Early-Stopping Dishonest Majority Broadcast with Linear Rounds and Polynomial Communication

Renas Bacho[1] and Kecheng Shi[2]

[1] Ruhr University Bochum
[2] CISPA Helmholtz Center for Information Security

Abstract. Early-stopping Byzantine broadcast protocols adapt their round complexity to the number $f \leq t$ of parties that are actually corrupted in an execution, where $t < n$ is the corruption threshold known to the protocol. For every $t < n$, Loss and Nielsen (Eurocrypt 2024) gave a protocol that terminates within $O(\min\{f^2 + 1, t + 1\})$ rounds, using polynomial communication and a plain public key infrastructure (PKI). More recent work attains $O(f + 1)$ rounds for every $t < n$ via exponential information gathering, but at the cost of exponential communication. It has therefore remained open whether $O(f+1)$ rounds can be achieved for every $t < n$ with polynomial communication and computation from a plain PKI.

We resolve this question with a deterministic Byzantine broadcast protocol that is secure against a strongly adaptive rushing adversary. For every $f \leq t < n$, all honest parties output and terminate within $O(f + 1)$ rounds, using polynomial communication and computation. This matches the classical early-stopping lower bound of $\min\{f + 2, t + 1\}$ rounds up to a constant factor. Our basic protocol terminates within $14f + 9$ rounds and communicates $O(\kappa n^7)$ bits, where κ is the length of a signature. Two optimized variants improve both bounds: the first terminates within $5f + 2\lceil \log_2(f + 1) \rceil + 10$ rounds and communicates $O(\kappa n^5)$ bits, and the second terminates within $5f + 2\lfloor \sqrt{f} \rfloor + 11$ rounds and communicates $O(\kappa n^4 + n^{9/2})$ bits.

1 Introduction

In Byzantine broadcast [26], a designated sender holds an input that it wants to communicate to n parties, up to t of which may deviate arbitrarily from the protocol. All honest parties must output the same value, and if the sender is honest, this value must be its input. Broadcast is a fundamental building block of distributed cryptographic protocols such as multi-party computation and verifiable secret sharing. In a synchronous network with authenticated channels and signatures, the Dolev–Strong protocol [17] runs for $t + 1$ rounds, even in executions with few or no corruptions. Early-stopping broadcast asks how quickly all honest parties can output and terminate when only $f \leq t$ parties are actually corrupted. For $t < n - 1$, the classical lower bound of Dolev, Reischuk, and Strong [16] is $\min\{f + 2, t + 1\}$ rounds.

For every $t < n$, Loss and Nielsen [28] gave a protocol that terminates within $O(\min\{f^2 + 1, t + 1\})$ rounds, with polynomial communication and a plain public key infrastructure (PKI). Deligios, Klasovitá, and Liu-Zhang [13] then achieved $O(f + 1)$ rounds with polynomial communication and a plain PKI, but only when $t < (1 - \varepsilon)n$ for a constant $\varepsilon > 0$. For every $t < n$, protocols based on exponential information gathering achieve $O(f + 1)$ rounds [12,6]. These results leave open the following central question.

Can deterministic Byzantine broadcast achieve $O(f + 1)$ rounds with polynomial communication and computation for every $t < n$ using only a plain PKI?

1.1 Our Results

We answer this question affirmatively with a deterministic binary broadcast protocol that terminates within $O(f + 1)$ rounds and tolerates any number $t < n$ of corruptions.

Theorem 1.1. *For every $t < n$, there is a deterministic binary Byzantine broadcast protocol for synchronous pairwise authenticated channels and a plain PKI with the following guarantees against a strongly adaptive rushing adversary that corrupts $f \leq t$ parties, except with negligible probability. All honest parties output the same bit and terminate within $14f + 9$ rounds, including their last messages. If the sender is honest, this bit is its input. The honest parties communicate $O(\kappa n^7)$ bits in total and use polynomial computation and memory, where κ is the length of a signature and $\kappa > \log_2 n$.*

When $f < n/2$, the protocol uses a Byzantine agreement subroutine for an honest majority to let the honest parties agree quickly on a candidate. Each party then reports its candidate. For each report, honest parties accept at most one bit, even if the party that sends the report is corrupted. An honest party can therefore set its output as soon as it accepts reports with the same bit from a strict majority of the parties. Otherwise, all honest parties obtain the same complete vector of reports by a time cap and apply the same deterministic output rule. This time cap is $O(t+1)$ rounds, which is $O(f+1)$ whenever $f \geq n/2$.

Section 5 analyzes the complexity of the basic protocol and describes two optimized variants, which also work for every $t < n$. The first terminates within $5f + 2\lceil \log_2(f+1) \rceil + 10$ rounds and communicates $O(\kappa n^5)$ bits. The second terminates within $5f + 2\lfloor \sqrt{f} \rfloor + 11$ rounds and communicates $O(\kappa n^4 + n^{9/2})$ bits. While our protocols are stated for binary inputs, they extend directly to inputs of polynomial length. Namely, run one instance per bit in parallel and concatenate the outputs. This preserves the round bound and keeps communication polynomial.

1.2 Related Work

Table 1 compares our protocol with existing deterministic Byzantine agreement (BA) and Byzantine broadcast (BB) protocols. In BA every party starts with an input, whereas in BB only the designated sender holds an input. The resilience column specifies the range of t in which the displayed round complexity applies.

Table 1. Comparison of deterministic Byzantine agreement (BA) and Byzantine broadcast (BB) protocols. Here t is the maximum number of corruptions tolerated, f is the number of parties actually corrupted in an execution, $\epsilon > 0$ is a fixed constant, and κ is the signature length in bits. TS denotes threshold-signature setup and an asterisk indicates that the analysis assumes idealized transferable signatures.

Protocol	Task	Resilience	Rounds	Communication	Setup
Garay and Moses [23]	BA	$t < n/3$	$\min\{f+5, t+1\}$	polynomial	none
Berman et al. [8]	BA	$t < n/3$	$\min\{f+2, t+1\}$	exponential	none
Abraham and Dolev [1]	BA	$t < n/3$	$\min\{f+2, t+1\}$	polynomial	none
Elsheimy et al. [18]	BA	$t < n/3$	$f + O(\sqrt{f}+1)$	quadratic	none
Elsheimy et al. [19]	BA	$t < n/2$	$(1+\epsilon)f + O(1)$	polynomial	plain PKI
Elsheimy et al. [20]	BA	$t < n/2$	$f + 6\lceil\sqrt{f}\rceil + 6$	polynomial	plain PKI
Elsheimy et al. [18]	BA	$t < n/2$	$f + O(\sqrt{f}+1)$	quadratic	PKI + TS
Choi et al. [10]	BB	$t < n/2$	$\min\{f+2, t+1\}$	exponential	plain PKI*
Deligios et al. [12]	BB	$t < n/2$	$\min\{f+2, t+1\}$	exponential	plain PKI*
Dolev and Strong [17]	BB	$t < n$	$t+1$	polynomial	plain PKI
Loss and Nielsen [28]	BB	$t < n$	$O(\min\{f^2+1, t+1\})$	polynomial	plain PKI
Deligios et al. [13]	BB	$t < (1-\epsilon)n$	$O(f+1)$	polynomial	plain PKI
Deligios et al. [12]	BB	$t < n$	$9f + 7$ for $f \geq 1$	exponential	plain PKI*
Bacho and Shi [6]	BB	$t < n$	$\min\{f+3, t+1\}$	exponential	plain PKI*
This work (Section 4)	**BB**	$\boldsymbol{t < n}$	$\boldsymbol{14f + 9}$	**polynomial**	**plain PKI**
This work (Section 5.2)	**BB**	$\boldsymbol{t < n}$	$\boldsymbol{5f + O(\log(f+2))}$	**polynomial**	**plain PKI**

Classical round bounds and early stopping without signatures. Pease, Shostak, and Lamport [30,26] introduced Byzantine agreement and broadcast, along with the recursive exchange of views that underlies exponential information gathering (EIG). Fischer and Lynch [21] established the worst-case round lower bound of $t+1$ without authentication, and Dolev and Strong [17] extended this lower bound to authenticated broadcast while giving a polynomial-communication protocol that terminates in $t+1$ rounds. Dolev, Reischuk, and Strong [16] initiated the study of early-stopping agreement, showing that eventual agreement requires $\min\{f+2, t+1\}$

rounds for $t < n - 1$ when counting the final round in which an honest party sends a message. In the unauthenticated setting ($t < n/3$), early-stopping protocols were developed through broadcast primitives simulating authentication [33], algorithm gear-shifting [7], fault diagnosis [11], EIG trees [8], and polynomial-communication constructions [23], culminating in the polynomial-time protocol of Abraham and Dolev [1] achieving the optimal $\min\{f + 2, t + 1\}$ bound.

Early stopping with an honest majority. For $t < n/2$ with signatures in a plain PKI, Elsheimy, Loss, and Papamanthou [19] improved upon the classical $\min\{2f + 4, 2t + 2\}$-round protocol of Perry and Toueg [31] to achieve $(1 + \epsilon)f + O(1)$ rounds. Subsequently, Elsheimy, Loss, and Papamanthou [20] obtained $f + 6\lceil\sqrt{f}\rceil + 6$ rounds by running iterations of d-detecting graded consensus in which honest parties either reach agreement or commonly detect at least d additional corrupted parties. Our binary agreement subroutine (Section 4.1) uses three-round iterations ($d = 1$) and relies on the same kind of common fault detection. Elsheimy, Kamp, Loss, and Papamanthou [18] further achieved $f + O(\sqrt{f} + 1)$ rounds together with $O(n(t + 1)\kappa)$ communication under threshold signature setup for $t < n/2$, as well as setup-free results for $t < n/3$. Using exponential communication, Choi et al. [10] and Deligios et al. [12] attained $\min\{f + 2, t + 1\}$ rounds for $t < n/2$.

