle \mathsf{v}, r \rangle, QC_{\mathsf{commit}}) = 1$ **then**
33: $\mathsf{v}_{\mathsf{commit}} = \mathsf{v}, \mathsf{rank}_{\mathsf{commit}\,i} = r$, $\pi_{\mathsf{commit}} = QC_{\mathsf{commit}}$

(decide)
Round 6:
leader:
34: **if** received set DS of size $s \geq n - t$ many signatures on $\langle decide, \mathsf{v}, r \rangle$ **then**
35: $QC_{\mathsf{decide}} = \mathsf{SS.Combine}(\{\mathsf{pk}_{i_1}, \ldots, \mathsf{pk}_{i_s}\}, \langle decide, \mathsf{v}, r \rangle, DS)$
36: $\text{SEND}([n], \langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle_{\sigma_i})$

replica:
37: **if** received $\langle decide, \mathsf{v}, r, QC_{\mathsf{decide}} \rangle_{\sigma_r}$ **and** $\mathsf{SS.Verify}(\mathsf{pk}_{ss}, \langle decide, \mathsf{v}, r \rangle, QC_{\mathsf{decide}}) = 1$ **then**
38: $\mathsf{y_i} = \mathsf{v}$

return $\mathsf{y_i}$

Figure 6: Code of Π_{WBA} for party p_i.

We state the theorem for the correctness of Π_{WBA}, which we prove in Appendix A.

THEOREM 4.1. *Our protocol* Π_{WBA} *(Figure 6), instantiated with* ThresholdPredicate*, achieves Weak Byzantine Agreement per Definition 2.3 for* $t < (1/2 - \epsilon)n$.

Finally, we prove the communication complexity of Π_{WBA}.

THEOREM 4.2. Π_{WBA} *has* $O(n \cdot f)$ *communication complexity.*

Proof. Each phase requires constantly many steps of communication between the leader and all parties back and forth. Each message transmitted during this communication is of size $O(1)$, meaning that each phase incurs a communication complexity of $O(n)$. According to Lemma 5, every honest party is decided within $f + 1$ phases. Once an honest party is decided, it stays silent in its leader phase and does not incur any communication. Thus, the total communication of the protocol is $O(n \cdot f)$. □

5 Conclusion and Future Directions

In this paper, we study the communication complexity of deterministic binary Byzantine agreement, which is crucial for efficient large-scale applications. We introduce a practical construction with communication complexity of $O(n \cdot f)$, where f is the exact number of malicious parties. This constitutes a major improvement over the quadratic $O(n^2)$ communication bound achieved by previous work [29], since for executions where $f = o(n)$, our protocol achieves $o(n^2)$ communication. This sets a precedence on ways to circumvent the theoretical lower bounds on communication exchange for BA. However, several directions remain unanswered.

Extending to multivalued settings. Our protocol can be trivially extended to achieve BA in multivalued settings. For L-bit input values, parties can run L (even parallel) executions of our binary BA protocol. We notice that the communication of this solution is $O(n \cdot f \cdot L)$. Since L is the dominating factor in many practical applications, such communication could be prohibitive. Therefore extending the result to multivalued BA, where the values' size i) is not part of the word size and ii) is not a multiplicative factor in the $O(n \cdot f)$ communication complexity, is an interesting future direction.

Strong BA. In the related literature, the term BA is sometimes conflated with *Strong* BA, where a strictly stronger validity property (alas, strong validity) is achieved, namely that the output of honest parties is always the input of some honest party. Confusion between the two terms might stem from the fact that in the binary case, BA and Strong BA are equivalent; Since the input domain is 2, if honest parties start with different values, then their inputs cover the entire domain, and strong validity is guaranteed from agreement. However, in the multivalued setting, this property is much more difficult to achieve. This is also why we refrained from describing our result as achieving binary Strong BA, even though it does.

Achieving Strong Byzantine Agreement (Strong BA) in the multivalued setting is challenging, as the resilience threshold is inversely related to the input domain size. As shown in an earlier work by [22], let a value domain of size m. Then, assume the case where all honest parties start with $m - 1$ different inputs, each chosen evenly by $\frac{n-t}{m-1}$ honest parties. Then, t malicious parties can claim their input to be the value in the domain not input by any honest party. Other than that, malicious parties act honestly throughout the protocol. No protocol could distinguish this case from a case where simply all parties are honest and t start with that value. So, for a protocol to achieve Strong BA, it must tolerate $t < \frac{n-t}{m-1}$, i.e. $t < n/m$ malicious parties. Interestingly notice that for $m = 2$, we naturally get the $t < n/2$ optimal resilience of binary (Strong) BA. Exploring the optimality of protocols' communication for multivalued Strong BA is thus an open question, with exciting practical applications.

Other Directions. Since we achieve near-optimal resilience $t < (1/2-\epsilon)n$ due to the use of expanders, achieving deterministic Byzantine Agreement with optimal resilience $t < n/2$ and $O(n \cdot f)$ communication remains an open problem. Another direction that is worth exploring is how to simplify the protocol for the reliable voting. A modern approach that could be promising would be to combine silent phases with *gossiping* and *batching* techniques (e.g. [4, 33, 5]) to achieve randomized BA or parallel broadcast protocols with improved, adaptive communication compared to the current best.

