OmniBA: Round-Efficient BA with Quadratic Communication under Mixed Faults

Simon Holmgaard Kamp[4], Julian Loss[4], Kartik Nayak[2], and Kecheng Shi[1,3]

[1] CISPA Helmholtz Center for Information Security
[2] Duke University
[3] Saarland University
[4] Ruhr University Bochum

Abstract. We present a simple and efficient Byzantine Agreement protocol in the mixed fault model where up to t parties can be Byzantine, up to s parties can be send-omission, and up to r parties can be receive-omission such that $2t + s + r < n$. Our synchronous protocol greatly improves over the efficiency of the state-of-the-art solution due to Loss and Stern [TCC '23]. Specifically, our protocol incurs an expected communication complexity of $O(n^2)$ instead of $O(n^5)$ in their construction, while maintaining the same resilience. Our protocol terminates in an expected constant number of rounds, provided that a constant fraction of parties are non-faulty.

1 Introduction

Byzantine agreement (BA) is one of the most fundamental problems in the field of distributed computing. It has various applications in both distributed systems and cryptographic protocols, such as state machine replication (SMR), secure multi-party computation (MPC), verifiable secret sharing (VSS), and more. In the Byzantine Agreement problem, a set of n parties, each with an initial value, must agree on a common output value. The Byzantine Agreement problem has been studied extensively in the literature over the past years, under various network and adversary assumptions. Whereas most works in the area study BA protocols with fully malicious faults, we study the Byzantine agreement problem in a more fine-grained model of *mixed faults*. A model which distinguishes benign omission faults, for which some messages can be dropped by the adversary (but who honestly follow the protocol code) from malicious faults that can exhibit arbitrary behavior. This allows one to maintain meaningful security guarantees for the former type of fault. This model has recently seen increased popularity in the literature, as it provides a more realistic reflection of the types of faults that occur in real-world protocols while simultaneously offering some degree of security for parties who experience them, such as uniformity [19] and recovery [6].

The mixed fault model (as used in our work and other recent ones) was first introduced by Zikas et al. [22] who studied it in the context of error-free BA (and MPC). Concretely, they give protocols tolerating t Byzantine faults, r receive faults (whose incoming messages chosen by the adversary may be dropped) and s

send faults (whose outgoing messages chosen by the adversary may be dropped) such that $3t + r + s < n$. Much later, Eldefrawy et al. [10] revisited the mixed fault model in the cryptographic setting and proposed a protocol for the setting with $2t+s+r < n$ and a restricted type of send-omission fault. Protocols for the mixed fault model with full send-omission faults and security for $2t + s + r < n$ were shown by Loss and Stern [15] and Loss, Shi, and Stern [14].

Unfortunately, the communication complexity of the state-of-the-art protocol with resilience $2t+s+r < n$ [15] is quite expensive. Without threshold signatures, their protocol has communication complexity of $O(n^5)$. Even with threshold signatures, the protocol suffers from an inhibitive communication complexity of $O(n^4)$. Moreover, the good-case latency [3] of these protocols is not very satisfactory: their round complexity is high even if almost all of the parties are non-faulty. Motivated by the recent interest in protocols for the mixed fault model, we seek to answer the following open question:

– *Is it possible to achieve an efficient BA protocol in the mixed fault model for* $2t + s + r < n$ *and an adaptive adversary?*

Protocol	**Strongly adaptive?**	**Overlapping Faults?**	**Expected rounds complexity**	**Comm. complexity**	**Notes**
Loss–Stern [15]	✓	✗	27	$O(n^5)$	PKI
Loss–Shi–Stern [14]	✓	✓	31	$O(n^5)$	PKI
*Feng et al. [11]	✓	✓	44	$O(n^2)$	Threshold signatures
	✓	✗	40	$O(n^2)$	Threshold signatures
This work with $t + r + s < 3n/4$	✗	✗	32.2	$O(n^2)$	Threshold signatures, secure erasure.
This work with $t + r + s < n/2$	✗	✗	18.1	$O(n^2)$	Threshold signatures, secure erasure.

Table 1: BA Protocols with the optimal threshold $2t+s+r \leq n$, all protocols are adaptively secure. Our protocol has expected rounds complexity with at most $7.05n/t + 4$, see Theorem 3.

[4] * Feng et al. [Asiacrypt'25] is concurrent work.

We answer this by constructing a simple BA protocol against an adaptive adversary with expected $O(n^2)$ communication complexity and $7.05n/t + 4$ expected round complexity, given any constant fraction of non-faulty parties.

Concurrent Work. Independent of our work, Feng et al. [Asiacrypt'25] [11] studied consensus with omission faults and obtained optimal resilience even under overlapping corruptions. Their protocol further develops the established *weak consensus + common coin* paradigm. In contrast, our work departs from this paradigm and introduces a leader-based approach, which has better expected round complexity as long as the number of non-faulty parties $> n/5$ (which is guaranteed to be the case when $t + s + r < 4n/5$). We further note our protocol has much better good-case latency [3] in terms of a static adversary, which means when leader stays non-faulty, our BA protocol terminates within 9 rounds. A detailed comparison is given in Table 1.

1.1 Technical Overview

Receive-omission faults and send-omission faults characterize realistic failures, where messages may be dropped due to network failures. The adversary can selectively drop any messages sent to a receive-omission faulty party and selectively drop any messages sent from a send-omission faulty party. However, omission faults continue to run the correct protocol honestly and so we would like to be able to provide them with the same output guarantees as those for honest parties whenever they do manage to generate output. Moreover, we would like to take into account the inputs of parties who are neither send faulty nor Byzantine. This warrants special techniques that are very specific to the mixed fault setting, which we describe in the following.

Zombies and Ghosts. Following Zikas et al. [22] and Loss and Stern [15], we rely on the zombies and ghosts framework in which omission faults self-detect and then silence themselves from further participation in the protocol. This prevents them from causing additional harm to other participants or outputting incorrect values due to not receiving all relevant messages. The intuition is that self-detected receive-omissions become zombies who are entirely 'dead' with regards to the protocol, meaning that they send no further messages and do not output once the protocol completes for the parties who are still alive. Ghosts, on the other hand, are self-detected send-omissions who have also silenced themselves, but are still able to observe all relevant information and output consistently with the honest parties at the end of the protocol. To this end, the aforementioned prior works present specialized communication layers with self-detection over which the rest of the protocol can be run. While effective from a feasibility standpoint, these extra layers add a large amount of communication and rounds to the basic protocol logic, which results in protocols which are overall inefficient. Thus, at a technical level, our work's main contribution is to implement the above principles in a far more round-and communication efficient manner.

Core Protocol. We adopt VRF-based local leader election. In each iteration k, any party P_j with $(y_j, \pi_j) \leftarrow \mathsf{Eval}(sk_j, k)$ and $y_j < D$ multicasts a proposal including $t+r+1$ valid Status messages and proposing the value v_L of the highest-rank certificate among them. The parameter D is set (in Theorem 3) such that in expectation only a constant number of parties will send a proposal. Receivers deterministically select the valid proposal with the smallest verified y and send all votes to that triple (k, j, y_j). We argue that it happens with negligible probability that more than one valid proposal attains the same VRF evaluation y, through the property of VRF. Two lock steps use message-counting rules aware of the exact number of messages received to vote, culminating in a rank-k certificate formed from $t+1$ matching vote_3 messages for the same triple (k, j, y_j) (using t+1-out-of-n threshold signature schemes such as [7,4]). A lightweight ghost-check prevents a send-omission party from committing without ensuring propagation to at least one honest non-send-omission party. Finally, a notify aggregation ensures termination of all non-faulty and send-omission parties after a good iteration. We start by introducing our core protocol, which is designed to ensure agreement/safety of the Byzantine Agreement, while achieving a weaker form of validity and termination. Specifically, it achieves:

- **Termination.** All non-faulty parties and send-omission faulty parties eventually commit.
- **Agreement/Safety.** All non-Byzantine parties (including omission faults) do not commit on conflicting values.
- **Validity.** If all non-faulty parties and all send-omission faulty parties start with certificates, which is a quorum of signatures from $t+r+1$ distinct parties on their initial inputs for the same value v (that is regarded as a special certificate with rank 0), and no Byzantine party starts with a certificate of a different value v', all non-faulty parties and all send-omission faulty parties commit on v.