Early stopping with a dishonest majority. Loss and Nielsen [28] gave the first deterministic early-stopping broadcast protocol tolerating any $t < n$ corruptions in the plain PKI model. Their construction introduced *polarizers*—transferable cryptographic evidence of missing messages built from signed accusations—within a Send Transferable Message (STM) primitive and its agreement extension (STMA), achieving polynomial communication and $O(\min\{f^2+1, t+1\})$ rounds. Deligios, Klasovitá, and Liu-Zhang [13] generalized polarizers to accusation graphs in which honest parties accuse neighbors whose distance to the sender remains too small; because honest parties never accuse each other, the honest parties form a clique and any shortest path from the honest clique to a corrupted sender has length at most f. This graph-based STM primitive allowed [13] to achieve $O(f + 1)$ rounds when a constant fraction of parties is honest ($t < (1 - \epsilon)n$), while retaining $O(\min\{f^2 + 1, t + 1\})$ rounds for general $t < n$. Recently, Deligios et al. [12] used EIG with signed confirmations to obtain $9f + 7$ rounds for $f \geq 1$ and $t < n$, at the cost of exponential communication. Our protocol builds on the graph-based STM primitive of [28,13], the two-layer report structure of STMA [28], and Dolev–Strong signature chains [17], combining them with concurrent honest-majority BA pairs to achieve $O(f + 1)$ rounds with polynomial communication and computation for every $t < n$.

Communication, fault models, and randomized broadcast. Lower bounds on messages and signatures for early-deciding consensus were studied by Dolev and Reischuk [15] and Dolev and Lenzen [14], while Lenzen and Sheikholeslami [27] gave communication-efficient recursive phase-king protocols for $t < n/3$. Stopping bounds across crash, omission, and Byzantine models have been analyzed via layered and knowledge-based frameworks [29,25,5,10]. A complementary line of work studies *good-case latency*—the round complexity when the sender is honest—across synchronous and partially synchronous models [2,3] or adds optimistic fast tracks to dishonest-majority broadcast [24]; by contrast, early-stopping broadcast bounds termination as a function of f even when the sender is corrupted. Finally, randomized broadcast protocols [22,9,32,34,4] achieve sublinear or expected constant round complexity against dishonest majorities, whereas our focus is on deterministic worst-case termination in $O(f + 1)$ rounds.

2 Technical Overview

We first review recent work on early-stopping broadcast with a dishonest majority. We then outline the main techniques used in our base protocol in Section 4.

2.1 Polarizer Protocols

Loss and Nielsen [28] introduce polarizers to address a basic problem: a party that receives nothing has no sender-signed message to forward. We use the accusation-graph formulation of Deligios et al. [13]. Start with a complete graph on the parties and remove an edge if either endpoint has signed an accusation against the other. Honest parties never accuse one another, so they remain a clique. Thus, if a set of accusations disconnects one honest verifier from the sender, it disconnects every honest verifier. This cannot happen when

the sender is honest. Send Transferable Message (STM) gives each honest party either a valid signed sender message or NoMsg, together with a proof called a *justifier*. A justifier for NoMsg is a polarizer. A justifier that verifies at one honest party also verifies at every other honest party. When the sender is faulty, honest parties can still obtain different local outputs: for example, one can obtain a signed message while another obtains NoMsg.

Loss and Nielsen then use STM to build STMA, which guarantees that at most one message other than NoMsg has an acceptable proof. Some parties may still obtain NoMsg. They then build a justified gradecast that outputs a value with a grade in $\{0, 1, 2\}$, where each gradecast contains several sequential STM calls, and each call takes $O(f + 1)$ rounds. If a valid grade-2 proof for v exists, every valid output of that instance has positive grade and value v. The outer protocol runs one gradecast with a distinct leader in each iteration. A later leader must provide proofs of the earlier graded outputs and use them to justify its proposed value. Once an iteration has a grade-2 proof for v, these checks prevent later iterations from producing a positive-grade output for another value. A party can thus forward a grade-2 proof and decide. At most $f + 1$ leaders are needed before an honest leader runs the gradecast and gives all honest parties grade 2. Since each gradecast takes $O(f + 1)$ rounds, this part of the protocol takes $O((f + 1)^2)$ rounds in total.

Deligios, Klasovitá, and Liu-Zhang [13] reduce the round complexity for the inner primitive. They use the known lower bound $n - t$ on the number of honest parties. Their protocol represents accusations by a graph in which honest parties remain a clique. Knowing that this clique has at least $n - t$ vertices allows the protocol to remove additional edges without disconnecting honest parties from one another. The protocol can then obtain a sender message or a polarizer in fewer rounds. When $n - t > \epsilon n$ for a fixed $\epsilon > 0$, each inner call takes a constant number of rounds, and the full broadcast protocol takes $O(f + 1)$ rounds. The graph operations can be implemented in polynomial time. The remaining case is when t is close to n but f is small.

2.2 Fast Candidate Selection with an Honest Majority

When $f \geq n/2$, we have $t + 1 \leq n \leq 2f$, so a deadline of $O(t + 1)$ rounds already gives termination within $O(f + 1)$ rounds. The challenge is to finish quickly when $f < n/2$, while keeping the protocol correct for every $f \leq t$. The parties do not know f, so they cannot treat a BA output as a broadcast decision. We use BA to select a candidate bit with a valid justifier. Let $q = \lfloor n/2 \rfloor + 1$ and $\sigma = \min\{t, \lfloor (n - 1)/2 \rfloor\}$. Here q is the minimum size of a strict majority, and for $f \leq t$ the honest parties form a strict majority exactly when $f \leq \sigma$. We use two guarantees, stated for an execution in which the outer protocol keeps running:

1. If $f \leq \sigma$, all honest parties choose the same candidate bit, each with a valid sender justifier, within $O(f + 1)$ rounds.
2. For every $f \leq t$, each honest party chooses a candidate bit with a valid sender justifier by the public deadline $D = O(t + 1)$. The candidates may differ when $f > \sigma$.

Sender justifiers come from the initial instance STM_S. A valid signed output of STM_S justifies its bit, while a valid polarizer of STM_S justifies the default bit 0. If the original sender is honest, only its input bit can be justified. Every honest party uses the same two-layer report construction, adapted from the STMA protocol of Loss and Nielsen [28]. In the first layer, after fixing its candidate, party j sends the candidate and its STM_S justifier through its own STM. In the second layer, each party sends what it obtained from that first instance through a separate STM. A tuple for report j contains one second-layer STM output per party. First, verify every entry. Then skip an entry only if the second-layer output itself is NoMsg, with a valid polarizer against its second-layer sender. The tuple certifies v if at least one entry remains and every remaining entry supports v. A signed second-layer message saying that its sender obtained NoMsg in the first layer is not skipped. It prevents that tuple from certifying either bit.

To see why report j cannot be certified for both bits, fix any honest second-layer sender. A polarizer against that party cannot verify, so every valid report certificate must contain its signed second-layer payload. The party signs at most one such payload. If it signs a bit, a report certificate can certify only that bit. If it signs $\mathsf{NoMsg}^{(j)}$, no report certificate can certify either bit. The delivery rule determines when a party accepts a certified report bit.

2.3 Obtaining an Early BA Result

Our three-round BA iteration uses the fault detection idea of [20]. The argument below assumes $f \leq \sigma$, so there are at least q honest parties and every set of q parties contains an honest party. In each iteration, party i computes a vector y_i, where $y_{i,j}$ is its estimate of party j's current input. It takes the majority of this vector as its next input, breaking ties in favor of 0. All entries are computed in parallel as follows.

1. In the Send round, each party with input 1 signs and sends that bit to everyone. A party with input 0 sends no signed bit.
2. In the Vote round, a party that received j's signed 1 directly sends a signed vote for j to everyone, unless it has already marked j. The vote includes j's signed bit.
3. In the Set round, a party that received valid votes for j directly from at least q distinct parties sends a SET containing q such votes. A party that received no valid vote sends ZERO. A party that received between 1 and $q-1$ valid votes sends neither and marks j for subsequent iterations.

Party i sets $y_{i,j} = 1$ if it receives a valid SET for j in the Set round, and sets $y_{i,j} = 0$ otherwise. Honest parties are never marked: if honest j has input 1, every honest party receives votes for j from all honest parties; if honest j has input 0, no valid vote for j exists. If two honest parties compute different next inputs, their vectors differ at some entry j. The SET that caused one party to set that entry to 1 contains an honest vote. Every honest party received that vote, but no honest party counted q votes: such a party would have sent a SET to all parties, making every honest party set the entry to 1. Thus all honest parties mark j. At least the honest voter had not marked j before, so j is new to the intersection of the honest parties' faulty lists. At most f iterations can leave the honest parties with different next inputs.