Acknowledgments

We would like to thank Julian Loss for valuable discussions as well as the SODA reviewers for their feedback. This work was supported by the National Science Foundation, VMware and Protocol Labs.

References

[1] Abraham, I., Aguilera, M. & Malkhi, D. Fast Asynchronous Consensus with Optimal Resilience. *24th International Symposium On Distributed Computing (DISC 2010)*.

[2] Abraham, I., Chan, T., Dolev, D., Nayak, K., Pass, R., Ren, L. & Shi, E. Communication complexity of byzantine agreement, revisited. *Proceedings Of The 2019 ACM Symposium On Principles Of Distributed Computing*.

[3] Abraham, I. & Dolev, D. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. *Proceedings Of The Forty-seventh Annual ACM Symposium On Theory Of Computing*. pp. 605-614 (2015)

[4] Abraham, I., Malkhi, D. & Spiegelman, A. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. (2019)

[5] Abraham, I., Nayak, K., Shrestha, N. Communication and Round Efficient Parallel Broadcast Protocols. (Cryptology ePrint Archive, 2023), https://eprint.iacr.org/2023/1172

[6] Baudet, M., Ching, A., Chursin, A., Danezis, G., Garillot, F., Li, Z., Malkhi, D., Naor, O., Perelman, D. & Sonnino, A. State Machine Replication in the Libra Blockchain. *The Diem Association*. (2019)

[7] Ben-Sasson, E., Bentov, I., Horesh, Y. & Riabzev, M. Scalable, transparent, and post-quantum secure computational integrity. *IACR Cryptol. EPrint Arch.*. **2018** , https://api.semanticscholar.org/CorpusID:44557939

[8] Berman, P., Garay, J. & Perry, K. Bit optimal distributed consensus. *Computer Science: research and applications*. pp. 313-321 (1992)

[9] Blum, E., Katz, J., Liu-Zhang, C. & Loss, J. Asynchronous Byzantine Agreement with Subquadratic Communication. *Theory Of Cryptography: 18th International Conference, TCC 2020, USA, November 16–19, 2020, Proceedings, Part I 18*. pp. 353-380 (2020)

[10] Boldyreva, A. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. *Public Key Cryptography - PKC 2003*. pp. 31-46 (2003,1)

[11] Boneh, D., Gentry, C., Lynn, B. & Shacham, H. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. *Advances in Cryptology—EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003*.

[12] Boneh, D., Lynn, B. & Shacham, H. Short Signatures from the Weil Pairing. *J. Cryptol.*. **17**, 297-319 (2004,9), https://doi.org/10.1007/s00145-004-0314-9

[13] Cachin, C., Kursawe, K., Petzold, F. & Shoup, V. Secure and Efficient Asynchronous Broadcast Protocols. *Proceedings Of The 21st Annual International Cryptology Conference On Advances In Cryptology*. pp. 524-541 (2001)

[14] Civit, P., Gilbert, S., Guerraoui, R., Komatovic, J. & Vidigueira, M. Strong Byzantine Agreement with Adaptive Word Complexity. https://arxiv.org/abs/2308.03524

[15] Cohen, S., Keidar, I. & Spiegelman, A. Make Every Word Count: Adaptive Byzantine Agreement with Fewer Words. *26th International Conference On Principles Of Distributed Systems (OPODIS 2022)*. **253** (2023)

[16] Desmedt, Y. Society and Group Oriented Cryptography: A New Concept. *Annual International Cryptology Conference*. (1987)

[17] Dolev, D. & Lenzen, C. Early-Deciding Consensus is Expensive. *Proceedings Of The 2013 ACM Symposium On Principles Of Distributed Computing*. pp. 270-279 (2013),

[18] Dolev, D. & Reischuk, R. Bounds on Information Exchange for Byzantine Agreement. *J. ACM*. **32**, 191-204 (1985)

[19] Dolev, D., Reischuk, R. & Strong, H. Early Stopping in Byzantine Agreement. *J. ACM*. **37**, 720-741 (1990)

[20] Dolev, D. & Strong, H. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.*. **12** pp. 656-666 (1983)

[21] Fischer, M., Lynch, N. & Paterson, M. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*. **32**, 374-382 (1985)

[22] Fitzi, M. & Garay, J. Efficient Player-Optimal Protocols for Strong and Differential Consensus. *Proceedings Of The Twenty-Second Annual Symposium On Principles Of Distributed Computing*. pp. 211-220 (2003)

[23] Gelashvili, R., Kokoris-Kogias, L., Sonnino, A., Spiegelman, A. & Xiang, Z. Jolteon and Ditto: Network-Adaptive Efficient Consensus with Asynchronous Fallback. *International Conference On Financial Cryptography And Data Security*. pp. 296-315 (2022)