Consistency Checks. Similar to classic BA protocols, we need to prevent parties from committing to different values. Because of the unique characteristics of the omission-fault model, ensuring consistency with a simple all-to-all round of echoing round is insufficient. For now, we only focus on the security and we assume each party will not vote in the current iteration once it has seen multiple valid proposals (and it will multicasts conflicting headers). This would cost more iterations to terminate, and we will illustrate how to improve it later by breaking ties on VRF evaluations. In line with prior protocols such as those of Loss and Stern [15], we therefore count messages that a party has received in a more precise manner, to determine more accurately how many of the messages received come from send-omission or Byzantine parties (or receive-omission and non-faulty parties). Our design also ensures that the impact of send-omission faults on the execution of the protocol is limited and allows us to build a novel two-lock-step mechanism. Specifically, we consider two cases: (i) if a party receives strictly fewer than $n - s$ messages in total, and at least $n - t - s - r$ of them are $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$ with $v \neq \perp$, it proceeds; (ii) if a party receives

$n-s+x$ messages in total (for $x \geq 0$), and at least $n-t-s-r+x$ of them are $(v,\ \mathsf{vote}_1,\ (k,j,y_j))$ with $v \neq \perp$, it proceeds. At a high-level, this ensures at least one receive-omission faulty party or a non-faulty party must have voted for it, and its message will arrive at all non-faulty parties and all send-omission faulty parties. As a result, they cannot vote for a different value v' at the end of lock_{NS} or $\mathsf{lock}_{A}LL$. In the second lock-step, since each receive-omission faulty party must receive messages from at least $t+r+1$ distinct parties if it does not become a zombie, at least one of them must be a send-omission faulty party or a non-faulty party. Thus, if all send-omission faulty parties and all non-faulty parties can detect different valid proposals, all parties either vote for the same proposal or do not vote at the end of the second lock-step. In summary, our mechanism ensures that after two lock steps, there is no different votes between any two non-Byzantine parties, namely:

- In our first lock step ($\mathsf{lock}_{\mathsf{NS}}$), we can ensure that all send-omission faulty parties and all non-faulty parties cannot set $\mathsf{lock}_{\mathsf{NS}}$ to different values and send vote_2 for different values.
- In our second lock step ($\mathsf{lock}_{\mathsf{ALL}}$), we additionally ensure that no non-Byzantine parties set $\mathsf{lock}_{\mathsf{ALL}}$ (or send vote_3) to different values $v, v' \neq \perp$.

Based on the aforementioned consistency checks, parties can form a certificate consists of $t+1$ signatures of vote_3 in the Certificate Generation phase, and the security requirement that no certificates with different values are generated in the same iteration is achieved. If the unique leader is non-faulty or is a receive-omission faulty party that is alive, its proposal must reach all non-faulty parties and all send-omission faulty parties, and the iteration succeeds, which means all non-faulty parties must commit in this "good iteration".

Ghost-Check. It is problematic if a send-omission faulty party commits on the value v, and it fails to deliver the commit message to all other parties. Indeed, this could result in the remaining parties simply committing to a different value in the next iteration, which would lead to the parties outputting inconsistently. To address this issue, we add a special check process (called Ghost-Check) before any party commits. Much like the Ghost-Check introduced by Loss and Stern in their work [15], this mechanism ensures that a party only commits once it knows that at least one receive-omission or non-faulty party has already received its certificate and can propagate it to all non-receive faulty parties by the time the next iteration has started. However, in contrast to Loss and Stern's implementation of this mechanism: since the detection of receive-omission faulty parties is not necessary in our Ghost-Check process, our Ghost-Check is more round-efficient and also does not take up more than $O(n^2)$ total communication per iteration, collectively, over all parties performing it.

The Need for notify. Since we force each party to commit if it has passed the ghost-check, a new issue arises. Because send-omission faulty parties are required to terminate and output the correct value if the adversary causes them to fail the ghost-check, they may never commit. To help those send-omission faulty parties to commit, we add an additional notify mechanism, where for each party that has committed, multicasts a notify message to all other parties. Each party commits on v if it has received $t+1$ notify messages for the same value v (whether $t+1$ notify messages come from a threshold signature or from $t+1$ distinct parties directly). This ensures that if there exists exactly one non-faulty leader in the current iteration, all non-faulty parties and send-omission faulty parties will terminate in the next iteration (or earlier).

Dealing with Zombies. Even in an iteration with an honest leader, receive-omission faulty parties may neither commit on the value nor become a zombie. We notice that after all non-faulty parties have committed and sent notify message to all other parties, all non-faulty parties and all send-omission faulty parties will terminate. For those receive-omission faulty parties, either they receive combined $t+1$ notify messages from any party that is going to terminate, or they have to become a zombie in the next iteration since all non-faulty parties and send-omission faulty parties have terminated.

Improving the Round Complexity. To elect leaders in each iteration using VRF with the rule that parties will not for any value if they have seen different valid proposals suffices to achieve quadratic communication complexity, but the overall round complexity is not satisfactory. The protocol would only terminate after a "good iteration" where a non-faulty or a receive-omission faulty party that is alive becomes the unique leader. To improve the concrete round complexity, we introduce a tie-break mechanism, where each party only votes for a valid proposal with the smallest VRF evaluation. This means each party can only vote for a proposal with the (locally) smallest VRF evaluation y_j, so that we are in a "good iteration" as long as the eligible party with the lowest (unique) evaluation is non-faulty or a receive-omission faulty party that is alive. Now we need to ensure security is still held within each iteration. At a high level, each party must multicast the triple (k, j, y_j) with the lowest VRF evaluation y_j to all other parties if it does not vote for (k, j, y_j); any party that receives this message will not vote for any valid proposal with a different value v' and VRF evaluation $y'_j \geq y_j$.

1.2 Related Work

After being proposed by Lamport et al. [13], the agreement problem has been studied extensively. We introduce related works from two perspectives: BA in the mixed fault model and BA protocols focusing on communication complexity. Studies about the mixed fault model have a long history, both in the computational setting and the information-theoretic setting. In the information theoretic-setting, Perry and Toueg [18] provided a generic construction for any number of send-omission faults, without uniformity. Garay and Perry [12] showed Byzantine Agreement is possible with c crash faults and t Byzantine faults if $n > 3t+c$. Siu, Chin and Yang [21] built Byzantine agreement protocol as long as $n > 3t+k$, for any k non-malicious faults. More recently, Zikas et al. [22] constructed consensus with resilience $3t + s + r < n$, and studied omission faults in the field of multiparty computation (MPC). Very recently, Brazitikos and Zikas [8] fully characterized the feasibility in the presence of full-omission faults and Byzantine corruptions in the general adversary structure. Loss et al. [14] demonstrated that uniform consensus is achievable even with $s + r = n$ when $(s < n)$. As for the computational setting, Abraham et al. [2] constructed a protocol with Byzantine corruptions and crash faults under the constraint $n > 2t + c$. Eldefrawy et al. [10] constructed a protocol for Byzantine and omission faults in the constraint $n > 2t+2s+r$ without the spotty assumption, and with $n > 2t+s+r$ under the spotty assumption. Loss and Stern [15] constructed a protocol with the resilience condition $2t + s + r < n$ without the spotty assumption. Very recently, Loss et al. [14] built upon the protocol of Loss and Stern [15] to tolerate full-omission faults, where a party can be both a send-omission and a receive-omission, albeit with increased round and communication complexities. In a novel independent and concurrent work [11], Feng et al. build a first Byzantine Agreement protocol under the optimal resilience $2t + s + r < n$ with quadratic communication complexity and expected constant round with a strongly adaptive adversary that can make a party be full-omission, a detailed comparison table is provided in Table 1.

2 Preliminaries

2.1 Adversary Model

We consider an adversary to be weakly adaptive, which means it can choose which party to corrupt at any point of the execution of the protocol. By "weakly adaptive", we mean that if the adversary chooses to corrupt a party P_i after P_i has already sent its message in round k, it cannot affect the delivery of messages sent from P_i to all other parties. We note that the adversary can send additional messages on behalf of P_i after it corrupts it in round k. We assume there exists a constant fraction of non-faulty parties. The adversary can cause each party to become one of the following three types of corruption:

- **Send-Omission Corruption.** The adversary can choose to drop any messages sent from a send-omission corrupted party. But this party honestly follows the protocol code. We denote the set of send-omission faulty parties by $\mathcal{S}$ and the maximum number of send-omission corruptions by s.
- **Receive-Omission Corruption.** The adversary can choose to drop any messages sent to a receive-omission corrupted party. But this party honestly follows the protocol code. We denote the set of receive-omission faulty parties by $\mathcal{R}$ and the maximum number of receive-omission corruptions by r.
- **Byzantine Corruption.** The adversary gains full control over a Byzantine corrupted party. We denote the set of Byzantine corrupted parties by $\mathcal{B}$ and the maximum number of Byzantine corruptions by t.

We denote the full set of n parties $\mathcal{P}$ and the remaining uncorrupted parties $\mathcal{N} = \mathcal{P} \setminus \mathcal{S} \setminus \mathcal{R} \setminus \mathcal{B}$ as non-faulty parties. In this work, we always assume the maximum number of each corruption type satisfies the constraints $2t+s+r < n$, as in the state-of-the-art protocol [15].