An entry equal to 1 has high grade if the party received valid SET messages for that entry directly from at least q distinct parties in the Set round. At least one of those parties is honest and sent its SET to everyone. An entry equal to 0 has high grade if the party received ZERO messages directly from at least q distinct parties in that round. At least one honest party then counted no valid vote, which rules out any valid SET for that entry. In either case, all honest parties have the same value for the entry. A party sets its BA output to v when at least q entries have value v and high grade. These entries make v the common next input, and a common input gives every honest party at least q high-grade entries with that value in the next iteration. Since a common input is preserved, all honest parties have a common next input by the end of iteration $f+1$ and set their BA outputs by iteration $f+2$. Setting a BA output does not end the instance: parties continue sending messages and updating their state through iteration $\sigma+2$, unless the broadcast protocol halts first. At that iteration cap, any unset BA output is set to $\perp$.

2.4 Completing the Broadcast Protocol

Figure 1 shows how the protocol components fit together and when the BA pairs run.

Waiting for the worst-case STM_S deadline before starting BA would lose early stopping. Instead, for each $e = 1, \ldots, \sigma+2$, we run a pair of BA instances, one for each candidate bit. In round e, each party that already holds its own STM_S output announces it in the input field for pair e. At the end of that round, a party sets its input to the instance for v to 1 if it received a valid justifier for v in that field, and to 0 otherwise. Both BA instances begin in round $e+1$. In that same round, each party forwards its stored justifier for each bit whose BA input it set to 1. These forwards can supply a missing justifier for candidate selection; they do not change the BA inputs. A pair completes locally when both of the party's BA outputs are set. The pair selects the least bit v for which its instance for v returns 1 and the party holds a valid sender justifier for v in that pair. It selects nothing if either output is $\perp$ or if no such bit exists. A party uses the first pair that selects a bit, in order of starting round, once all earlier pairs have completed locally. It does not wait for later pairs. A party still without a candidate at the end of round $D = 4\sigma + 8$ uses the bit supported by its own STM_S output, interpreting NoMsg as bit 0, and uses that output as its sender justifier.

Through round $B = D + 4t + 7$, parties accept valid report certificates directly and forward a report's first accepted certificate in the next round if it was accepted before B. In round $B+1$, a party starts a Dolev–Strong chain for each report whose certificate it holds, even if it already forwarded that certificate. In round $B+\ell$, it accepts only valid chains of length ℓ with distinct signers and a valid report certificate for the chain's bit. If $\ell \leq t$, a party that has not signed a chain for this report extends one valid chain that does not

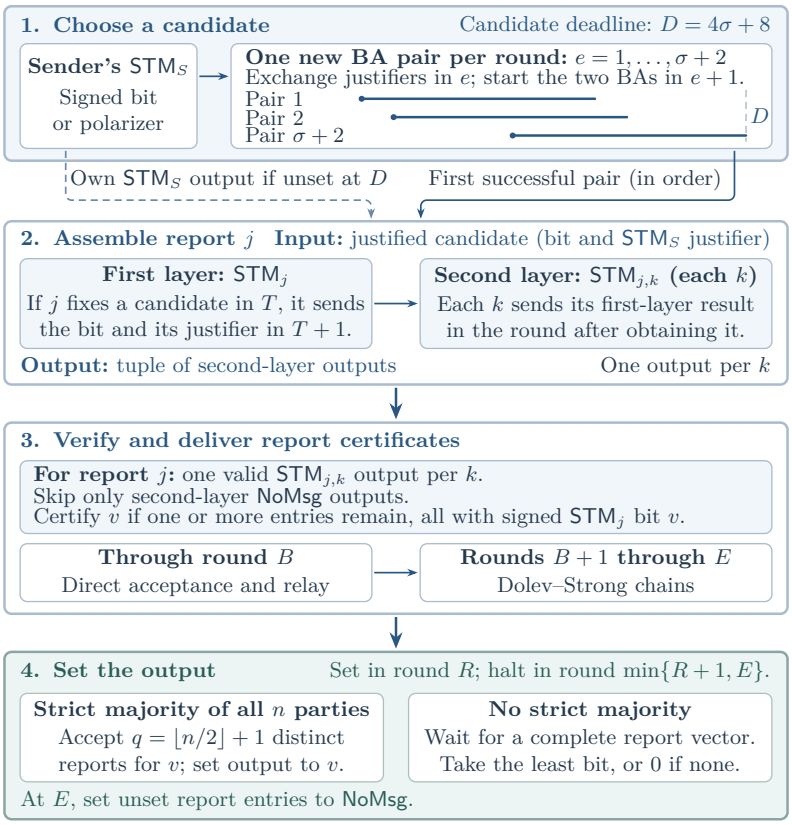


Fig. 1. Overview of the broadcast protocol. The bars show the scheduled BA rounds: the last pair starts in $\sigma + 3$ and runs at most through D. A party selects the first successful pair in start order, after all earlier pairs complete. Reports can start before D. Unresolved first-layer instances use the full STM rules from $D + 1$. A report certificate cannot contain a signed second-layer claim $\mathsf{NoMsg}^{(j)}$ stating that its sender obtained NoMsg in the first layer. The majority threshold is relative to all n parties, even if some reports are missing. An entry set to NoMsg at E means that no bit for that report was accepted by the deadline.

already contain its name and sends it in the next round. A chain accepted in the last round, $E = B + t + 1$, must already contain an honest signer who sent its prefix to everyone. At E, each party assigns NoMsg to any report for which it has not accepted a bit. Here NoMsg marks a missing report entry; it is not a justified STM output.

A party sets its output when q accepted reports support the same bit, or when it has a complete report vector. For a complete vector, it takes the strict majority if one exists, the least bit present otherwise, and 0 if no bit is present. By Lemma 4.4, two honest parties cannot accept different bits for the same report j. If all honest parties continue through round E, the delivery phase gives them the same complete report vector, and any q matching reports used earlier remain a strict majority in that vector.

In the actual execution, a party that sets its output in round R continues through round $\min\{R + 1, E\}$ before producing its output and halting. Thus, if the first honest output is set before round E, all honest parties continue through the following round. As shown in Section 4.6, every report that enabled that output reaches all honest parties by then, so they set the same output. The case in which no honest party sets its output before round E follows from the common complete report vector.

Suppose $f \leq \sigma$. In a full execution, a sender justifier gives all honest parties input 1 for the same bit in one BA pair within $O(f + 1)$ rounds. This pair and all earlier pairs complete within $O(f + 1)$ rounds, so the honest parties choose a common candidate. Within another $O(f + 1)$ rounds, they accept matching reports from a strict majority. The proof in Section 4.6 uses this bound to show when honest parties halt in the actual execution. When $f > \sigma$, the public deadline is already $O(f + 1)$.

3 Preliminaries

We define the model, early-stopping broadcast, and transferable justifiers, and review STM.

3.1 The Model and the Task

There are n parties with identifiers in $[n] = \{1, \dots, n\}$, a public upper bound $0 \le t < n$ on the number of corruptions, and a designated sender $S \in [n]$ holding an input bit $b_S \in \{0, 1\}$. Throughout the paper we write

$$[n] = \{1, \dots, n\}, \qquad q = \lfloor n/2 \rfloor + 1, \qquad \sigma = \min\{t, \lfloor (n-1)/2 \rfloor\}.$$

The parameter q is the size of a strict majority of $[n]$, and σ is the maximum number of corruptions under which the honest parties still form a strict majority of at least q parties.

Network and rounds. The parties start in the same round 1 with a fresh common session identifier. Execution proceeds in synchronous rounds. In each round, a party first computes and sends its outgoing messages based on its state at the end of the preceding round, and then receives and processes the messages delivered in the current round. The parties are connected by a complete network of pairwise authenticated channels, so every recipient knows which party sent a message directly on a channel. A message that a party sends to all parties is also delivered to the party itself. All protocol choices break ties deterministically (for example, in favor of the smallest party identifier or the bit 0). A party terminates by producing its output and halting, after which it sends no further messages.

Corruptions. The adversary is rushing and corrupts parties adaptively and permanently over the course of the execution. In each round, the adversary may inspect the messages sent by honest parties before deciding which parties to corrupt in that round and which messages the corrupted parties send. When the adversary corrupts a party, it learns that party's internal state and signing key, and because it is *strongly adaptive*, it may also delete undelivered messages sent by newly corrupted parties in the round of their corruption. Let $F \subseteq [n]$ denote the set of all parties corrupted at any point during the execution, let $f = |F| \le t$ be the actual number of corruptions, and let $H = [n] \setminus F$ be the set of *honest* parties (those never corrupted). Every message sent between two parties in H is delivered in the round in which it is sent.

Plain PKI and idealized signatures. All cryptographic signatures use a plain public-key infrastructure (PKI): prior to the execution, each party $i \in [n]$ holds a signing key and all parties know the verification key of every party $j \in [n]$. Every signed message includes the common session identifier, the signer's identifier, the sub-protocol instance name, and the message type, alongside the message payload.

Following [28,13], we first analyze the protocol in the *idealized signature model*, in which signatures are treated as ideal unforgeable objects: a signature under the key of an honest party $i \in H$ is valid if and only if i has previously signed that exact message. Whenever our analysis uses a valid signature of an honest party $i \in H$ as evidence that i sent a message to all parties in some round, the protocol specifies that i creates that signature in the very round in which it sends the message; thus the adversary cannot obtain the signature before i's message is sent to all honest parties, even if i were corrupted later (which cannot happen for $i \in H$). In Section 3.2, we sketch the standard reduction showing that a violation of the idealized signature model yields an existential forgery.