[24] Groth, J. On the Size of Pairing-Based Non-interactive Arguments. *Advances In Cryptology–EUROCRYPT 2016: 35th Annual International Conference On The Theory And Applications Of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*

[25] King, V.Saia, J. Breaking the $O(n^2)$ Bit Barrier: Scalable Byzantine agreement with an Adaptive Adversary. *Journal Of The ACM (JACM)*. **58**

[26] Lamport, L. The Part-Time Parliament. *ACM Trans. Comput. Syst.*. **16**, 133-169 (1998)

[27] Lamport, L., Shostak, R. & Pease, M. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*. **4**, 382-401 (1982)

[28] Merkle, R. A digital signature based on a conventional encryption function. *Advances In Cryptology—CRYPTO'87: Proceedings 7*. pp. 369-378 (1988)

[29] Momose, A. & Ren, L. Optimal Communication Complexity of Authenticated Byzantine Agreement. (arXiv,2020), https://arxiv.org/abs/2007.13175
[30] Pease, M., Shostak, R. & Lamport, L. Reaching Agreement in the Presence of Faults. *J. ACM.* **27** pp. 228-234 (1980)
[31] Shoup, V. Practical Threshold Signatures. *International Conference On The Theory And Application Of Cryptographic Techniques.* (2000), https://api.semanticscholar.org/CorpusID:1179922
[32] Spiegelman, A. In Search for a Linear Byzantine Agreement. *CoRR.* **abs/2002.06993** (2020), https://arxiv.org/abs/2002.06993
[33] Tsimos, G., Loss, J., Papamanthou, C. Gossiping for Communication-Efficient Broadcast. *CRYPTO 2022.* (2022)
[34] Vassantlal, R., Alchieri, E., Ferreira, B. & Bessani, A. COBRA: Dynamic Proactive Secret Sharing for Confidential BFT Services. *2022 IEEE Symposium On Security And Privacy (SP).* pp. 1335-1353 (2022)
[35] Yin, M., Malkhi, D., Reiter, M., Gueta, G. & Abraham, I. HotStuff: BFT consensus with linearity and responsiveness. *Proceedings Of The 2019 ACM Symposium On Principles Of Distributed Computing*

A Correctness Proofs

In this Section we prove the correctness of our Polling, Rebuttal and WBA protocols. We start with proofs for Polling and show that Theorem 3.1 holds.

A.1 Polling Correctness Proof.

LEMMA 6. *If a leader creates a valid commit certificate C_1 in a phase, no valid conflicting certificate $C_1' \neq C_1$ is created in the same phase, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. For leader p_r in phase r to generate a valid commit certificate say C_1, they have to aggregate at least $n-t$ distinct valid signatures for the respective message $\langle \mathsf{ballot}, r\rangle$. Unless the leader forges signatures, which happens with probability $\mathsf{negl}(\lambda)$, the leader needs to receive the signatures from at least $n-t$ distinct parties. Since $t < (1/2-\epsilon)n$, then $n-t > t+2\epsilon n$, which means that more than $2\epsilon n$ of those senders are honest. These honest parties propagate ballot via the expander graph. By the expander's property (Definition 2.7), more than $(1-2\epsilon)n > 2t$ parties receive that specific ballot related to the commit certificate. Out of these parties, at least $t+1$ are honest. In Round 3, those honest parties do not sign any conflicting precommit message $\langle \mathsf{ballot}', r\rangle$, where $\mathsf{ballot}' \neq C_1.\mathsf{ballot}$. Thus, the leader cannot collect more than $n-(t+1)$ signatures on any such message $\langle \mathsf{ballot}', r\rangle$ and therefore cannot create a valid conflicting commit certificate, except with negligible probability. □

LEMMA 7. *During* Π_{Polling}, *there can be exactly one ballot* ballot^* *for which there exist valid decide certificates, except with negligible probability,* $\mathsf{negl}(\lambda)$.

Proof. First, we prove that there exists at least one ballot with valid decide certificates. By Lemma 2, at least one honest party is decided during Π_{Polling}. By construction, an honest party becomes decided on a ballot ballot during phase r, if it receives valid decide certificate on $\langle \mathit{decide}, \mathsf{ballot}, r\rangle$. Thus, there exists at least one ballot for which there exist valid decide certificates.

Now, we prove that there exists exactly one such ballot. Let ballot^* be the ballot for which there could exist a valid decide certificate on the earliest phase. More specifically, ballot^* is the ballot with the earliest phase r^*, for which at least $n-t$ distinct signatures on $\langle \mathit{decide}, \mathsf{ballot}^*, r^*\rangle$ were sent to the leader p_{r^*}. We prove that there cannot exist any other ballot ballot with valid decide certificates.