2.2 Network Model

Our Byzantine Agreement protocol works in the synchronous setting, where all messages sent from a non-faulty party (or a receive-omission faulty party), if not dropped (which can happen when messages are sent to a receive-omission faulty party), are guaranteed to be delivered before the next round begins. We assume a complete n-to-n network, where each pair of parties is connected via an authenticated channel.

2.3 Cryptographic Tools

Signature Schemes. Our model employs several standard cryptographic primitives. For signing proposals, we utilize a forward-secure digital signature scheme [16] for signing the proposals. In the KeyGen, each party is given the constant public key pk and the initial secret key $sk^{(0)}$. Each party can run Update with the current iteration period k and current secret key $sk^{(k)}$ to get the next secret key $sk^{(k+1)}$. It outputs the secret key for the next iteration $k + 1$ as $sk^{(k+1)}$, and $sk^{(k)}$ is erased through the updating procedure $sk^{(k+1)} \leftarrow \mathsf{Update}(k, sk^{(k)})$. Each party gets its signature for signing a proposal in iteration k as $(\sigma, k) \leftarrow \mathsf{Sign}(k, sk^{(k)}, M)$. The signature is verified through $\{\mathsf{True}, \mathsf{False}\} \leftarrow \mathsf{Verify}(pk, M, (\sigma, k))$. We implicitly assume that parties always run Update in each iteration to prevent adversary from adaptively corrupting parties that are eligible to propose, where the adversary can get its signatures on its equivocated proposal. The forward-security property ensures that a compromise of the secret key for the current period does not allow an adversary to forge signatures for any prior periods. We also employ threshold signature schemes (e.g., [20,4,5]) with trusted setup, instantiated with thresholds $t+1$ and $t+r+1$.

Verifiable Random Function (VRF). In addition, we rely on a verifiable random function (VRF) [17,9]. For party P_j and iteration k, the VRF is evaluated as $(y_j, \pi_j) \leftarrow \mathsf{Eval}(sk_j, k)$, and verified by $\mathsf{Verify}(pk_j, k, y_j, \pi_j)$, the VRF outputs are of size security parameter κ. Each party P_j is *eligible* to propose in iteration k iff $y_j < D$ for a publicly fixed difficulty $D = p \cdot 2^{\ell_{\mathsf{VRF}}}$ where ℓ_{VRF} is the number of bits in the VRF output, p is each party's probability of proposing, and $n \cdot p$ is the expected number of proposers. The choices of c is described in the proof (given in the Appendix B) of Theorem 3) Multiple eligible proposers may exist; each receiver processes only the valid proposal with the *smallest* y it observed. We note it happens with negligible probability for different eligible parties to attain the same VRF evaluation.

3 Core Protocol

3.1 Core Ideas

Before jumping into our core protocol, it is good to know the core protocol presented in [1]. It is a leader-based Byzantine Agreement protocol that achieves security, termination and a weak form of validity. Their core protocol consists of elect, status, propose, commit, notify. In the elect phase, a leader is elected. Then each party sends its status messages (which includes its stored certificate) to the leader in the status phase. Next, the leader sends its proposal in the propose phase based on status messages it has received. Later on, each party votes for the proposal to all other parties if the value leader proposed is consistent with a certificate of rank no lower than the current certificate of the party. Subsequently, each party commits on the value based on a certificate formed by $t+1$ votes when no equivocations have been seen. The reason it works is that at least one of these $t+1$ parties is honest and forwards the message to all other honest parties. So, if two conflicting certificates exists, all parties learn about an equivocation and refrain from committing. Finally, each party sends a notify message if they have committed. Now we highlight the difficulties that arise when adapting this approach in the presence of omission faults:

- In the status phase, send-omission faulty parties may not vote for proposal proposed by a non-faulty leader who has a lower certificate, as its status message may be dropped. In addition, a send-omission faulty party that has committed may not deliver its status message to other parties. Other parties may commit on a different value proposed by subsequent leaders.
- In the vote phase, if we simply set the vote threshold to be $t+r+1$ or $t+s+r+1$, liveness does not hold because a send-omission faulty party may not deliver its vote to other parties, and a receive-omission faulty party may not vote (as it may not receive a proposal) even if the leader is non-faulty. If we set the vote threshold to be $t+1$, receive-omission faulty parties may vote for a different value as they can drop messages sent from honest parties, and vote for different values together with t Byzantine parties. As a result, certificates with different values may be generated, which would break the consistency.
- The notify mechanism may not work for receive-omission-faulty parties, as they may drop notify messages. Also send-omission faulty parties may not be able to send notify messages.

At a high level, our protocol deals with these issues as follows:

- In the propose phase, the proposal must contain at least $t+r+1$ Status messages, which must include at least one message from a send-omission faulty party or a non-faulty party. This design separates send-omission faulty parties' messages, so that they must vote for a non-faulty eligible proposer with minimal y. We note that although their vote messages can be dropped, it is important for us that they "tried to vote", so that it is difficult for a Byzantine leader to let honest parties vote for different values. Any send-omission faulty party that is going to commit must pass our refined ghost check phase inspired from the weak multicast protocol of [15]. Where each party only commits if it has delivered its certificate to at least one receive-omission faulty party or a non-faulty party. Our ghost-check is more efficient as we avoid the expensive n-to-n communication of [15] as we do not detect

receive-omission faults for this process. And the receive-omission faulty party or a non-faulty party must have delivered its certificate to all other non-faulty parties and all send-omission faulty parties before the status phase.
– In the vote phase, we design a novel two-lock-step mechanism, where all parties must vote not only based on the number of valid vote messages for the same value, but also on the exact number of messages they received in the previous round. This helps distinguish between send-omission faulty parties and equivocated Byzantine parties, which we will specify later.
– Similar to the notify mechanism described in [1], we can achieve the termination property that all non-faulty parties and send-omission faulty parties will terminate after a non-faulty eligible proposer with minimal y. To force receive-omissions either to terminate or become a zombie, the solution is quite simple and efficient: after all non-faulty parties and send-omission faulty parties have terminated, they will not participate in later iterations. Then all receive-omission faulty parties either receive at least one message from among all of the non-faulty parties or send-omission faulty parties, or they can only receive at most $t + r$ many messages in following iterations and thus become a zombie.

We present a leader-based synchronous Byzantine Agreement protocol secure against t Byzantine parties, s send-omission faulty parties, and r receive-omission faulty parties for n parties where $n > 2t + s + r$. The protocol guarantees that

– All non-faulty parties and all send-omission faulty parties eventually commit on the same value.
– There is no non-Byzantine party that commits on a different value.

Our core protocol also achieves a weak form of validity, meaning that if all of the following conditions are satisfied, all non-faulty parties and send-omission faulty parties will commit on the same value v:

– All non-faulty parties and send-omission faulty parties start with certificates for value v.
– The adversary does not start with a certificate for a different value $v' \neq v$.

In the Appendix C.1, we introduce how to initiate our *Propose* phase with VRF and erasure. We also provide an adaptive security argument in the Appendix C.2. In section 4, we show how to obtain the initial certificate by constructing a simple sub-protocol, which helps to achieve validity. In addition, we also introduce our design of notify and Terminate to ensure that all send-omission and non-faulty parties must terminate after a non-faulty eligible proposer with minimal y, and each receive-omission faulty party P_i either outputs a consistent and valid value or detects it is a zombie (where it outputs $Z_i = \mathsf{True}$ instead). We also provide a communication complexity analysis and a round analysis at the end of section 4.

3.2 Notation

When we say that party P_i multicasts to all parties, it means that party P_i sends messages to all parties, including itself (which must arrive regardless of the behavior of the adversary). We note that each party P_i accepts a message m from party P_j only if the signature is correct. A status message of P_i is $\langle\, k,\ \mathsf{Status},\ \mathsf{Cert}_i,\ v_i \,\rangle_i$, where k is the iteration, Cert_i is either $\perp$ or a rank-k' certificate consisting of a $t{+}1$ threshold signature on $(v,\ \mathsf{vote}_3,\ (k', j, y_j))$ for some $v \neq \perp$, and v_i is the value of Cert_i (or $\perp$).

A proposal by P_j in iteration k is $\left\langle (y_j, \pi_j),\ k,\ \mathsf{proposal},\ \{\mathsf{Status}_\ell\}_{\ell\in J},\ \langle v_L\rangle_j \right\rangle_j$, where $|J| = t{+}r{+}1$, $\mathsf{Verify}(pk_j, k, y_j, \pi_j) = \mathsf{True}$, and v_L is the value carried by the highest-rank certificate among the bundled Status messages. If no certificate of rank ≥ 0 appears, P_j may choose v_L freely. We note that signing keys for the proposals are updated in each iteration using the aforementioned forward-secure signature scheme without further specification.