Definition 3.1 (Early-stopping Byzantine broadcast). *Let $T(f, t)$ be a round bound. A protocol solves* early-stopping Byzantine broadcast *with stopping bound $T(f, t)$ if the following three properties hold in every execution with $f \le t$ corrupted parties:*

- ***Validity.*** *If the designated sender S is honest, every honest party outputs the sender's input bit b_S.*
- ***Agreement.*** *All honest parties output the same bit.*
- ***Termination.*** *Every honest party outputs and terminates—including the last round in which any honest party sends a message and the last round in which any honest party produces an output—by round $T(f, t)$.*

Transferable verification. A verification predicate for a message and a proof is *transferable* if a fixed message–proof pair that verifies at one honest party in an execution also verifies at every honest party in that execution [28,13]. This property concerns verification of the proof itself; the protocol may separately require the message to arrive in a particular round. We call the proofs used by STM *justifiers*.

3.2 Signatures and Unforgeability

Let λ be the security parameter. We use deterministic signatures that are existentially unforgeable under adaptive chosen-message attack (EUF-CMA), take $n = \mathrm{poly}(\lambda)$, and consider adversaries that run in probabilistic polynomial time in λ. Given the keys and inputs, the protocol is deterministic.

Lemma 3.2 (Reduction to EUF-CMA). *If a PPT adversary using the real signature scheme violates a protocol guarantee proved in the idealized signature model with probability $\varepsilon(\lambda)$, then a PPT EUF-CMA forger succeeds with probability at least $\varepsilon(\lambda)/n$.*

Proof. A violation requires an honest party to verify a signature under the key of some $j \in H$ on a message that j has not yet signed. Guess the signer in the first such verification uniformly from $[n]$. Use the challenge key for the guessed party, generate the other keys, simulate its signing with the signing oracle, and abort if it is corrupted. Whenever a simulated party verifies a signature under the challenge key on a message not yet queried to the oracle, output the message and signature if verification succeeds. This includes signatures inside stored or forwarded justifiers. If the guess is correct, the signer is never corrupted and the forger succeeds. The guess is correct with probability $1/n$ on each violating execution. The check is on the message queried to the oracle, regardless of the signature bytes; the polynomial work bound follows from Section 5. □

3.3 Send Transferable Message

We use the send transferable message protocol of Loss and Nielsen [28] in the accusation-graph formulation of Deligios, Klasovitá, and Liu-Zhang [13, Section 3.1]. An instance has a name ι, a designated sender $s \in [n]$, and an input predicate $J_{\mathrm{In}}(i, m, \pi)$, where i is the verifying party, m is a message, and π is its input justifier. The predicate is transferable: if it accepts (m, π) for one $i \in H$, it accepts the same pair for every $i \in H$. An honest sender starts with a pair that its own predicate accepts. The distinguished symbols NoMsg and $\perp$ are not input messages.

Signed messages and accusations. A signed message of instance ι is a tuple (m, η_s, π), where η_s is s's signature on m for this instance and message type, and π is checked separately by J_{In}. An accusation $\mathsf{Acc}(i, j)$ in instance ι, for $i \neq j$, is a signature by i on the instance and the ordered pair (i, j) for the accusation message type. Both signatures use the session and signer identifiers of Section 3. A forward retains the original signatures. Any round number on the enclosing message is the round of the forward, not the round in which those signatures were originally created.

For a set A of valid accusations in this instance, the accusation graph $G(A)$ has vertex set $[n]$ and contains the edge $\{i, j\}$ exactly when neither $\mathsf{Acc}(i, j)$ nor $\mathsf{Acc}(j, i)$ is in A. Thus adding accusations only removes edges [13, Remark 4]. We write $d_{G(A)}(i, s)$ for the length of a shortest path from i to s, with value ∞ when no path exists.

Definition 3.3 (Valid justified STM output). *For the instance $(\iota, s, J_{\mathrm{In}})$, a verifier i accepts the following forms:*

1. *$(m, (\eta_s, \pi))$ if η_s verifies as s's signature on m in this instance and $J_{\mathrm{In}}(i, m, \pi)$ accepts.*
2. *(NoMsg, A) if every entry of A is a valid accusation in this instance, there is at most one per ordered pair, and $d_{G(A)}(i, s) = \infty$. In this case A is a* polarizer *against s for i.*

No other form, including $\perp$, is a justified output.

These checks use the supplied object, independently of the verifier's local accusation set or the round in which it received the object. The protocol specifies separately when to act on a received message. If the designated sender is corrupted, honest parties can obtain different local outputs even though their justifiers are transferable.

Protocol. Party i starts with an empty accusation set A_i and an unset local output. It keeps at most one signature per ordered pair in A_i. The following steps specify its actions in protocol round u.

1. **Send.** In round 1, only the designated sender signs its input message and sends it to all parties with the input justifier. In a later round, if i set a justified output in the preceding round, it sends that output to all parties and then stops the instance. Otherwise, if it is still running, it signs the accusations computed in the preceding round, adds them to A_i, and sends A_i to all parties.
2. **Receive.** A party still running adds all valid accusations received for this instance to A_i, including those carried in forwarded polarizers. It then takes the first applicable action:
   (a) If it has received a signed message accepted by Definition 3.3, it fixes that message and its justifier as the local output.
   (b) If $d_{G(A_i)}(i, s) = \infty$, it fixes (NoMsg, A_i) as the local output. Receiving a polarizer from another party only supplies accusations; the disconnection test is made at i.
   (c) Otherwise, for every neighbor j of i in $G(A_i)$ with $d_{G(A_i)}(j, s) \leq u - 1$, it records the pair (i, j) to sign in the next send stage.

A party never changes its local output. If it still has no justified output after processing round $t+2$, it sets the output to $\perp$ and stops without forwarding. A justified output obtained in round $t + 2$ can still be forwarded in round $t + 3$.

Lemma 3.4 (Properties of STM). *Suppose the input predicate is transferable and the parties run the instance according to its schedule.*

1. *No honest party accuses another honest party. Every accusation graph contains the clique on H, and justified outputs verified by one honest party also verify at every honest party.*
2. *Every honest party obtains a justified local output by the end of protocol round $f + 1$. Honest local output rounds differ by at most one, and each honest party forwards its output in the next protocol round.*
3. *If s is honest, every honest party obtains s's message in protocol round 1. Any justified output accepted by an honest verifier is for that same message.*

Proof. For the honest clique, use the argument of [13, Lemmas 4–5], also described in Section 2.1. Suppose an honest party i first records an accusation against an honest party j in protocol round u. If s is honest, every honest party outputs in round 1. Otherwise, the only party that can be accused in round 1 is the faulty sender s. For $u > 1$, if j previously output a signed message, it forwards it by round u. The input predicate is transferable, so i outputs rather than accuses. If j previously output a polarizer, its forwarded accusations disconnect j from s in i's graph, so j is not eligible. Otherwise j sends its accusation set in round u, including accusations against the neighbors that were within distance $u - 2$ of s in its previous graph. After merging that set, i's graph contains no path of length at most $u - 1$ from j to s: any such path would begin with an edge removed by j's accusations. Thus i cannot accuse j.

The clique implies that in any valid accusation graph, either every honest party can reach s or none can. This gives transferability of the polarizers; for signed messages it follows from the input predicate. When s is honest, a polarizer cannot separate an honest verifier from s, and s signs only its input message [13, Lemmas 6 and 9].

The first honest output is sent in the next round. A signed message passes verification at every honest party; a forwarded polarizer disconnects every honest verifier after merging accusations. Thus every honest party still running outputs in that round. In particular, an honest output by round f suffices for all to output by $f + 1$. If s is faulty and no honest party has output by f, use the distance argument of [13, Lemma 10]: for each $h \in H$, its round-$(f + 1)$ accusation set cuts all paths of length at most f from h to s in its previous graph. Every honest party includes this set in its own graph, so every honest vertex has distance at least $f + 1$ there. Any shortest path from H to s, if one exists, has only faulty vertices after its first vertex and therefore at most f edges. Consequently all honest parties output in round $f + 1$ by the disconnection rule, unless they output a signed message in that round instead. □

The sender's instance. In STM_S, which starts in communication round 1, the designated sender is S. Its input predicate accepts exactly the pairs $(v, \emptyset)$ for $v \in \{0, 1\}$. For the rest of the protocol, a *sender justifier*

for v is a valid justified STM_S output whose signed message is v, or a valid justified NoMsg output if $v = 0$. A verifier uses Definition 3.3 for this check. Thus no justifier for bit 1 is obtained from NoMsg, and $\perp$ is never a justifier. Sender justifiers are transferable. If S is honest, every accepted sender justifier is for b_S.

4 The Broadcast Protocol

We run STM_S from Section 3.3 and a sequence of pairs of BA instances. Each party obtains a candidate bit with a justifier for that bit, then reports it. We use three deadlines:

$$D = 4(\sigma + 2), \qquad B = D + 4t + 7, \qquad E = B + t + 1.$$

A party that is still running at D sets its candidate by the end of that round. Parties accept report certificates directly through B and terminate by E.

Party i keeps a candidate bit and its sender justifier, an array ρ_i of n report values, and an output bit. Initially, the candidate and output bit are unset, the candidate justifier is absent, and every entry of ρ_i is $\perp$. Entry $\rho_i[j]$ is set to the bit in the first valid report certificate or chain that i accepts for report j, and an accepted bit never changes. The complete protocol below specifies the final assignment of NoMsg to entries that remain $\perp$.

Messages and local processing. Messages are specified by session, instance, and type. Each party discards invalid messages. We break ties between valid messages, justifiers, or certificates in lexicographic order.