Let ballot have a valid decide certificate in some phase. Since ballot^* has a valid decide certificate for phase r^*, at least $n-t$ parties signed $\langle \mathit{decide}, \mathsf{ballot}^*, r^*\rangle$. Thus, at least $n-2t > 2\epsilon n$ honest parties signed that message. These honest parties, also sent $\langle \mathsf{ballot}^*, r^*, QC_{\mathsf{commit}}\rangle$ through the expander graph, meaning that at least $(1-2\epsilon)n > 2t$ parties received it. Out of all such parties, at most t were corrupted, so at least $t+1$ were honest. In round 5, these at least $t+1$ honest parties committed to ballot^*, and set their $\mathsf{rank}_{\mathsf{commit}}$ to r^*. In any subsequent phase $l > r^*$, these at least $t+1$ parties propose ballot^* to the leader p_l unless they receive a commit certificate C_l with a higher rank $l > r^*$. For a leader to create a commit certificate on ballot with a higher rank, they have to gather at least $n-t$ votes on ballot, unless the leader forges signatures, which can occur with negligible probability $\mathsf{negl}(\lambda)$. If the leader proposes ballot via a commit certificate that was created in phases $m < r^*$, those $t+1$ honest parties will not send their votes as they also see ballot^* with higher commit rank. Also, from Lemma 7 no valid conflicting commit certificate could have been created in phase r^*. If the leader proposes ballot with a valid commit certificate and a higher rank than ballot^*, then either the leader forges the necessary

signatures or breaks the SNRK_1 scheme, which happens with negligible probability $\mathsf{negl}(\lambda)$, or $\mathsf{ballot} = \mathsf{ballot}^*$. This is because, to form a new ballot the leader would need to gather initial values from at least $n-t$ parties in round 1. Since these at least $t+1$ honest parties have already committed to ballot^*, then they send ballot^* back to the leader during round 1, so the leader does not receive enough initial values for forming a new ballot. Thus, $\mathsf{ballot} = \mathsf{ballot}^*$, except with negligible probability $\mathsf{negl}(\lambda)$.
Thus, there exists exactly one ballot for which there exist valid decide certificates. □

LEMMA 8. Π_{Polling} *achieves Agreement according to Theorem 3.1, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. By Lemma 7, there exists exactly one ballot ballot^* for which there exist valid decide certificates. Also, by Lemma 2, at least $n-2f$ honest parties are decided before (*Processing*). During *Processing*, honest parties with $\mathsf{ballot}_i = \perp$ ask for help from all n parties. They receive back ballot^* along with valid decide proofs, and they will also thus output ballot^*. □

LEMMA 9. Π_{Polling} *achieves Correctness, Count validity, Reliability, and Integrity according to Theorem 3.1, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. (*Correctness*) Party p_i only sets its $\mathsf{ballot}_i = \mathsf{ballot}$ after receiving a valid decide certificate. To create such a certificate, at least $n-t$ parties must send a decide message for ballot in the commit certificate they received from the leader. To form a commit certificate $\langle \mathsf{ballot}, r, QC_{\mathsf{commit}} \rangle$, at least $n-t$ parties must verify and sign $\langle \mathsf{ballot}, r \rangle$, except with negligible probability $\mathsf{negl}(\lambda)$ that the leader will forge their votes. Thus, at least $n-t \geq t+2\epsilon n$ parties must have voted for the precommit message. Among those parties, at least $2\epsilon n$ are honest parties, so to vote on t$\langle \mathsf{ballot}, r \rangle$, they must have verified it. To verify ballot an honest party first checks that it is of the form $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$, where π_{ballot} is a SNARK_1 proof. The verification of the SNARK proof in Line 23 proves the correctness property of Polling since it ensures that a party votes only if properties i) to iv) for Correctness hold, unless the leader breaks the Knowledge Soundness property of the SNARK scheme, which happens with negligible probability in λ. Similarly, π_{ballot} satisfies the correctness property v).
(*Count validity*) Without loss of generality let all honest parties start Π_{Polling} with input values $\mathsf{v}_i = 1$ (for $\mathsf{v}_i = 0$ the same arguments hold symmetrically.) Then, each honest party signs only messages of the form $\langle 1, r \rangle$ in round 1 of phases while it is not committed. By construction, a leader can only generate valid ballots of the form $\langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$, where either the signature of an honest party p_i is taken into account on count_0, or $p_i \in A : \mathsf{Hash}(A) = H_A$, because otherwise the ballot will not be verified by the SNARK, except with negligible probability $\mathsf{negl}(\lambda)$ that the adversary forges their votes on $\langle 0, r \rangle$. From Agreement all honest parties agree on the same ballot ballot and from correctness $\mathsf{ballot} = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$. Thus, if $h = |A| : \mathsf{Hash}(A) = H_A$, then by construction each honest party is accounted for either in count_1 or in h. Since there are at least $n-t$ honest parties, we thus have $h + \mathsf{count}_1 \geq n-t$.
(Integrity),(Reliability) Once a party decides on a ballot, they set their $\mathsf{accList}$ variable to $\mathsf{cert} = \Psi_i[\mathsf{ballot}_i].\mathsf{accList}$ at Line 45 or Line 52 if they stored it in the (*precommit*) step. Thus, all honest parties either set $\mathsf{accList} = \Psi_i[\mathsf{ballot}_i].\mathsf{accList}$ or $\perp$ and at least $2\epsilon n \geq 1$ honest parties set $\mathsf{accList} \neq \perp$. □

LEMMA 10. Π_{Polling} *achieves Termination.*

Proof. Trivial; after the n phases and the 3 additional rounds each party terminates. □

From Lemmas 8,9, and 10, we conclude that Theorem 3.1 holds.