Processing with the locally lowest y. In iteration k, each party fixes the *accepted triple* (k, j, y_j) to be the valid proposal it received with the lowest y. If the minimizer is unique, the party may cast a non-$\perp$ vote in rounds $\ell \in \{1, 2, 3\}$ *only* for that triple; different valid proposals from the same sender are treated as an equivocation and the party casts only $\perp$ (and multicasts the conflicting headers as specified). $\left\langle \langle v\rangle_j,\ \mathsf{vote}_\ell,\ (k, j, y_j)\right\rangle_i$.

Counting convention. When we require "at least M messages of the form $(v,\ \mathsf{vote}_\ell,\ (k, j, y_j))$," all M messages must carry the *same* triple (k, j, y_j) and the same $v \neq \perp$.

3.3 Detailed Core Protocol

We are ready to introduce our core protocol, the details are shown in Figure 1.

A Leader-Based Protocol in an iteration k

Input and Initialization: Each party P_i, $i \in [n]$ starts with input Z_i and Cert_i (for value v_i). Parties execute the following steps:

1. **Status.** For party P_i:
   If $Z_i = \mathsf{False}$, P_i multicasts a status message $\langle k, \mathsf{Status}, \mathsf{Cert}_i, v_i\rangle_i$ to all other parties to report its current status. If $Z_i = \mathsf{True}$, P_i multicasts $\langle Z_i = \mathsf{True}\rangle_i$ to all other parties instead. All parties update their current certificate to be the one with highest rank. If party P_i has committed on value v, send $\langle \mathsf{notify}, v\rangle_i$ to all other parties.
2. **Propose.** If $Z_i = \mathsf{False}$, party P_i computes $(y_i, \pi_i) \leftarrow \mathsf{Eval}(sk_i, (k, i))$. If $y_i < D$, P_i signs a proposal.
   $$\Big\langle (y_i, \pi_i),\ k,\ \mathsf{proposal},\ \{\mathsf{Status}_\ell \mid \ell \in J,\ J \subseteq [n],\ |J| = t{+}r{+}1\},\ \Big\langle v_L = v_{\arg\max_{\ell\in J} R_\ell}\Big\rangle_i \Big\rangle_i$$
   to all parties, where R_ℓ is the rank of the certificate contained in Status_ℓ and v_L is the value of the highest-rank certificate among those $t{+}r{+}1$ Status messages. If no certificate has rank > 0, P_i may choose v_L freely. Each party P_i then deletes its signing key for iteration k, multicasts the proposal to all other parties.

3. **Forward.** (vote_1) Let Q be the set of indices q from whom P_i received a valid proposal in this phase. If $Q \neq \emptyset$, set

$$j \leftarrow \arg\min_{q \in Q} \{ y_q \}$$

Party P_i multicasts $\left\langle \langle v_{j \to i} \rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j) \right\rangle_i$ iff all of the following hold for the proposal from P_j:
   - $\mathsf{Verify}(pk_j, k, y_j, \pi_j) = \mathsf{True}$;
   - $v_{j \to i} \neq \bot$;
   - the proposal includes at least $t{+}r{+}1$ valid Status messages, and the value of the highest-ranked certificate among them equals $v_{j \to i}$;
   - $y_j < D$.

   If no such valid minimum-y proposal exists, P_i multicasts $\langle \bot \rangle_i$.
4. **First Lock Step.** (vote_2) For party P_i:
If P_i has received strictly fewer than $n - t - s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If there exist two valid proposals with different values v and v' for the same (locally) lowest VRF evaluation, P_i multicasts conflicting headers to all other parties.
   - If no equivocation is detected and if one of following conditions a) and b) is satisfied, P_i sets $\mathsf{lock}_{\mathsf{NS}} = v$ and sends $\left\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \right\rangle_i$ to all parties:
   a) If party P_i receives strictly fewer than $n - s$ messages in which at least $n - t - s - r$ of them are $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$ for $v \neq \bot$.
   b) If party P_i receives $n - s + x$ (where $x \geq 0$) messages in which at least $n - t - s - r + x$ of them are $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$ for $v \neq \bot$.
   - Otherwise, P_i sets $\mathsf{lock}_{\mathsf{NS}} = \bot$ and sends triples $\langle k, j, y_j \rangle$ for the lowest VRF evaluation y_j (if there is no such valid triples, sends $\langle \bot \rangle_i$) to all other parties.
5. **Second Lock Step.** ($\mathsf{lock}_{\mathsf{ALL}}$) For party P_i: If P_i has received strictly fewer than $n - t - s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If an equivocation (with (locally) smallest VRF evaluation) is detected by P_i, P_i multicast conflicting headers. Otherwise, if P_i has not received a valid proposal from P_j in the proposal phase for the (locally) lowest triple of (k, j, y_j) received in the forward phase, P_i sets $\mathsf{lock}_{\mathsf{ALL}} = \bot$ and multicasts $\bot$ to all other parties.
   - If no equivocation is detected by P_i, and P_i has received the valid proposal with the same triple (k, j, y_j) with the (locally) smallest y_j, and if one of following conditions a) and b) is satisfied for party P_i, P_i sets $\mathsf{lock}_{\mathsf{ALL}} = v$ and sends $\left\langle \langle v \rangle_j,\ \mathsf{vote}_3,\ (k, j, y_j) \right\rangle_i$ to all parties:
   a) If P_i has received strictly fewer than $n - s$ messages in which $n - t - s - r$ of them are $(v,\ \mathsf{vote}_2,\ (k, j, y_j))$ for $v \neq \bot$.
   b) If P_i has received $n - s + x$ (where $x \geq 0$) messages in which $n - t - s - r + x$ of them are $(v,\ \mathsf{vote}_2,\ (k, j, y_j))$ for $v \neq \bot$.
   - Otherwise, P_i sets $\mathsf{lock}_{\mathsf{NS}} = \bot$ and sends triples $\langle k, j, y_j \rangle$ for the lowest VRF evaluation y_j (if there is no such valid triples, sends $\langle \bot \rangle_i$) to all other parties.

6. **Certificate Generation.** For party P_i: If P_i has received strictly fewer than $n-t-s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If one of the following conditions a) or b) is satisfied for party P_i, then P_i aggregates messages of the form $\langle\langle v\rangle_j,\ \mathsf{vote}_3,\ (k,j,y_j)\rangle$ (same v and same triple (k,j,y_j)) from any $t{+}1$ distinct parties into a certificate C_i using a $t{+}1$-out-of-n threshold signature scheme. P_i sets $\mathsf{Cert}_i = C_i$ and multicasts Cert_i to all other parties:

     a) P_i has received strictly fewer than $n-s$ messages, of which at least $n-t-s-r$ are $(v,\ \mathsf{vote}_3,\ (k,j,y_j))$ with $v \neq \perp$.

     b) P_i has received $n-s+x$ messages (for some $x \geq 0$), of which at least $n-t-s-r+x$ are $(v,\ \mathsf{vote}_3,\ (k,j,y_j))$ with $v \neq \perp$.
7. **Ghost-Check and Commit.** For each party P_i, P_i sets its local certificate Cert_i to be the certificate C_i^* with the highest rank.
   - If P_i did not receive a valid certificate from party P_j, P_i sends $\langle\mathsf{Cert}_i\rangle_i$ together with $\langle\mathsf{Nomessage}\rangle_i$ to party P_j.
   - If P_i has received a valid certificate from party P_j, P_i sends $\langle\mathsf{Cert}_i\rangle_i$ together with $\langle\mathsf{Received}\rangle$ to party P_j.

   At the end of the phase, for each party P_i:
   - If P_i has received messages from strictly fewer than $n-t-s$ parties, P_i sets $Z_i = \mathsf{True}$ and sends $\langle Z_i = \mathsf{True}\rangle_i$ to all other parties.
   - Otherwise, if P_i has already set $\mathsf{lock}_{\mathsf{ALL}} = v$ and received $\mathsf{Nomessage}$ from strictly fewer than $t+r+1$ parties, P_i commits on v.

   Each party P_i sets $k = k+1$ and enters the next iteration.

Commit by notify. At any time of the protocol, if party P_i received notification headers $\langle\mathsf{notify}, v\rangle$ from $t+1$ distinct parties, combine those notify messages into a $t+1$-out-of-n threshold signature, send it to all other parties and terminate. This mechanism allows any send-omission faulty party to commit after $t+1$ non-faulty parties have committed. In addition, after all honest parties and send-omission faulty parties have terminated, all receive-omission faulty parties either terminate at the end of the next iteration or output $Z_i = \mathsf{True}$ in the next iteration.