4.1 Binary Byzantine Agreement

We use a three-round iteration inspired by the fault detection property of [20, Definition 4]. If $f \le \sigma$ and honest parties compute different next inputs, a new faulty party is added to the intersection of their faulty lists. The protocol runs the same steps for every $f \le t$; its agreement, validity, and fault detection guarantees below apply when $f \le \sigma$. In each BA instance, party i starts with a bit x_i, an empty faulty list $L_i \subseteq [n]$, and an unset BA output. The instance runs for $\sigma + 2$ iterations, each consisting of a Send, Vote, and Set round. The instance name and iteration number distinguish all messages and signatures of the instance. Each Vote or Set round message has at most one entry for each $j \in [n]$; absent entries represent no vote or no Set message, respectively. A vote identifies its voter and the party j being voted for.

One iteration. Let m be the iteration number. A party uses the input and faulty list it held at the start of the iteration. The following steps run in parallel for every party $j \in [n]$.

1. **Send.** If $x_j = 1$, party j signs 1 for this instance and iteration and sends it to all parties. If $x_j = 0$, it sends no signed bit for this iteration.
2. **Vote.** Party i signs and sends to all parties a vote for j if it received a valid signed 1 directly from j in the Send round and $j \notin L_i$. A valid vote identifies its voter and j, verifies under the voter's key for this instance and iteration, and includes j's valid signed 1 for that same instance and iteration. At the end of the Vote round, let $c_{i,j}$ count the distinct channel senders from which i received a valid vote for j in this round. Count a vote only if its signer is the authenticated sender of the channel on which it arrived.
3. **Set.** If $c_{i,j} \ge q$, party i sends a SET for j to all parties, containing the q counted votes with the smallest voter identifiers. If $c_{i,j} = 0$, it sends a ZERO marker for j to all parties. If $1 \le c_{i,j} < q$, it sends neither and adds j to L_i for subsequent iterations. A SET is valid if it contains q valid votes for j from distinct voters in the current instance and iteration. For votes embedded in a SET, verify each vote under its own voter's key; the voter need not be the sender of the SET.

At the end of the Set round, i sets $y_{i,j} = 1$ if it received a valid SET message for j in this round, and sets $y_{i,j} = 0$ otherwise. The value $y_{i,j}$ has *high grade* if it is 1 and i received valid SET messages for j directly from at least q distinct parties in this round, or if it is 0 and i received ZERO messages for j directly from at least q distinct parties in this round. Party i sets x_i for the next iteration to the majority of $(y_{i,1}, \ldots, y_{i,n})$, with ties broken in favor of 0.

Party i obtains *high grade on* v in iteration m if at least q entries $y_{i,j}$ equal v with high grade. The first iteration in which this occurs sets i's BA output to v. This output never changes, but i continues to update its input and faulty list and send the prescribed messages. At the end of iteration $\sigma + 2$, an unset BA output is set to $\perp$.

4.2 Choosing a Candidate

For each $e \in \{1, \ldots, \sigma + 2\}$, run two BA instances, one for each bit $v \in \{0, 1\}$. We call these two instances *pair* e. The BA inputs indicate whether a party has received a valid STM_S justifier for the corresponding bit; the inputs are not the candidate bits themselves. Party i keeps at most one justifier per bit for this pair, initially none. Iteration m of pair e uses communication rounds $e + 3m - 2$, $e + 3m - 1$, and $e + 3m$ for Send, Vote, and Set, respectively.

1. **Round e: inputs.** Each party that already has an output of STM_S at the start of round e sends it to all parties in the input field for pair e; otherwise this field is empty. This field is separate from the initial STM_S message and its forwards. At the end of round e, for each bit v, party i checks whether any input field for pair e contains a valid STM_S justifier for v. If so, it stores one and sets $x_{i,e,v} = 1$; otherwise it sets $x_{i,e,v} = 0$.
2. **Round $e + 1$: forwarding and startup.** For every v with $x_{i,e,v} = 1$, party i forwards its stored justifier for v to all parties in the corresponding field for pair e. This is also the Send round of iteration 1 in both BA instances. At the end of the round, i stores a valid forwarded justifier for each bit for which it did not already store one, if available. These new justifiers can be used for candidate selection but do not change the BA inputs already supplied.
3. **Completion.** Pair e completes at i once both BA outputs are set, including $\perp$ if the last iteration of an instance ends without the high grade. Iteration m ends in round $e + 3m$. If both BA outputs are bits, the completed pair selects the least v for which the instance for v output 1 and i stored a justifier for v. If there is no such v, or one output is $\perp$, the pair selects nothing. The selection for a completed pair is fixed.

Party i adopts the selected bit and a stored justifier of the least-indexed pair that selects a bit once that pair and all earlier pairs have completed at i. The pairs may complete in different orders; waiting for earlier pairs ensures that completion order does not determine the choice. The candidate is set only once. The last iteration of the last pair ends in round

$$D = (\sigma + 2) + 3(\sigma + 2) = 4\sigma + 8.$$

At the end of round D, after checking the pairs, any party that still has no candidate takes the bit supported by its own STM_S output, regarding NoMsg as bit 0, and uses that output as its sender justifier.

4.3 One Report per Party

Each report j uses a first-layer instance STM_j with designated sender j and a second-layer instance $\mathsf{STM}_{j,k}$ with designated sender k for every $k \in [n]$. The first layer supplies a justified STM output, which may be NoMsg. The second-layer outputs are used to construct a report certificate. This two-layer structure is adapted from the send transferable message with agreement (STMA) protocol of Loss and Nielsen [28]. We use Dolev–Strong signature chains [17] to agree on which report bits are accepted.

We use distinct signed instance identifiers for the initial STM, the first-layer instances, and the second-layer instances, for example $(\mathsf{initial}, S)$, (first, j), and (second, j, k). In particular, when $j = S$, the initial instance and the first-layer report instance are still distinct.

Constructing a Report Certificate

First layer: STM_j. If an honest party j sets its candidate to v in round T, it signs v for this instance and sends it to all parties with its STM_S justifier in round $T+1$. The input predicate requires a bit and a valid STM_S justifier for that bit. Through round D, each party that has not yet output in STM_j checks the signed input and forwarded signed messages for this instance. If any is valid, it sets its local output to one such message and schedules its forward in the next communication round. Through round D, no party generates or processes accusations for this first-layer instance. From communication round $D+1$ on, a party still running STM_j uses the STM rules with protocol round $u = r - D$ in communication round r. A forward already due in $D+1$ is still sent. An honest j sends its initial input only once.

Second layer: $\mathsf{STM}_{j,k}$ *for each* $k \in [n]$. Fix report j, and let a_i be the communication round immediately after party i obtains its local output in STM_j. In round a_i, party i starts its local part of all n second-layer instances for report j. Each instance has its own empty accusation set and unset local output at i. In $\mathsf{STM}_{j,k}$, the designated sender is k. Its input predicate accepts exactly the following:

1. a bit v with a valid signed message for v in STM_j;
2. the symbol $\mathsf{NoMsg}^{(j)}$ with a valid first-layer polarizer against j.

Honest k signs the form corresponding to its own output of STM_j and sends it in round a_k. Here $\mathsf{NoMsg}^{(j)}$ is a signed payload saying that k obtained NoMsg in the first layer. It is distinct from a second-layer output NoMsg, which is justified by a polarizer against k in $\mathsf{STM}_{j,k}$. In either layer, a verifier checks a polarizer relative to its own identity as in Section 3.3.

We adapt the staggering compiler of [28, Section 2]. For each protocol round $u \geq 1$ of $\mathsf{STM}_{j,k}$, party i sends in communication round $s_i(u) = a_i + 3(u-1)$ and processes the messages at the end of communication round $s_i(u)+1$. It uses messages for this instance and protocol round u delivered in $[s_i(u)-1,\, s_i(u)+1]$. Starting in communication round 2, party i stores messages tagged for protocol round 1, even if it has not yet started this instance. Once its starting round a_i is known, it keeps only messages received in communication rounds $a_i - 1$, a_i, or $a_i + 1$; messages received earlier or later are discarded. For each later protocol round u, it keeps only messages tagged for u and received in communication rounds $s_i(u)-1$, $s_i(u)$, or $s_i(u)+1$.

When an STM input is forwarded, it keeps the sender's original signature. The forwarding message is tagged with the protocol round in which it is forwarded. At the end of communication round $s_i(u)+1$, party i applies the STM receive step of Section 3.3 to the messages kept for protocol round u. It signs any new accusations only in its next scheduled send round. After sending the final forward for this instance, it takes no further actions in the instance.

Definition 4.1 (Valid report certificate). *A tuple* $(\pi_1, \ldots, \pi_n)$ *verifies at party* i *as a report certificate for report* j *and bit* $v \in \{0,1\}$ *if every* π_k *is a valid justified output of* $\mathsf{STM}_{j,k}$ *at* i *according to Definition 3.3, at least one output value is not* NoMsg*, and every output value other than* NoMsg *equals* v*. In particular, an entry with output value* $\mathsf{NoMsg}^{(j)}$ *prevents the tuple from certifying either bit.*

As soon as party i has all n local second-layer outputs for report j, it forms their tuple and checks whether it certifies a bit. Party i checks a received tuple as supplied, without substituting any of its local outputs. A failed check leaves $\rho_i[j]$ unchanged. The delivery rules below determine whether a tuple that verifies is accepted.