A.2 Rebuttal Correctness Proof.

LEMMA 11. *Let each honest party* p_i *call* $\Pi_{\mathsf{Rebuttal}}(\mathsf{v}_i, \mathsf{ballot}_i, \mathsf{accList}_i)$, *where* $(\mathsf{ballot}_i, \mathsf{accList}_i) \leftarrow \Pi_{\mathsf{Polling}}(\mathsf{v}_i)$. *Let* A *be the subset of parties for which* $\mathsf{Hash}(A) = H_A \in \mathsf{ballot}_i$. *Then, by the end of* Π_{Rebuttal}, *every honest party* p_i *receives from every honest* $p_j \in A$ *message* $\langle \mathsf{v}_j, r \rangle_{\sigma_j}$, *where* $r \in \mathsf{ballot}_i$.

Proof. From Agreement of Π_{Polling} (Theorem 3.1), all honest parties have the same ballot ballot as input to Π_{Rebuttal}. From Reliability of Π_{Polling} (Theorem 3.1), at least one honest party p_i has $\mathsf{accList}_i = A : \mathsf{Hash}(A) = H_A \in \mathsf{ballot}$. In Round 1 of Π_{Rebuttal}, this honest party sends accusing messages (see Line 3) to every party belonging to A. Therefore, each honest accused party p_j receives an accusing message and subsequently during Round 2 it sends to all parties message $\langle \mathsf{v}_j, r \rangle_{\sigma_j}$, where $r \in \mathsf{ballot}$. □

LEMMA 12. Π_{Rebuttal} *achieves Provable Validity per Theorem 3.3, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. Assume that all honest parties start Π_{Polling} with inputs $\mathsf{v}_i = \mathsf{v}$. Then, each party p_i starts Π_{Rebuttal} with input $(\mathsf{v}, \mathsf{ballot}_i, \mathsf{accList}_i)$, where $\mathsf{ballot}_i = \langle r, \mathsf{count}_0, \mathsf{count}_1, H_A, \pi_{\mathsf{ballot}} \rangle$ and $\mathsf{accList}_i$ are the outputs of Π_{Polling}. Therefore, $\mathsf{ballot}_i, \mathsf{accList}_i$ achieve the properties of Theorem 3.1. Specifically:

(1) All honest parties start Π_{Rebuttal} with the same $\mathsf{ballot}_i = \mathsf{ballot}$ and $\mathsf{accList}_i \in \{A, \perp\}$, where A is the list of parties whose votes were not tallied in ballot and

(2) The sum of $\mathsf{count}_{\mathsf{v}}$ and the number of honest parties within A with initial value $\mathsf{v}_i = \mathsf{v}$ is at least $n - t$ for exactly one $\mathsf{v} \in \{0, 1\}$.

As per Lemma 11, every honest party, whose initial value was not included in ballot, receives an accusing message in the first round of the Rebuttal phase. Once an honest party p_j receives a valid accusing message, it sends $\langle \mathsf{v}, r \rangle_{\sigma_j}$ to all parties. Thus, every honest party receives the initial values of the honest parties in A during Rebuttal. By (2) above and the execution of the protocol in Line 10, all honest parties set $\pi_{\mathsf{v}} = \mathsf{SNRK}_3.\mathsf{Prove}(\mathsf{crs}_3, (\mathsf{v}), (\mathsf{votes}_{\mathsf{v}}, \mathsf{ballot}))$ and π_{v} is a valid SNARK_3 proof. Thus, they will set $\mathsf{y_i} = \mathsf{v}$ and $\mathsf{aux}_i = \pi_{\mathsf{v}}$ in Line 11. Since $\mathsf{count}_{1-\mathsf{v}} \leq t$ and no honest party sends $\langle 1 - \mathsf{v}, r \rangle_{\sigma_j}$, the adversary cannot collect more than $t < n - t$ signatures on $1 - \mathsf{v}$. Therefore, a valid SNARK_3 proof on $1 - \mathsf{v}$ cannot be generated. Thus, all honest parties output $\mathsf{y_i} = \mathsf{v}$ and aux_i s.t. $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$ and no party outputs $(\mathsf{v}', \mathsf{aux}_i)$ s.t. $\mathsf{ThresholdPredicate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$. □

LEMMA 13. Π_{Rebuttal} *achieves Termination.*

Proof. Π_{Rebuttal} runs for 3 synchronous rounds. Accused parties have the opportunity to redeem themselves and send their initial values. At the end of Rebuttal, parties check if they can construct a SNARK_3 proof on v. If so, they update their $\mathsf{y_i}$ from $\perp$ to v, and update aux_i to comprise the SNARK_3 proof on v. Finally, the parties output $\mathsf{y_i}$ and aux_i. □

From Lemmas 12, and 13, we conclude that Theorem 3.3 holds.