Fig. 1: Core Protocol

In the core protocol, leader eligibility for iteration k is determined via a VRF with difficulty parameter D. Any party P_j whose private VRF evaluation $(y_j, \pi_j) \leftarrow \mathsf{Eval}(sk_j, k)$ satisfies $y_j < D$ is eligible to propose. Receivers apply a lowest-value rule: among all valid proposals they received in iteration k, they proceed only with the one having the smallest y. Our core protocol consists of 7 rounds per iteration. For a more intuitive presentation of the role of each round in the Byzantine agreement protocol, we present the lemmas alongside the explanations, and provide formal proofs in the Appendix B.

In the status phase, each party sends its current status (including its stored certificate) to all parties. In the propose phase, eligibility is determined by VRF as above; multiple proposals may exist, and each receiver proceeds only with the valid proposal with the smallest y it observed (lowest-value rule). A valid proposal packages Status messages from t+r+1 distinct parties and proposes the value whose certificate has the highest rank among them. Receivers then forward a $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$, tying every vote to the same accepted proposal. Parties enter the vote_2 phase and vote_3 phase using the same thresholds as specified, counting both the total messages received and the number of matching votes, exactly as in the algorithm.

We ensure that, after the round $\mathsf{lock}_{\mathsf{NS}}$, any two parties in the set of send-omission faulty parties and non-faulty parties cannot vote for different values.

Lemma 1 (Unanimous $\mathsf{lock}_{\mathsf{NS}}$). *For any two parties P and P' who are non-faulty or send-omission faulty (i.e., from the set $\mathcal{N} \cup \mathcal{S}$) and who respectively update $\mathsf{lock}_{\mathsf{NS}}$ to values $v \neq \perp$ and $v' \neq \perp$: $v = v'$.*

However, there could still be equivocation between non-faulty parties/send-omission parties and receive-omission parties, or equivocation between two receive-omission faulty parties. Thus, we need one more round, where each party that has received sufficiently many vote_2 messages among any form of messages it has received and has received messages with the same value while not detecting any form of equivocations will set $\mathsf{lock}_{\mathsf{ALL}} = v$. The idea is that, as they must receive messages from at least $t + r + 1$ distinct parties, at least one of them must be a send-omission faulty party or a non-faulty party, if all send-omission faulty parties and all non-faulty parties can detect the equivocation, all parties that are not Byzantine cannot vote for different values.

Lemma 2 (Unanimous $\mathsf{lock}_{\mathsf{ALL}}$ for $\mathcal{N} \cup \mathcal{S}$ and $\mathcal{R}$). *For any party P who is non-faulty or send-omission faulty (i.e., from the set $\mathcal{N} \cup \mathcal{S}$) and any party P' who is receive-omission faulty (i.e., from the set $\mathcal{R}$) that respectively update $\mathsf{lock}_{\mathsf{ALL}}$ to values $v \neq \perp$ and $v' \neq \perp$: $v = v'$.*

Based on this observation, we show that any two of non-Byzantine parties cannot set $\mathsf{lock}_{\mathsf{ALL}}$ to be different values $v \neq v'$ for $v \neq \perp$ and $v' \neq \perp$.

Lemma 3 (Unanimous $\mathsf{lock}_{\mathsf{ALL}}$). *For any two parties P and P' who are non-Byzantine (i.e., from the set $\mathcal{N} \cup \mathcal{S} \cup \mathcal{R}$) and who respectively update $\mathsf{lock}_{\mathsf{ALL}}$ to values $v_i \neq \perp$ and $v_j \neq \perp$: $v_i = v_j$.*

Thus, in the next round called Certificate Generation, we can ensure that there cannot be certificates with different values generated in each iteration.

Lemma 4. *Assume P_h is the first non-Byzantine party to commit, and it commits on v' in iteration k'. If there exists a certificate for (v, k'), then $v = v'$.*

Now we need to make sure that a committed party must deliver its certificate to all other parties. This is a bit subtle, as a send-omission faulty party may fail to do so. So we add a special checking process, which is inspired by the weak multicast protocol in [15], the difference is that we only need to ensure no party can commit without delivering its certificate to any receive-omission faulty party or any non-faulty party, so we do not require receive-omission detection, which is of $O(n^2)$ communication complexity. As a result, our ghost-check mechanism costs only $O(n)$ communication complexity for each sender.

Lemma 5 (Ghost-Check). *For any non-Byzantine party P_i that commits on v in iteration k at the end of* Ghost-Check and Commit*, all non-faulty parties and all send-omission faulty parties must have certificate with the same value at the end of* Ghost-Check and Commit*.*

Now we argue that if there exists a party that has committed in iteration k, no certificates with a rank higher than k can be generated. By property of the ghost-check, at least one receive-omission faulty party or a non-faulty party must have received its certificate, it must deliver the certificate to all send-omission faulty parties and all non-faulty parties. Recall that any of $t + r + 1$ Status messages must contain at least one message from a send-omission faulty party or a non-faulty party, no valid proposal with a different value can be generated from now on.

Lemma 6. *If the following conditions are satisfied at the beginning of iteration k:*

- *Each non-faulty party and each send-omission faulty party has a certificate (v, k_i) for value v.*
- *All certificates with a different value have a lower rank. (which means any certificate with (v', k') where $v' \neq v$ must have $k' < k_i$.)*

Then the conditions will hold at the end of iteration k.

Finally, we achieve the security that no two non-Byzantine parties can commit on different values, as we state as Theorem 1 below.

Theorem 1 (Security). *If two non-Byzantine parties commit on v and v', respectively, then $v = v'$.*

In order to let send-omission faulty parties terminate, we design notify mechanism which is similar to [1], where $t + 1$ notifications will let any parties to send to all other parties and terminate. This ensures that once $t + 1$ non-faulty parties are ready to terminate, this knowledge is disseminated to all non-faulty parties and send-omission faulty parties, guaranteeing liveness. We note that after a non-faulty eligible proposer with minimal y, all non-faulty parties would commit and send a notification to all other parties, and send-omission faulty parties must receive them and terminate in the next round.

Theorem 2. *If the minimal-VRF-evaluation valid proposal in iteration k is sent by a non-faulty party, then all non-faulty parties and all send-omission faulty parties terminate one round after iteration k (or earlier).*

After all non-faulty parties and all send-omission faulty parties have terminated, any receive-omission faulty party either receives $t+1$ notify messages from one of them (and they must terminate) or fail to receive $n - t - s$ messages in the next iteration, so each receive-omission faulty party must output that it is a zombie ($Z_i =$ True) and terminate. Our protocol terminates in expectation $7.05n/t + 4$.

Theorem 3. *The expected round complexity of our protocol is at most* $7.05n/t + 4$.

All formal proofs are provided in the Appendix B.

4 Byzantine Agreement

We are ready to build a (t, s, r)-secure Byzantine Agreement protocol. The formal definition is the same as prior works [15,14] and provided in Appendix A for completeness. The full protocol description is provided in Appendix D.

Pre-round. In the pre-round, each party multicasts its input; if party P_i receives strictly fewer than $n-t-s$ messages, P_i sets $Z_i = \mathsf{True}$. Messages received from $t+r+1$ parties with the same value form the *initial special certificate* with rank 0, party P_i who forms an initial certificate C_i sets $\mathsf{Cert}_i = C_i$. For party P_i who does not form a certificate, set $\mathsf{Cert}_i = \perp$. Then we invoke the core protocol. Parties that have not committed after λ iterations output $Z_i = \mathsf{True}$. The consistency properties are guaranteed by Theorem 1. The termination property is ensured by Lemma 3 and Theorem 2 for all non-faulty parties and all send-omission faulty parties, as they must receive $t+1$ notification headers and terminate. (we will specify for receive-omission faulty parties later). If all non-Byzantine parties start with the same value, at the end of the pre-round, all send-omission faulty parties and all non-faulty parties will hold certificates of the same value v at the beginning of the first iteration, as they must have received messages from all non-faulty parties and all receive-omission faulty parties, thus the conditions of Lemma 6 are satisfied. As a result, all non-faulty parties, all send-omission faulty parties must commit on value v.

Notify and Terminate. If we let all non-faulty parties and send-omission faulty parties keep participating in the protocol, even after they have committed and sent notify messages, it is hard for receive-omissions to detect their faults, output $Z = \mathsf{True}$ and terminate. This is because non-faulty parties and send-omission faulty parties may keep receiving $\perp$ messages. To solve this, we let all non-faulty parties and send-omission faulty parties terminate after sending out $t+1$ headers of notify messages (which is combined in a t+1-out-of-n threshold signature). Since non-faulty parties and send-omission faulty parties never drop messages sent from non-faulty parties, they must terminate and output the correct value. In addition, messages sent from send-omission faulty parties can always be dropped by the adversary, the No Living Undead property is preserved.

Dealing with Zombies. A party can only become a zombie if it receives messages from strictly fewer than $n-t-s$ distinct parties, which means at least one message sent from a non-faulty party or a receive-omission faulty party has been dropped, indicating that it must be a receive-omission faulty party. After the last non-faulty party has committed, all non-faulty parties and all send-omission faulty parties will terminate in the next round. As a result, a receive-omission faulty party P_i has received one notify message and terminates, or it becomes zombie in the next iteration.