Agreeing on a Report

Direct acceptance (through round B*).* A party checks each certificate for report j that it constructs locally or receives, using Definition 4.1. On accepting its first valid certificate for that report, it records the bit and stores the certificate. If this happens in round $R < B$, it forwards the certificate to all parties in round $R+1$. A party that holds a certificate at the end of round B keeps it for the chain phase, whether or not it has already forwarded it.

Signature chains (rounds $B+1$ *through* E*).* For each report independently, run the signature chain rule of Dolev and Strong [17]. The chain carries a valid report certificate for a bit v and an ordered list of distinct signers $(p_1, \ldots, p_\ell)$. For each $k = 1, \ldots, \ell$, the signature of p_k is on the chain instance, report index j, bit v,

and the list $(p_1, \ldots, p_k)$. A party accepts a chain of length ℓ only in round $B + \ell$, for $1 \le \ell \le t + 1$, and only after checking the report certificate using Definition 4.1, checking that the signers are distinct, and verifying every signature.

Each party keeps a flag for each report, initially false, that records whether it has signed a chain for that report. In round $B + 1$, each party that kept a certificate starts a chain by signing a list containing only itself and sends it to all parties. In every chain round a party checks the incoming chains even if it already recorded the bit. It records the bit if none was recorded. If $\ell \le t$, the flag for j is false, and its name is not already on a valid chain's list, it schedules one such chain for extension. In the send stage of round $B + \ell + 1$ it appends its name and signature and sends the extended chain to all parties. It sets the flag whenever it signs, including when it starts a length-one chain, so it signs at most one chain per report. The final assignment of NoMsg to unresolved entries is specified in Section 4.4.

4.4 The Complete Broadcast Protocol

The protocol is given in Figure 2.

Instructions for party i in round r, if it has not halted

1. **Send.** Send the STM_S, BA, and report messages prescribed for round r.
2. **Choose a candidate.** Process the STM_S and BA messages. If the candidate is unset, take the bit and justifier from the first pair that selects a bit and for which all earlier pairs have completed. If it is still unset at the end of round D, take the justified output of STM_S (with NoMsg justifying 0). If a candidate is set in this step, schedule its first-layer report for round $r + 1$.
3. **Process reports.** Process STM_j and $\mathsf{STM}_{j,k}$ for every report. If $r \le B$, accept valid report certificates produced locally or received. If $B < r \le E$, accept valid chains of length $r - B$. Record each report's first accepted bit in ρ_i and schedule the forwards prescribed in Section 4.3. If $r = E$, set the remaining entries of ρ_i to NoMsg.
4. **Set the output.** If the output is unset and at least q entries of ρ_i equal the same bit, set the output to that bit. Otherwise, if the output is unset and no entry is $\perp$, set it to the least bit in ρ_i, or to 0 if no bit is present.
5. **Terminate.** If $r = E$ or the output was set in round $r - 1$, produce the recorded bit as output and halt.

Fig. 2. Instructions for party i.

If party i sets its output in round R_i, it follows all prescribed actions through round $\min\{R_i + 1, E\}$, including that round's send stage, and then halts.

4.5 Helper Lemmas

Throughout this subsection, a *full execution* is one in which every honest party follows its scheduled actions through round E. Parties set their outputs as prescribed, but do not halt the outer protocol early. The stopping rules of the individual subprotocols are unchanged. We relate full executions to actual executions in Section 4.6.

Lemma 4.2 (Early-deciding BA under an honest majority). *If $f \le \sigma$, all honest parties set the same BA output bit by the end of iteration $f + 2$. If they start with the same input bit, they set that bit as their BA output in the first iteration.*

Proof. We first show that no honest party is ever added to an honest party's faulty list. For this, fix $j \in H$. We show by induction that j never enters L_i for $i \in H$ and every honest i obtains $y_{i,j} = x_j$ with high grade. Initially j is outside every faulty list. If $x_j = 1$ in the current iteration, every honest party receives j's signed 1 and sends a vote. Every honest party therefore counts at least q votes and receives SET messages from at least q parties. If $x_j = 0$, no signed 1 of j exists for this iteration, so no valid vote or SET message for j exists.

Every honest party counts zero votes and receives ZERO messages from at least q parties. In both cases j remains outside every honest faulty list.

High grade. If an honest party obtains $y_{i,j} = 1$ with high grade, one of the q parties that sent a SET message to i is honest and sent the same valid message to all parties. Hence every honest party obtains 1 for j. If an honest party obtains $y_{i,j} = 0$ with high grade, one of the q senders of ZERO messages is an honest party that counted no vote for j. Any valid SET message for j must include a vote of an honest party, and that vote must have reached this sender in the Vote round. Thus no valid SET message exists for j, and every honest party obtains 0 for j. It follows that high grade on v at one honest party makes v the strict majority of every honest party's vector and thus their common next input. If the honest parties begin an iteration with a common input v, the $|H| \geq q$ honest coordinates give every honest party high grade on v and preserve the common input.

Fault detection and early decision. Suppose two honest parties i and i' compute different next inputs at the end of an iteration. Their vectors differ at some coordinate j, say with $y_{i,j} = 1$ and $y_{i',j} = 0$. The SET message received by i contains a vote from an honest party $h \in H$. That vote reached every honest party in the Vote round. No honest party counted q votes, since it would then have sent a SET message to i'. Thus every honest party counted between 1 and $q - 1$ votes and adds j to its faulty list. We must have $j \in F$, and j was not in L_h before the iteration, since h voted for j. Thus each such iteration adds a new element of F to $\bigcap_{h \in H} L_h$. By the end of iteration $f + 1$, all honest parties have the same next input v. In the next iteration, every honest party obtains at least q high-grade entries with value v, corresponding to the honest parties, and therefore fixes its BA output to v.

For agreement, consider the first iteration in which an honest party obtains high grade on a bit v. No honest party can obtain high grade on $1 - v$ in that iteration; otherwise the two grades would force opposite next inputs. If there is another iteration, all honest parties start it with v and obtain high grade on v from then on. Thus every honest BA output equals v. If the honest inputs already agree at the start, their common input has high grade in the first iteration. □

Lemma 4.3 (Common candidate under an honest majority). *If $f \leq \sigma$, all honest parties set the same candidate bit, each with a valid justifier for that bit, by round $(f + 2) + 3(f + 2) = 4f + 8$. For arbitrary $f \leq t$, every honest party sets a candidate with a valid justifier by round D.*

Proof. Suppose $f \leq \sigma$. Fix a pair e. By Lemma 4.2, each BA instance has a common output bit among the honest parties. If the instance for v outputs 1, at least one honest party had input 1: otherwise all honest inputs would have been 0, forcing output 0. That honest party forwarded a justifier for v in round $e + 1$. Transferability of STM_S justifiers means that every honest party stores a valid justifier for v in pair e. Thus all honest parties agree on whether pair e selects a bit and on the selected bit, if one exists.

By Lemma 3.4, an honest party obtains a justified output of STM_S by round $f + 1$. It sends the corresponding justifier to all parties in round $f + 2$, which exists as a pair index since $f \leq \sigma$. The BA instance for the bit of that justifier in pair $f + 2$ then has all honest inputs equal to 1 and outputs 1. In particular, pair $f + 2$ selects a bit. Both instances of each pair $e \leq f + 2$ complete by round $e + 3(f + 2) \leq 4f + 8$ by Lemma 4.2. Consequently, all honest parties adopt the bit of the same least-indexed selecting pair by round $4f + 8 \leq D$. For arbitrary $f \leq t$, a selection from any pair already comes with a valid justifier. Since $f + 1 \leq t + 1 \leq 2\sigma + 2 < D$, Lemma 3.4 guarantees that every honest party already has its own justified STM_S output when the fallback is invoked at round D. □

Lemma 4.4 (Transferability and uniqueness of the certified report bit). *A report certificate that verifies at an honest party is transferable. For each $j \in [n]$, all such certificates are for the same bit, and that bit has a valid sender justifier. If an honest party j sets its candidate to v by round $T \leq D$, then every honest party obtains a certificate for report j and bit v by round $T + 3f + 3$.*

Proof. Before round $D + 1$, any honest first-layer output is a valid signed message and is forwarded in the next communication round. If no honest party outputs by D, all honest parties run STM from $D + 1$. Together with Lemma 3.4, these rules ensure that honest parties never accuse one another in the first layer and that its justified outputs are transferable. Once an honest party outputs, all other honest parties output by the next round. Every honest party obtains a first-layer output by $D + f + 1$, so $a_i \leq D + f + 2$ and $|a_i - a_h| \leq 1$ for honest i, h.

The second-layer input predicate checks the justifier for the claimed first-layer output and is therefore transferable. For honest i, h and each protocol round u,

$$s_i(u) - 1 \le s_h(u) \le s_i(u) + 1 < s_h(u+1).$$

Thus every honest round-u message arrives in the prescribed receive window, and every honest party processes round u before any honest party signs a round-$(u+1)$ message. The inductive no-honest-accusation argument in Lemma 3.4 therefore applies to each completed second-layer receive step. In particular, every valid accusation graph in a full execution contains the clique on H. Thus first- and second-layer justified outputs are transferable, and so are valid report certificates.

To prove uniqueness, fix an honest second-layer sender k. A polarizer against k cannot verify at an honest party, because the honest clique connects that party to k. Hence every valid report certificate must contain k's signed second-layer message, which encodes k's first-layer output. Party k signs at most one such message. If it has not signed one, or if its payload is $\mathsf{NoMsg}^{(j)}$, no report certificate for a bit can verify. Otherwise, its payload is a bit v, and every valid report certificate must be for v. The nested first-layer message also contains a valid sender justifier for v.