A.3 Weak Byzantine Agreement Correctness Proof.

LEMMA 14. *If a leader creates a valid commit certificate C in a phase r, no conflicting certificate $C' \neq C$ is created in the same phase, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. Let phase r. For party p_r to generate a valid commit certificate say C, they need to aggregate at least $n - t$ distinct valid signatures for the respective message $\langle \mathsf{v}, r \rangle$, unless they forge the necessary signatures, which can occur with negligble probability in λ. Since $t < (1/2 - \epsilon)n$, then $n - t > t + 2\epsilon n$. Thus, more than $2\epsilon n$ honest parties signed and propagated the leader's proposed value $\mathsf{v} \in C$ via the expander graph. By the expander's property (Definition 2.7), more than $(1 - 2\epsilon)n > 2t$ parties receive (*precommit*) message containing $\mathsf{v} \in C$. Out of these parties, at least $t + 1$ are honest. In Round 3, those honest parties do not vote for any conflicting $\mathsf{v}' \neq \mathsf{v}$, in other words, they do not send $\langle \mathsf{v}', r \rangle$ signed back to the leader. So, the leader cannot collect more than $n - (t + 1)$ signatures on any such message $\langle \mathsf{v}' \neq \mathsf{v}, r \rangle$, and thus cannot create a valid conflicting commit certificate, except with negligible probability $\mathsf{negl}(\lambda)$. □

LEMMA 15. *If all honest parties start with input $\mathsf{v}_i = \mathsf{v}$ and aux_i s.t. $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$, and no party starts with v' s.t $\mathsf{ThresholdPredicate}(\mathsf{v}', \mathsf{aux}_i) = \mathsf{true}$, then, no leader can create a valid commit certificate on $\mathsf{v}' \neq \mathsf{v}$, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. For a leader to successfully create a valid commit certificate on v', they must receive at least $n - t$ signatures on $\langle \mathsf{v}', r \rangle$ to form a valid QC_{commit}, unless they forge the necessary signatures, which can occur with negligble probability in λ. If a malicious leader proposes $(\mathsf{v}_r = \mathsf{v}', \mathsf{aux}_r)$, with $\mathsf{ThresholdPredicate}(\mathsf{v}_r, \mathsf{aux}_r) = \mathsf{false}$, then no honest party propagates the leader's proposal in Line 20, or send their vote to the leader in Line 22 as $\mathsf{v}_{\mathsf{commit}} \neq \perp$ and $\mathsf{SafeVal}(\mathsf{v}_r, \mathsf{aux}_r, \mathsf{v}, \mathsf{aux}_i, \mathsf{v}_{\mathsf{commit}}) = \mathsf{false}$. Since, $t < n - t$, the adversary can not collect enough votes on v' to create a valid QC_{commit} and thus a commit certificate on v', except with negligible probability $\mathsf{negl}(\lambda)$. □

LEMMA 16. Π_{WBA} *achieves Restricted Validity, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. Assume all honest parties have the same input value, denoted as v, and auxiliary information aux_i such that $\mathsf{ThresholdPredicate}(\mathsf{v}, \mathsf{aux}_i) = \mathsf{true}$. Based on Lemma 15, no leader can create a valid commit certificate on $\mathsf{v}' \neq \mathsf{v}$, except with negligible probability in λ. Therefore, during the protocol, if a party receives a commit certificate with a valid QC_{commit}, it must be for the input value v. Subsequently, parties only send matching decide message on v. This means that a leader cannot collect enough votes to form a valid decide certificate for $\mathsf{v}' \neq \mathsf{v}$ because they cannot create a valid commit certificate for v' in the first place. Thus, if a party receives a valid decide certificate on v, they output v. According to Lemma 5, all parties decide within $f+1$ phases, and hence, everyone outputs v, except with negligible probability $\mathsf{negl}(\lambda)$. □

LEMMA 17. Π_{WBA} *achieves Agreement, except with negligible probability* $\mathsf{negl}(\lambda)$.

Proof. By Lemma 5, all parties are decided at the end of the protocol. Let p_i be the honest party who decides on the earliest phase, say $k > 0$. Assume that p_i decides value v. For every other honest party p_j who decides value v' we distinguish two cases, based on the phase in which p_j decides.

Case 1: p_j decides v' in phase k. Since p_i decided v, they received a valid decide certificate for v. To create that certificate, the leader must have collected at least $n-t$ votes (signatures) on $\langle decide, \mathsf{v}, k\rangle$, unless the leader forges votes, which happens with negligible probability $\mathsf{negl}(\lambda)$. A party only sends their vote once they receive a valid commit certificate with rank k from the leader. From Lemma 14, the leader can create only one commit certificate in their phase, except with negligble probability in λ. Since p_i decided v, there must be a valid commit certificate in phase k for v. Thus the only commit certificate the leader can generate in phase k is for value v. Since p_j decides v', there must be a valid commit certificate in phase k for v', therefore, $\mathsf{v}' = \mathsf{v}$, except with negligible probability $\mathsf{negl}(\lambda)$.