Theorem 4. *If there are at most t Byzantine parties, s send-omission faulty parties, r receive-omission faulty parties, where $n > 2t+s+r$ and t is at least ϵn for some constant $\epsilon > 0$, then the Byzantine Agreement protocol $\Pi_{t,s,r}$ terminates with expected $O(1)$ round complexity and $O(n^2)$ communication complexity.*

References

1. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected o (1) rounds, expected communication, and optimal resilience. In: International Conference on Financial Cryptography and Data Security. pp. 320–334. Springer (2019)
2. Abraham, I., Dolev, D., Kagan, A., Stern, G.: Brief announcement: Authenticated consensus in synchronous systems with mixed faults. In: Scheideler, C. (ed.) 36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA. LIPIcs, vol. 246, pp. 38:1–38:3. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.DISC.2022.38, https://doi.org/10.4230/LIPIcs.DISC.2022.38
3. 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. pp. 331–341 (2021)
4. Bacho, R., Loss, J.: On the adaptive security of the threshold BLS signature scheme. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 193–207. ACM Press (Nov 2022). https://doi.org/10.1145/3548606.3560656
5. Bacho, R., Loss, J., Tessaro, S., Wagner, B., Zhu, C.: Twinkle: Threshold signatures from DDH with full adaptive security. In: Joye, M., Leander, G. (eds.) EUROCRYPT 2024, Part I. LNCS, vol. 14651, pp. 429–459. Springer, Cham (May 2024). https://doi.org/10.1007/978-3-031-58716-0_15
6. Backes, M., Cachin, C.: Reliable broadcast in a computational hybrid model with byzantine faults, crashes, and recoveries. In: DSN. vol. 3, pp. 37–46 (2003)
7. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In: Desmedt, Y. (ed.) PKC 2003. LNCS, vol. 2567, pp. 31–46. Springer, Berlin, Heidelberg (Jan 2003). https://doi.org/10.1007/3-540-36288-6_3
8. Brazitikos, K., Zikas, V.: General adversary structures in byzantine agreement and multi-party computation with active and omission corruption. In: Boyle, E., Mahmoody, M. (eds.) TCC 2024, Part IV. LNCS, vol. 15367, pp. 200–233. Springer, Cham (Dec 2024). https://doi.org/10.1007/978-3-031-78023-3_7
9. David, B., Gazi, P., Kiayias, A., Russell, A.: Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: Nielsen, J.B., Rijmen, V. (eds.) EUROCRYPT 2018, Part II. LNCS, vol. 10821, pp. 66–98. Springer, Cham (Apr / May 2018). https://doi.org/10.1007/978-3-319-78375-8_3
10. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: Ateniese, G., Venturi, D. (eds.) ACNS 2022. LNCS, vol. 13269, pp. 684–704. Springer, Cham (Jun 2022). https://doi.org/10.1007/978-3-031-09234-3_34
11. Feng, H., Lu, Z., Tang, Q., Ye, Y.: Optimal byzantine agreement in the presence of message drops. Springer-Verlag (2025)
12. Garay, J.A., Perry, K.J.: A continuum of failure models for distributed computing. In: Segall, A., Zaks, S. (eds.) Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings. Lecture Notes in Computer Science, vol. 647, pp. 153–165. Springer (1992). https://doi.org/10.1007/3-540-56188-9_11, https://doi.org/10.1007/3-540-56188-9_11
13. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176, https://doi.org/10.1145/357172.357176

14. Loss, J., Shi, K., Stern, G.: Consensus in the presence of overlapping faults and total omission. In: Boyle, E., Mahmoody, M. (eds.) TCC 2024, Part I. LNCS, vol. 15364, pp. 353–382. Springer, Cham (Dec 2024). https://doi.org/10.1007/978-3-031-78011-0_12
15. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: Rothblum, G.N., Wee, H. (eds.) TCC 2023, Part IV. LNCS, vol. 14372, pp. 395–421. Springer, Cham (Nov / Dec 2023). https://doi.org/10.1007/978-3-031-48624-1_15
16. Malkin, T., Micciancio, D., Miner, S.: Efficient generic forward-secure signatures with an unbounded number of time periods. In: International Conference on the Theory and Applications of Cryptographic Techniques. pp. 400–417. Springer (2002)
17. Micali, S., Rabin, M., Vadhan, S.: Verifiable random functions. In: 40th annual symposium on foundations of computer science (cat. No. 99CB37039). pp. 120–130. IEEE (1999)
18. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Software Eng. **12**(3), 477–482 (1986). https://doi.org/10.1109/TSE.1986.6312888, https://doi.org/10.1109/TSE.1986.6312888
19. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: 2002 Pacific Rim International Symposium on Dependable Computing, 2002. Proceedings. pp. 221–228. IEEE (2002)
20. Shoup, V.: Practical threshold signatures. In: Advances in Cryptology—EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings 19. pp. 207–220. Springer (2000)
21. Siu, H.S., Chin, Y.H., Yang, W.P.: Byzantine agreement in the presence of mixed faults on processors and links. IEEE Transactions on Parallel and Distributed Systems **9**(4), 335–345 (1998). https://doi.org/10.1109/71.667895
22. Zikas, V., Hauser, S., Maurer, U.M.: Realistic failures in secure multi-party computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 274–293. Springer, Berlin, Heidelberg (Mar 2009). https://doi.org/10.1007/978-3-642-00457-5_17

A Definitions

Definition 1 ((t,s,r)-secure Byzantine Agreement). *Let Π be a protocol executed by n parties $P_1, ..., P_n$, where each party P_i starts with an input m_i, and every party P_i outputs (x_i, Z_i) at the end of the protocol.*

- ***Validity.*** *If each non-Byzantine party starts with the same input v, every party i with $Z_i \neq \mathsf{True}$ at the end of the protocol must output v at the end of the protocol.*
- ***Consistency.*** *All parties with $Z_i \neq \mathsf{True}$ at the end of the protocol must output the same value.*
- ***Termination.*** *All parties eventually terminate and output.*
- ***No Living Undead.*** *For any party P_i that outputs $Z_i = \mathsf{True}$, P_i must be a receive-omission faulty party.*

If Π satisfies properties Validity*,* Consistency*,* Termination*, and* No Living Undead *whenever up to t parties are Byzantine, up to s parties are send-omission faulty parties and up to r parties are receive-omission faulty parties, Π is called a (t,s,r)-secure Byzantine Agreement protocol.*

B Security Proofs

This section contains all of the formal proofs in section 3.3.

Message format (VRF). Every vote explicitly carries the VRF triple; we write votes as

$$\big\langle \langle v\rangle_j,\ \mathsf{vote}_\ell,\ (k, j, y_j)\big\rangle$$

and whenever we count M such messages in the proofs below, they are M messages for the *same* triple (k, j, y_j) and the same value $v \neq \perp$.

Lemma 1 (Unanimous $\mathsf{lock}_{\mathsf{NS}}$). *For any two parties P and P' who are non-faulty or send-omission faulty (i.e., from the set $\mathcal{N} \cup \mathcal{S}$) and who respectively update $\mathsf{lock}_{\mathsf{NS}}$ to values $v \neq \perp$ and $v' \neq \perp$: $v = v'$.*

Proof. Without loss of generality we assume the VRF value y (corresponding to proposal value v) is no higher than y' (corresponding to proposal value v'). Since P and P' are not receive-omission faulty, they must have received all messages from non-faulty parties and receive-omission faults. Now we consider two possible cases of how they may set $\mathsf{lock}_{\mathsf{NS}} \neq \perp$:

- If P is a non-faulty party or send-omission faulty party who has received strictly fewer than $n-s$ messages, with at least $n-t-s-r$ many $\big\langle \langle v\rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j)\big\rangle$ with $v \neq \perp$ (for the same (k, j, y_j)). Since at least s messages have not been delivered to the non-receive-omission faulty party, and these messages must be dropped because of send-omission corruptions and Byzantine corruptions, at most $s+t-s=t$ of those vote_1 messages can come from send-omission faulty parties or Byzantine parties. As a result, at least $n - t - s - r - t \geq 1$ of the $\langle \langle v \rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j) \rangle$ messages must come from a non-faulty party or a receive-omission faulty party. As this party sends messages consistently, its message must arrive at all non-faulty parties and all send-omission faults, which will prevent them from setting $\mathsf{lock}_{\mathsf{NS}}$ and sending $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$, $\langle \langle v' \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ for any different values $v' \neq v$, where $v \neq \perp$ and $v' \neq \perp$.