For the timing bound, Lemma 3.4 gives all second-layer outputs at party i by $a_i + 3f + 1 \le D + 4f + 3$; their final forwards are sent by $D + 4f + 5 < B$. If honest j sets its candidate to v by $T \le D$, every honest party starts the second layer by $T + 2$. First-layer validity forces every second-layer output other than NoMsg to carry v, and every honest second-layer sender contributes v. Thus each honest party obtains a valid report certificate by $T + 2 + 3f + 1 = T + 3f + 3$. □

Lemma 4.5 (Agreement on reports). *In a full execution, all honest parties assign the same value in $\{0, 1, \mathsf{NoMsg}\}$ to each report $j \in [n]$. If $R \le E$ is the first honest acceptance round for a bit of report j, then all honest parties accept that bit by $\min\{R + 1, E\}$. If honest j sets its candidate to v by round $T \le D$, all honest parties accept its report with bit v by $T + 3f + 3 < B$.*

Proof. Every acceptance of a report bit is backed by a valid report certificate. By Lemma 4.4, no two honest parties can accept different bits for report j. Let $i \in H$ first accept the bit v in round R, if there is such a round. We consider three cases based on whether R is before, at, or after the deadline B for directly accepting report certificates.

- If $R < B$, party i forwards the transferable certificate in round $R + 1 \le B$. All honest parties accept the bit by that round.
- If $R = B$, party i starts a length-one chain in round $B + 1$. The chain and its certificate are valid at every honest party.
- If $R = B + \ell$ and the chain has an honest signer p_k, that signer created its signature only when sending the length-k chain to all parties in round $B + k$. Every honest party accepts the bit by that round, which is no later than R. If all signers are in F, then $\ell \le f \le t$ and $R < E$. Since i had not accepted the bit before R, it had not signed a chain for j before receiving this one. It is not a signer of the chain and extends it in the send stage of round $R + 1$. Every honest party accepts by that round.

If no honest party accepts a bit by E, each assigns NoMsg. Finally, Lemma 4.4 yields local certificates for the report of an honest j by $T + 3f + 3 \le D + 3t + 3 < B$. Local certificates are accepted directly through B. □

4.6 Main Proof

Proof of Theorem 1.1. We work first in the idealized signature model.

Agreement and validity. Fix an actual execution with final corruption set F. Let R be the first round in which an honest party sets its output, and put $L = \min\{R + 1, E\}$. Such a round exists because the decision rule applies to a complete report vector at E. All honest parties follow their prescribed actions through the end of round L. Construct a full execution with the same inputs and keys. Its adversary corrupts all of F at the outset and reproduces the messages and omissions of those parties through round L, then keeps them silent. By induction, the parties outside F send and receive the same messages in both executions through

L, so every signature under their keys used in the replay is available at the same stage. Their states agree through L apart from the outer halt status.

By Lemma 4.5, the full execution has a common final report vector $w \in \{0, 1, \mathsf{NoMsg}\}^n$. If an honest party sets its output using q matching reports, those reports remain a strict majority in w. If it uses a complete vector before E, every entry is already a bit and remains unchanged, so that vector equals w. Thus every output set in the full execution equals the result of the decision rule on w. By Lemmas 4.3 and 4.5, w contains an accepted report from at least one honest party. If S is honest, every bit in w has a valid sender justifier and equals b_S by Lemmas 3.4 and 4.4. If $R < E$, Lemma 4.5 ensures that every honest party accepts by round L the reports used by the first honest party to set its output. Those reports suffice to set an output: either they are q matching reports or they fill the whole report vector. If $R = E$, the decision rule sets all remaining outputs in round $L = E$. Thus all honest parties set their outputs within the shared prefix. Agreement and validity carry over to the actual execution, and every honest party halts by $\min\{L + 1, E\}$.

Number of rounds. For $f = 0$, STM_S outputs at the end of round 1. Both instances of pair 1 have input 0 at each party and finish in round 4. In round 2 every party supplies the sender's bit to pair 2, which selects that bit in round 5. The parties send their report inputs in round 6 and start the second layer in round 7. They process its first protocol round in round 8, accept all the reports, and set their output. They forward the accepted report certificates and halt in round 9.

For $1 \le f \le \sigma$, Lemmas 4.3 and 4.5 give every honest party at least q matching reports in the full execution by round $(4f + 8) + 3f + 3 = 7f + 11$. Every honest party therefore sets its output by $\min\{7f + 11, L\}$ in the full execution and hence in the actual execution. All honest parties halt by $\min\{7f + 12, E\}$.

For $f > \sigma$, we have $\sigma = \lfloor (n - 1)/2 \rfloor$ and $\sigma \le f - 1$. As $t \le n - 1 \le 2\sigma + 1$, it follows that $t \le 2f - 1$. The last round is at most

$$E = 4\sigma + 5t + 16 \le 4(f - 1) + 5(2f - 1) + 16 = 14f + 7.$$

In all cases the last output and the last send are by round $14f + 9$. Section 5 gives the communication, computation, and memory bounds. The reduction in Section 3.2 extends the result to EUF-CMA signatures except with negligible probability. □

5 Efficiency and Optimizations

We analyze the base protocol of Section 4 and then outline variants with better efficiency.

5.1 Efficiency

By Theorem 1.1, all honest parties output and halt by round $14f + 9$, including their last messages. We count bits sent by honest parties, charging a send to all parties once per recipient. Each identifier, instance name, and round number uses $O(\log n)$ bits, which is $O(\kappa)$ for signature length $\kappa > \log_2 n$. The common session identifier is implicit in transmitted messages but remains part of the signed contents.

Communication. By Lemma 3.4, every valid accusation has a corrupted endpoint, so an accusation set with at most one entry per ordered pair has $O(nf)$ entries. A justified STM output then has size $O((1 + nf)\kappa)$, including any nested justifier. The costs are as follows.

- Before broadcast halts, at most $O(f + 1)$ BA pairs start, and at most $O(f + 1)$ iterations of any one pair are executed. An iteration sends $O(n^2)$ signatures per channel, for $O(n^4(f + 1)^2\kappa)$ bits overall. The sender justifiers for the pairs take another $O(n^2(f + 1)(1 + nf)\kappa)$ bits.
- Only instances with a corrupted designated sender send honest accusations. There are $O(nf)$ such instances, and each sends an $O(nf\kappa)$-bit set on at most n^2 channels for $O(f + 1)$ rounds. This costs $O(n^4 f^2(f + 1)\kappa)$ bits. Initial messages and justified output forwards across all $O(n^2)$ instances cost $O(n^4(1 + nf)\kappa)$ bits.
- A report certificate or a chain carrying one has size $O(n(1 + nf)\kappa)$. Each party sends at most one direct certificate forward and one chain to all parties per report, for a total of $O(n^4(1 + nf)\kappa)$ bits.

The sum is $O(n^4(1 + nf + f^3)\kappa)$ bits, which is $O(n^7\kappa)$ since $f \le t < n$.

Memory and computation. With the message size limits of Section 4, each party receives at most $O(n^5\kappa)$ bits to process per round. Per sender, the $O(n^2)$ STM instances each use $O(n^2\kappa)$ bits, and the n reports each use $O(n^3\kappa)$ bits. The BA fields fit within this bound. Keeping the bounded transcript through $E = O(t+1)$ uses $O(n^5(t+1)\kappa)$ bits, in addition to key material and the workspace of the signature algorithms. Each party performs at most $O(n^5)$ signature operations and graph operations per round, using $O(n^2)$ graph operations per polarizer check. With $V(\lambda, n)$ the cost of one signature operation including encoding, this gives $O(n^5(t+1)(V(\lambda,n)+\kappa))$ work per honest party. This is polynomial in λ since $n = \mathrm{poly}(\lambda)$.

5.2 Optimizations

Table 2 lists the bounds for our variants; the round bounds include their last messages. This subsection outlines the changes and their role in the bounds.

Table 2. Summary of optimized protocol variants across resilience regimes.

Variant	Resilience	Rounds	Communication
Parallel BA instances	$t < n$	$5f + 2\lceil \log_2(f+1) \rceil + 10$	$O(n^5\kappa)$
Single BA instance	$t < n$	$5f + 2\lfloor \sqrt{f} \rfloor + 11$	$O(n^4\kappa + n^{9/2})$

Common optimizations. The two variants replace the three-round iteration of Section 4.1 by a d-detecting iteration [20]. Such an iteration uses d rounds for signature chains, one to vote, and one to determine grades. A vote requires a chain received on time. With an honest majority, an iteration that leaves different honest inputs adds at least d newly detected faulty parties to the common faulty list. If fewer than d undetected corrupted parties with a valid proof of participation remain, the iteration produces high grades. The variants also use accusations from STM_S: if its first honest output is in round r_0, those accusations identify at least $\max\{0, r_0 - 3\}$ corrupted parties other than S. The later BA iterations exclude these parties from their detection budget.

For reports, the variants use compressed justifiers and share an accusation graph so that each accusation needs to be sent only once, as in [28]. They allocate two communication rounds rather than three to each second-layer protocol round. They also replace the separate Dolev–Strong instances for all reports by one for a bundle of q matching reports and another for the sender's default. These changes affect the report time cap as well as the cost of sending and forwarding justifiers.

Parallel BA instances. Keep the ordered selection of BA pairs from Section 4.2. For pair e, the accusations reduce the budget on the sum of detection parameters to $\sigma - e + O(1)$. The first k parameters $d = 1, 2, 4, \ldots$ sum to $2^k - 1$ before truncating the last iteration to fit the budget. The voting and grading rounds therefore add $O(\log(f+2))$ rounds. Each proof of participation is sent once per pair.