Case 2: p_j decides v' in phase $l > k$. Again, since p_i decided v in phase k, they received a valid decide certificate for v and to create that certificate, the leader must have collected at least $n-t$ votes (signatures) on $\langle decide, \mathsf{v}, k\rangle$, unless the leader forges votes, which happens with negligible probability $\mathsf{negl}(\lambda)$. Even if all t malicious parties send their votes, at least $n - 2t \geq 2\epsilon n$ honest parties must also vote for it. This means they received a commit certificate C_k on v and propagated it to their neighbors in Line 28 before sending a matching decide vote. Due to the expansion property, at least $(1-2\epsilon)n > 2t$ parties received C_k. Out of all such parties, at most t were corrupted, so at least $t+1$ were honest. In round 5, these at least $t+1$ honest parties committed to v, and set their $\mathsf{rank}_{\mathsf{commit}}$ to k. In any subsequent phase $l > k$, these at least $t+1$ parties propose v to the leader p_l unless they receive a commit certificate C_l with a higher rank $l > k$. For a leader to create a commit certificate on v' with a higher rank, they have to gather at least $n-t$ votes on v'. If the leader proposes v' via a commit certificate that was created in phases $m < k$, those $t+1$ honest parties will not send their votes as they have a higher commit rank. If the leader proposes v' by sending $(\langle \mathsf{v}', r\rangle_{\sigma_r}), \langle \mathsf{aux}\rangle_{\sigma_r})$, and receives back enough votes, then at least one of those $t+1$ honest parties must have voted for v' except with negligible probability $\mathsf{negl}(\lambda)$ of the leader forging the necessary signatures. Since they have already committed to v and they voted for v', then $\mathsf{v}' = \mathsf{v}$, except with negligible probability $\mathsf{negl}(\lambda)$. □

LEMMA 18. Π_{WBA} *achieves Termination*

Proof. The protocol runs for n phases; each consists of 6 synchronous rounds. From Lemma 5, all honest parties are decided within $f+1$ phases. Since $f \leq t < \frac{n}{2}$, every honest party terminates by the end of the n phases. □

From Lemmas 16, 17 and 18, we conclude that Theorem 4.1 holds.

B WBA protocol from Cohen et al [15]

Overview. In this section, we give a brief overview of the WBA protocol from [15] and point out a potential attack (and a simple fix) in their `Invoke_Phase` subroutine. The protocol consists of two parts; `Invoke_Phase`, which they propose, and a fallback algorithm [29]. During the invoke phase, the honest parties aim to reach agreement on a value. If some parties fail, which only occurs when $f > \frac{t}{2}$, the protocol proceeds to the fallback algorithm [29], which has a communication complexity of $O(n^2)$. The invoke phase subroutine is based on the

silent phases framework. Each phase consists of three rounds; propose, commit, and decide.
(propose) First, the leader, if undecided, proposes the leader's initial value v_i and tries to get the parties to agree on the leader's proposed value. Upon receiving the leader's proposal, the honest parties either sign this value or respond with a value that they had committed to earlier along with the associated commit proof π_{commit}, which is an aggregate $\lceil \frac{n+t+1}{2} \rceil$ signatures on the committed value.
(commit) The leader broadcasts the commit message if it has been received from a party. Otherwise, if the leader has collected at least $\lceil \frac{n+t+1}{2} \rceil$ signatures on the proposal, the leader can generate a commit quorum QC_{commit} on the leader's proposed value. The threshold ensures that the leader cannot create two commit certificates on conflicting values, unless an honest party votes for both values in the same phase. Upon receiving a commit message on v consisting of the value and commit proof, the party sends a matching decide message on v, *even if it is committed on a different value.* Also, the party commits to this value if it has not already committed.
(decide) If a leader collects at least $\lceil \frac{n+t+1}{2} \rceil$ signatures on the same decide message, the leader can form a decide certificate QC_{decide} by aggregating the decide messages, which the leader then sends to the parties to ensure a safe decision on the proposed value. Finally, upon receiving a decide certificate from the leader for value v, the party outputs v. For a more detailed explanation of the protocol, the reader may refer to [15].

Agreement Attack. The `Invoke_Phase` subroutine could fail to satisfy the agreement property in the case that $f > \frac{t}{2}$ and the first few leaders are malicious. The root cause of this problem lies in the ability of consecutive malicious leaders to collect enough votes to form conflicting QC_{commit} and QC_{decide}. This would lead the parties to agree on two conflicting values. The vulnerability arises because honest parties send matching decide messages to the commit certificate they received from the leader, even if they are committed to a different value. Therefore, if malicious leaders create conflicting commit certificates, they can easily collect enough votes to create two valid conflicting decide certificates. This exploit is achievable by having the first few leaders in the run be malicious. Once the first malicious leader forms a valid commit certificate on v (which the leader can do since it is the first phase and no party is committed), the leader can strategically send the created certificate to a small subset of honest parties. This approach allows the following malicious leader to collect enough votes on the leader'sproposed value $\mathsf{v}' \neq \mathsf{v}$ and create a conflicting commitment certificate. The adversary now owns two conflicting commit certificates and can equivocate two values. So, honest parties will decide on conflicting values, thereby violating the agreement property. For clarity, we now illustrate an attack that breaks agreement.