- If one non-faulty party or send-omission faulty party has received $n - s + x$ (where $x \geq 0$) messages in which at least $n - t - s - r + x$ of them are $\langle \langle v \rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j) \rangle$ with $v \neq \perp$ (for the same (k, j, y_j)). Again, since at least $s - x$ messages have not been delivered to the party, which must have been dropped because of send-omission corruptions and Byzantine corruptions, at most $t + s - (s - x)$ of those vote_1 can come from send-omission faulty parties and Byzantine parties. As a result, at least $n - t - s - r + x - (t + s - (s - x)) \geq 1$ $\langle \langle v \rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j) \rangle$ messages must come from a non-faulty or a receive-omission faulty party. Its message must arrive at all non-faulty parties and all send-omission faults, which will prevent them from setting $\mathsf{lock}_{\mathsf{NS}}$ and sending $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ and $\langle \langle v' \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ for any different value $v' \neq v$ with $y' > y$.

Lemma 2 (Unanimous $\mathsf{lock}_{\mathsf{ALL}}$ for $\mathcal{N} \cup \mathcal{S}$ and $\mathcal{R}$). *For any party P who is non-faulty or send-omission faulty (i.e., from the set $\mathcal{N} \cup \mathcal{S}$) and any party P' who is receive-omission faulty (i.e., from the set $\mathcal{R}$) that respectively update $\mathsf{lock}_{\mathsf{ALL}}$ to values $v \neq \perp$ and $v' \neq \perp$: $v = v'$.*

Proof. Without loss of generality we assume the VRF value y (corresponding to proposal value v) is no higher than y' (corresponding to proposal value v'). Since P is not a receive-omission faulty party, it must have received all messages from non-faulty parties and receive-omission faults.

- If P has received strictly fewer than $n - s$ messages with at least $n - t - s - r$ $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ for value $v \neq \perp$ (for the same (k, j, y_j)), in this case, since at least s messages have not been delivered to the party, which must be dropped because of send-omission corruptions or Byzantine corruptions, at most $s + t - s = t$ of those vote_2 for value $v \neq \perp$ come from send-omission faulty parties or Byzantine parties. As a result, at least $n - t - s - r - t \geq 1$ $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ must come from a non-faulty party or a receive-omission faulty party. Its message must arrive at all non-faulty parties and all send-omission faults. As a result, no non-faulty parties or send-omission faulty parties can set $\mathsf{lock}_{\mathsf{ALL}}$ to be any different value $v' \neq v$ with $y' > y$.
- If P has received at least $n - s + x$ (where $x \geq 0$) messages in which at least $n - t - s - r + x$ of them are $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ for value $v \neq \perp$ (for the same (k, j, y_j)), at most $s + t - (s - x)$ votes can come from send-omission faults or Byzantine parties. As a result, at least $n - t - s - r + x - (s + t - (s - x)) \geq 1$ $\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \rangle$ comes from a receive-omission faulty or a non-faulty party. Its message must arrive at all non-faulty parties and all send-omission

faults. This will prevent them from setting $\mathsf{lock}_{\mathsf{ALL}}$ to be any different value $v' \neq v$ with $y' > y$.

Now assume there is a receive-omission faulty party P' that is alive and has $\mathsf{lock}_{\mathsf{ALL}}$ on v', it must have received $\langle\langle v'\rangle_j,\ \mathsf{vote}_2,\ (k,j,y_j)\rangle$ from at least a non-Byzantine party. By Lemma 1, if it comes from a send-omission faulty party or a non-faulty party, there cannot exist send-omission faulty parties and non-faulty parties that $\mathsf{lock}_{\mathsf{NS}}$ and send $\langle\langle v\rangle_j,\ \mathsf{vote}_2,\ (k,j,y_j)\rangle$ for a different value $v \neq \perp$, as it is implied in the first part of the proof, no non-faulty party or send-omission faulty party can vote for a different value. If it comes from a receive-omission faulty party, then all send-omission faulty parties and non-faulty parties will receive it through its $\langle\langle v'\rangle_j,\ \mathsf{vote}_2,\ (k,j,y_j)\rangle$ messages and thus, no send-omission faulty parties or non-faulty parties can set $\mathsf{lock}_{\mathsf{ALL}} = v$ for any value $v \neq v'$ and $v \neq \perp$.

Lemma 3 (Unanimous $\mathsf{lock}_{\mathsf{ALL}}$). *For any two parties P and P' who are non-Byzantine (i.e., from the set $\mathcal{N} \cup \mathcal{S} \cup \mathcal{R}$) and who respectively update $\mathsf{lock}_{\mathsf{ALL}}$ to values $v_i \neq \perp$ and $v_j \neq \perp$: $v_i = v_j$.*

Proof. Without loss of generality we assume the VRF value y (corresponding to proposal value v) is no higher than y' (corresponding to proposal value v'). By Lemma 2, we only need to prove the case where both of P and P' are receive-omission faulty parties. Assume P sets $\mathsf{lock}_{\mathsf{ALL}} = v$ and P' sets $\mathsf{lock}_{\mathsf{ALL}} = v'$, where $v \neq \perp$ and $v' \neq \perp$. According to the code, P and P' must have received v and v' respectively at the end of the proposal phase. In addition, they must not have detected the equivocation. Since both of P and P' have not become a zombie, P and P' must have received at least $n - t - s$ messages. At least $n - t - s - (t + r) \geq 1$ message must come from a non-faulty party or a send-omission faulty party P^*. But P^* can only vote for v' since $y' < y$, or it must have detected equivocation if both of P and P' received valid proposal of different values $\langle\langle v\rangle_j,\ \mathsf{vote}_1,\ (k,j,y_j)\rangle$ and $\langle\langle v'\rangle_{j'},\ \mathsf{vote}_1,\ (k,j',y_{j'})\rangle$ through vote_1 messages of P and P', where P^* must have attached the equivocation proof in its vote_2 messages. In this scenario, party P cannot set $\mathsf{lock}_{\mathsf{ALL}} = v$ and P' cannot set $\mathsf{lock}_{\mathsf{ALL}} = v'$. So we must have $v = v'$.

Lemma 4. *Assume P_h is the first non-Byzantine party to commit, and it commits on v' in iteration k'. If there exists a certificate for (v, k'), then $v = v'$.*

Proof. A certificate for v (or v') consists of at least $t{+}1$ votes $\langle\langle v\rangle_j,\ \mathsf{vote}_3,\ (k,j,y_j)\rangle$ (respectively for v') for the same value $v \neq \perp$ (or $v' \neq \perp$). At least one such vote_3 comes from a non-Byzantine party and at least one for the other certificate comes from a non-Byzantine party. So we have $v = v'$ by Lemma 3.

Lemma 5 (Ghost-Check). *For any non-Byzantine party P_i that commits on v in iteration k at the end of* Ghost-Check and Commit*, all non-faulty parties and all send-omission faulty parties must have certificate with the same value at the end of* Ghost-Check and Commit.

Proof. If P_i commits to v in iteration k at the end of *Ghost–Check and Commit*, P_i must receive "Nomessage" from strictly fewer than $t + r + 1$ parties, at most one non-faulty party or a receive-omission faulty party must have received the certificate sent by P_i at the end of *Certificate Generation*, and it must have sent the certificate to all non-faulty parties and all send-omission faulty parties in the *Ghost–Check and Commit*.

Lemma 6. *If the following conditions are satisfied at the beginning of iteration k:*

- *Each non-faulty party and each send-omission faulty party has a certificate (v, k_i) for value v.*
- *All certificates with a different value have a lower rank. (which means any certificate with (v', k') where $v' \neq v$ must have $k' < k_i$.)*

Then the conditions will hold at the end of iteration k.

Proof. Assume there exists a non-Byzantine party that gets a certificate for v' with rank higher than the certificate it had at the beginning of the iteration k. At least one non-Byzantine party, say h, must have set $\mathsf{lock}_{\mathsf{NS}} = v'$. In order for h to set $\mathsf{lock}_{\mathsf{NS}} = v'$, the sender must send a proposal with at least $n - t - s$ Status messages, which must contain at least $n - t - s - t - r \geq 1$ message from a send-omission faulty or a non-faulty party. By assumption, this Status message must contain certificate for value v and there must be a certificate with a higher rank for value v' at the same time, which contradicts the first condition. By induction, if the conditions hold at the beginning of the iteration k, they will hold true permanently.

Theorem 1 (Security). *If two non-Byzantine parties commit on v and v', respectively, then $v = v'$.*

Proof. Suppose party h is the first non-Byzantine party to commit, and it commits to v' in iteration k'. By Lemma 3, once a non-Byzantine party sets $\mathsf{lock}_{\mathsf{ALL}} = v$ in the iteration k, no other non-Byzantine party can $\mathsf{lock}_{\mathsf{ALL}}$ to be a different value v' in the same iteration k'. Thus, v' cannot be committed in the same iteration k'. By Lemma 5, any non-Byzantine party only commits if it received strictly fewer than $n - t - s$ many Nomessage. All non-faulty parties and all send-omission faulty parties must have certificate with the same value at the end of *Ghost–Check and Commit.* As a result, all non-faulty parties and all send-omission faulty parties will update their certificate with rank k' and value v', which satisfies two conditions of Lemma 6. Thus, no proposals with a different value can be formed from now on. Thus, by Lemma 3 $\mathsf{lock}_{\mathsf{ALL}}$ ensures that no two non-Byzantine parties can commit to different values. In addition, all non-faulty parties update their certificates and the conditions of Lemma 6 are satisfied, so we conclude that $v = v'$.