Single BA instance. Start one BA instance concurrently with STM_S. A party starts without an input and adopts a bit when it obtains its STM_S justifier. Subsequent BA inputs retain transferable justifiers. Use $d = 1, 3, 5, \ldots$, truncating the last iteration. The first k parameters sum to k^2 before truncation; the two rounds per iteration for voting and grading therefore add $2\lfloor \sqrt{f} \rfloor + O(1)$ rounds.

Use of AI tools

ChatGPT-6 Astra assisted with the preparation of all figures, language editing, and checking technical claims and proofs in the manuscript. We also used it to check our protocol ideas and help determine the exact deadlines and timing of the protocol's primitives. ChatGPT helped develop the optimized variants in Section 5.2, and the authors independently verified those results. The authors developed all other new ideas and the proofs of all new results, and take responsibility for all claims in the paper.

References

1. Abraham, I., Dolev, D.: Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In: Proceedings of the 47th Annual ACM Symposium on Theory of Computing (STOC 2015). pp. 605–614. ACM (2015). https://doi.org/10.1145/2746539.2746581
2. Abraham, I., Nayak, K., Ren, L., Xiang, Z.: Good-case latency of Byzantine broadcast: A complete categorization. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing (PODC 2021). pp. 331–341. ACM (2021). https://doi.org/10.1145/3465084.3467899
3. Albouy, T., Frey, D., Raynal, M., Taïani, F.: Good-case early-stopping latency of synchronous Byzantine reliable broadcast: The deterministic case. In: Scheideler, C. (ed.) 36th International Symposium on Distributed Computing (DISC 2022). Leibniz International Proceedings in Informatics, vol. 246, pp. 4:1–4:22. Schloss Dagstuhl – Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.DISC.2022.4
4. Alexandru, A.B., Loss, J., Papamanthou, C., Tsimos, G., Wagner, B.: Sublinear-round broadcast without trusted setup. In: Proceedings of the 2025 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2025). pp. 4132–4171. SIAM (2025). https://doi.org/10.1137/1.9781611978322.141
5. Alpturer, K., Halpern, J.Y., van der Meyden, R.: Optimal eventual Byzantine agreement protocols with omission failures. In: Proceedings of the 2023 ACM Symposium on Principles of Distributed Computing (PODC 2023). pp. 244–252. ACM (2023). https://doi.org/10.1145/3583668.3594573
6. Bacho, R., Shi, K.: Early-stopping dishonest majority broadcast with optimal round complexity. Cryptology ePrint Archive, Paper 2026/2883 (2026), https://eprint.iacr.org/2026/2883
7. Bar-Noy, A., Dolev, D., Dwork, C., Strong, H.R.: Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. Information and Computation **97**(2), 205–233 (1992). https://doi.org/10.1016/0890-5401(92)90035-E
8. Berman, P., Garay, J.A., Perry, K.J.: Optimal early stopping in distributed consensus. In: Segall, A., Zaks, S. (eds.) Distributed Algorithms – WDAG 1992. Lecture Notes in Computer Science, vol. 647, pp. 221–237. Springer (1992). https://doi.org/10.1007/3-540-56188-9_15
9. Chan, H., Pass, R., Shi, E.: Sublinear-round Byzantine agreement under corrupt majority. In: Kiayias, A., Kohlweiss, M., Wallden, P., Zikas, V. (eds.) Public-Key Cryptography – PKC 2020. Lecture Notes in Computer Science, vol. 12111, pp. 246–265. Springer (2020). https://doi.org/10.1007/978-3-030-45388-6_9
10. Choi, W., Choi, W., Liu, X., Zikas, V.: On the round complexity of early stopping. Cryptology ePrint Archive, Paper 2026/1878 (2026), https://eprint.iacr.org/2026/1878
11. Coan, B.A.: Efficient agreement using fault diagnosis. Distributed Computing **7**(2), 87–98 (1993). https://doi.org/10.1007/BF02280838
12. Deligios, G., Elsheimy, F., Kamp, S.H., Klasovitá, I., Liu-Zhang, C.D., Loss, J., Papamanthou, C.: Optimal early-stopping broadcast via exponential information gathering. Cryptology ePrint Archive, Paper 2026/2785 (2026), https://eprint.iacr.org/2026/2785
13. Deligios, G., Klasovitá, I., Liu-Zhang, C.D.: Asymptotically optimal early termination for dishonest majority broadcast. In: Fehr, S., Fouque, P.A. (eds.) Advances in Cryptology – EUROCRYPT 2025. Lecture Notes in Computer Science, vol. 15605, pp. 116–144. Springer (2025). https://doi.org/10.1007/978-3-031-91092-0_5
14. Dolev, D., Lenzen, C.: Early-deciding consensus is expensive. In: Proceedings of the 32nd ACM Symposium on Principles of Distributed Computing (PODC 2013). pp. 270–279. ACM (2013). https://doi.org/10.1145/2484239.2484269
15. Dolev, D., Reischuk, R.: Bounds on information exchange for Byzantine agreement. Journal of the ACM **32**(1), 191–204 (1985). https://doi.org/10.1145/2455.214112
16. Dolev, D., Reischuk, R., Strong, H.R.: Early stopping in Byzantine agreement. Journal of the ACM **37**(4), 720–741 (1990). https://doi.org/10.1145/96559.96565
17. Dolev, D., Strong, H.R.: Authenticated algorithms for Byzantine agreement. SIAM Journal on Computing **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045
18. Elsheimy, F., Kamp, S.H., Loss, J., Papamanthou, C.: Near-optimal early stopping with optimal communication. Cryptology ePrint Archive, Paper 2026/2553 (2026), https://eprint.iacr.org/2026/2553
19. Elsheimy, F., Loss, J., Papamanthou, C.: Early stopping Byzantine agreement in $(1+\epsilon)\cdot f$ rounds. In: Chung, K.M., Sasaki, Y. (eds.) Advances in Cryptology – ASIACRYPT 2024. Lecture Notes in Computer Science, vol. 15489, pp. 398–424. Springer (2024). https://doi.org/10.1007/978-981-96-0938-3_13
20. Elsheimy, F., Loss, J., Papamanthou, C.: Towards optimal early stopping agreement protocols. Cryptology ePrint Archive, Paper 2025/246 (2025), https://eprint.iacr.org/2025/246
21. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Information Processing Letters **14**(4), 183–186 (1982). https://doi.org/10.1016/0020-0190(82)90033-3
22. Garay, J.A., Katz, J., Koo, C.Y., Ostrovsky, R.: Round complexity of authenticated broadcast with a dishonest majority. In: 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007). pp. 658–668. IEEE Computer Society (2007). https://doi.org/10.1109/FOCS.2007.44

23. Garay, J.A., Moses, Y.: Fully polynomial Byzantine agreement for $n > 3t$ processors in $t+1$ rounds. SIAM Journal on Computing **27**(1), 247–290 (1998). https://doi.org/10.1137/S0097539794265232
24. Hou, R., Yu, H.: Optimistic execution in Byzantine broadcast protocols that tolerate malicious majority. Journal of Parallel and Distributed Computing **209**, 105203 (2026). https://doi.org/10.1016/j.jpdc.2025.105203
25. Keidar, I., Rajsbaum, S.: A simple proof of the uniform consensus synchronous lower bound. Information Processing Letters **85**(1), 47–52 (2003). https://doi.org/10.1016/S0020-0190(02)00333-2
26. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Transactions on Programming Languages and Systems **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
27. Lenzen, C., Sheikholeslami, S.: A recursive early-stopping phase king protocol. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC 2022). pp. 60–69. ACM (2022). https://doi.org/10.1145/3519270.3538425
28. Loss, J., Nielsen, J.B.: Early stopping for any number of corruptions. In: Joye, M., Leander, G. (eds.) Advances in Cryptology – EUROCRYPT 2024. Lecture Notes in Computer Science, vol. 14653, pp. 457–488. Springer (2024). https://doi.org/10.1007/978-3-031-58734-4_16
29. Moses, Y., Rajsbaum, S.: A layered analysis of consensus. SIAM Journal on Computing **31**(4), 989–1021 (2002). https://doi.org/10.1137/S0097539799364006
30. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM **27**(2), 228–234 (1980). https://doi.org/10.1145/322186.322188
31. Perry, K.J., Toueg, S.: An authenticated Byzantine generals algorithm with early stopping. Tech. Rep. TR 84-620, Department of Computer Science, Cornell University (1984)
32. Srinivasan, S., Loss, J., Malavolta, G., Nayak, K., Papamanthou, C., Thyagarajan, S.A.K.: Transparent batchable time-lock puzzles and applications to Byzantine consensus. In: Boldyreva, A., Kolesnikov, V. (eds.) Public-Key Cryptography – PKC 2023. Lecture Notes in Computer Science, vol. 13940, pp. 554–584. Springer (2023). https://doi.org/10.1007/978-3-031-31368-4_20
33. Toueg, S., Perry, K.J., Srikanth, T.K.: Fast distributed agreement. SIAM Journal on Computing **16**(3), 445–457 (1987). https://doi.org/10.1137/0216031
34. Wan, J., Xiao, H., Shi, E., Devadas, S.: Expected constant round Byzantine broadcast under dishonest majority. Journal of the ACM **72**(4), 30:1–30:39 (2025). https://doi.org/10.1145/3748254