This attack assumes that the first three leaders are malicious. Assume $f = t$, then, there are $n - t = t + 1$ honest parties. We will divide the honest parties into three groups: A, B and C, where $|A| = \frac{t}{2} - 2$, $|B| = \frac{t}{2} - 1$, and $|C| = 4$. The phases are as follows:

- *Phase 1*: P_1 proposes v, collects $\lceil \frac{n+t+1}{2} \rceil$ votes on v from parties, creates a valid commit certificate on v, and sends the certificate only to group A (only Group A is committed on v.)

- *Phase 2*: P_2 proposes v' to groups B and C, and ignores the commit messages received from group A. Since B and C are not committed, they sign the leader's proposal and return it back to P_2. Since $B+C+f > \lceil \frac{n+t+1}{2} \rceil$, P_2 creates a valid conflicting commit certificate on v' and forwards it to B and C. Once they receive it, they send a matching decide message (votes) on v'. Similarly, P_2 collects enough votes to create a valid decide certificate on v', which P_2 sends to group B only. Group B is now decided on v'.

- *Phase 3*: P_3 proposes any value and receives commit messages from all groups. P_3 broadcasts the commit message on v received from group A. Groups A and C send a matching decide message to P_3 according to line 16 of the `Invoke_Phase` protocol in [15]. Since $A + C + f > \lceil \frac{n+t+1}{2} \rceil$, P_3 creates a valid decide certificate on v for groups A, and C. Therefore, A and C decide on v.

In subsequent rounds, all honest parties are decided and every honest leader will remain silent. Eventually, group B will output v, and groups A and C will output v', thus breaking agreement.

Proposed Solution. To prevent the agreement attack against the protocol, we must prevent group C from sending a decide message on a value v that is different from the one to which it is committed (v'). Consequently, the leader will never collect enough decide messages on v and will fail in convincing groups A and C to decide on v; in essence, creating a valid decide certificate on v'. To implement this, we follow the intuition of certificate ranks [26] (we utilize it similarly in our protocols). Intuitively, a certificate is ranked based on the phase in which

it was created. In the suggested protocol, a party only updates its commit value and sends a matching decide message if the commitment message received has a higher $\mathsf{rank}_{\mathsf{commit}}$ than the one to which the party is committed. Therefore, in the agreement attack, when the group C receives a commitment message in phase 3 where π_{commit} aggregates $\lceil \frac{n+t+1}{2} \rceil$ messages $\langle \mathsf{v}, 1 \rangle_{\sigma_{p_1}}$ of rank 1, it will not send a matching decide message. This is due to the fact that the group C is already committed on v' with π_{commit} that aggregates $\lceil \frac{n+t+1}{2} \rceil$ messages $\langle \mathsf{v}', 2 \rangle_{\sigma_{p_2}}$ of rank 2.

Our modification only affects Lemma A.2 in [15]. All the other lemmas and arguments are not affected. Please refer to [15] for the terms and variables used here.

LEMMA 19. *If an honest party decides on* v *during* `Invoke_Phase`*, no honest party decides on* $\mathsf{v}' \neq \mathsf{v}$*. In addition, at most one decide certificate can be formed in all phases.*

Proof. Let p_i be the honest party who decides on the earliest phase, say $k > 0$. Assume that p_i decides value v. For every other honest party p_j who decides value v' we distinguish two cases, based on the phase in which p_j decides.

Case 1: p_j decides v' in phase k. A party decides a value once it receives a decide certificate. Thus, p_i and p_j must have received a valid decide on v and v' respectively. Since a decide certificate is only formed by aggregating $\lceil \frac{n+t+1}{2} \rceil$ decide messages on the same value, an honest party must have sent two decide messages on v and v' for the leader to create two conflicting valid decide certificates. This is impossible because the pseudo-code allows sending decide message only once by an honest party per phase.

Case 2: p_j decides v' in phase $l > k$. In phase k, p_i receives a decide certificate on value v signed by $\lceil \frac{n+t+1}{2} \rceil$ distinct parties. Hence, at least $\lceil \frac{n+t+1}{2} \rceil - t \geq \frac{t}{2} + 1$ parties updated their $\mathsf{commit\_var} = \mathsf{v}, \mathsf{rank}_{\mathsf{commit}} = k$, and π_{commit}. As they are committed, they do not vote on any subsequent leader's proposed value for any phases after k unless the leader provides a valid commit certificate with rank greater than k. For phases after k, at most $n - t - (\frac{t}{2} + 1) = \frac{t}{2}$ honest parties sign a conflicting proposed value v' by the leader, which is not enough to form a valid conflicting commitment message, as $\frac{t}{2} + t < \lceil \frac{n+t+1}{2} \rceil$. Thus, the leader will never collect enough decide messages on v' to form a conflicting valid decide certificate. Consequently, there is at most one decide certificate that can be formed in all phases. ∎