Termination. If the unique leader P_j in iteration k with lowest VRF evaluation y_j is non-faulty, every send-omission faulty party and every non-faulty party terminates one round after iteration k (or earlier).

Theorem 2. *If the minimal-VRF-evaluation valid proposal in iteration k is sent by a non-faulty party, then all non-faulty parties and all send-omission faulty parties terminate one round after iteration k (or earlier).*

Proof. If the unique leader P_j in iteration k with lowest VRF evaluation y_j is non-faulty, it will send a valid proposal consisting of at least $n - t - s$ status messages to all other parties and propose a value reported by the highest certificate it collected in the status phase. This certificate will be no lower than any certificate held by receive-omission faults and non-faulty parties. Furthermore, due to the unforgeability property of our digital signature scheme, the adversary cannot accuse L_k to be equivocating. Therefore, all non-faulty parties and all send-omission faulty parties will set $\mathsf{lock}_{\mathsf{NS}} = v$ and send $\big\langle \langle v\rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j)\big\rangle$ with the value $v \neq \perp$ sent from the leader, regardless of the fact that vote_2 messages from send-omission faulty parties can be dropped by the adversary. For each non-Byzantine party who is alive throughout the iteration, either it receives strictly fewer than $n - s$ messages in which at least $n - t - s - r$ of them are $\big\langle \langle v\rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j)\big\rangle$, or $n - s + x$ messages in which at least $n - t - s - r + x$ of them are $\big\langle \langle v\rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j)\big\rangle$. As a result, all send-omission faulty parties and non-faulty parties will $\mathsf{lock}_{\mathsf{NS}} = v$ and send $\big\langle \langle v\rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j)\big\rangle$ to all other parties. Similarly, all send-omission faulty parties and all non-faulty parties will set $\mathsf{lock}_{\mathsf{ALL}} = v$ and send $\big\langle \langle v\rangle_j,\ \mathsf{vote}_3,\ (k, j, y_j)\big\rangle$ to all other parties and a certificate will be formed. Each non-faulty party will receive $\mathsf{Nomessage}$ from at most $t + r + 1$ parties, so they will commit to v. All those honest parties will send notify for v to all other parties after they have committed. As a result, all send-omission faulty parties must receive at least $t + 1$ notify messages and terminate according to the protocol.

Theorem 3. *The expected round complexity of our protocol is at most* $7.05n/t + 4$.

Proof. In each iteration k, we assume there is no collisions for the lowest VRF evaluation, as this happens with negligible probability. We set the difficulty parameter D such that the probability for each party to be eligible is $p = c/n$ for some constant $c > 0$, where we pick $c = 5$ arbitrarily. That is, we set $D = p \cdot 2^{\ell_{\mathsf{VRF}}}$ where ℓ_{VRF} is the number of bits in each VRF output. Then we can bound the probability that the lowest VRF evaluation simultaneously belongs to an honest party and is below the threshold D, as the product of the two independent events that 1) it is not the case that all VRF evaluations for a proposal are above the threshold and 2) that the proposal with lowest VRF evaluation comes from a non-faulty party or a receive-omission that is not alive. I.e., as:

$$(1 - (1 - p)^n) \cdot \frac{n - t - s - r + r'}{n - r + r'} \geq (1 - e^{-c}) \cdot \frac{t + 1 + r'}{2t + 1 + s + r'}$$

Setting c to be 5, the expected round complexity is at most

$$\frac{2t + 1 + s + r'}{(t + 1 + r') \cdot (1 - e^{-c})} \times 7 + 4 \leq 7.05n/t + 4$$

Where r' denotes number receive-omission faulty parties that are alive before the $\mathsf{proposal}$ phase in iteration k. There exists 1 pre-round before the iteration, each iteration consists of 7 rounds, send-omission faulty parties and non-faulty parties terminate one round after one good iteration, where receive-omission faulty parties terminate two rounds after all send-omission faulty parties and non-faulty parties have terminated (to be more precise, in the vote_1 phase of the next iteration).

C Random Leader Election

C.1 VRF-based eligibility

In iteration k, party P_j computes $(y_j, \pi_j) \leftarrow \mathsf{Eval}(sk_j, k)$ and is eligible iff $y_j < D$. Eligible parties multicast their proposals bundling $t+r+1$ Status messages and proposing the value of the highest-rank certificate among them. Receivers verify $\mathsf{Verify}(pk_j, k, y_j, \pi_j) = \mathsf{True}$ and proceed only with the valid proposal having the smallest y they observed. All votes in the iteration are fixed to that accepted triple (k, j, y_j).

C.2 Adaptive Security

Before $\mathsf{proposal}$. The Status phase precedes proposals, so an adversary cannot know the minimal-y eligible proposer before $\mathsf{proposal}$ messages are sent.

After $\mathsf{proposal}$. We assume forward-secure signatures and secure erasure of proposal-signing keys; after sending its proposal, a non-faulty party cannot be made to equivocate by adaptive corruption. Any attempt to forge conflicting proposals violates unforgeability. Consequently, equivocation evidence against a non-faulty proposer cannot be produced after the fact.

D The (t, s, r)-secure Byzantine Agreement Protocol $\Pi_{t,s,r}$

Figure 2 includes the pseudo code for the BA protocol outlined in Section 4.

Byzantine Agreement Protocol $\Pi_{t,s,r}$

Input and Initialization: Each party P_i, $i \in [n]$, starts with input m_i, initializes $Z_i = \mathsf{False}$ and $\mathsf{Cert}_i = \perp$.

Pre-round 1. Each P_i multicasts its input m_i to all parties.

Pre-round 2. If P_i received fewer than $n - t - s$ messages in Pre-round 1, set $Z_i = \mathsf{True}$. Otherwise, if there exists a value v received from $t+r+1$ distinct parties, combine it into a certificate C_i and set $\mathsf{Cert}_i = C_i$.

For iterations $k = 1, 2, \ldots$ all parties execute:

1. **Status.** For party P_i:
   If $Z_i = \mathsf{False}$, P_i multicasts a status message $\langle k, \mathsf{Status}, \mathsf{Cert}_i, v_i \rangle_i$ to all other

parties to report its current status. If $Z_i = \mathsf{True}$, P_i multicasts $\langle Z_i = \mathsf{True} \rangle_i$ to all other parties instead. All parties update their current certificate to be the one with highest rank. If party P_i has committed on value v, send $\langle \mathsf{notify}, v \rangle_i$ to all other parties.

2. **Propose.** If $Z_i = \mathsf{False}$, party P_i computes $(y_i, \pi_i) \leftarrow \mathsf{Eval}(sk_i, (k, i))$. If $y_i < D$, P_i signs a proposal.

$$\Big\langle (y_i, \pi_i),\ k,\ \mathsf{proposal},\ \{\mathsf{Status}_\ell \mid \ell \in J,\ J \subseteq [n],\ |J| = t{+}r{+}1\}, \qquad \big\langle v_L = v_{\arg\max_{\ell \in J} R_\ell} \big\rangle_i \Big\rangle_i$$

to all parties, where R_ℓ is the rank of the certificate contained in Status_ℓ and v_L is the value of the highest-rank certificate among those $t{+}r{+}1$ Status messages. If no certificate has rank > 0, P_i may choose v_L freely. Each party P_i then deletes its signing key for iteration k, multicasts the proposal to all other parties.

3. **Forward.** (vote_1) Let Q be the set of indices q from whom P_i received a valid proposal in this phase. If $Q \neq \emptyset$, set

$$j \leftarrow \arg\min_{q \in Q} \{ y_q \}$$

Party P_i multicasts $\big\langle \langle v_{j \to i} \rangle_j,\ \mathsf{vote}_1,\ (k, j, y_j) \big\rangle_i$ iff all of the following hold for the proposal from P_j:
   - $\mathsf{Verify}(pk_j, k, y_j, \pi_j) = \mathsf{True}$;
   - $v_{j \to i} \neq \bot$;
   - the proposal includes at least $t{+}r{+}1$ valid Status messages, and the value of the highest-ranked certificate among them equals $v_{j \to i}$;
   - $y_j < D$.

   If no such valid minimum-y proposal exists, P_i multicasts $\langle \bot \rangle_i$.

4. **First Lock Step.** (vote_2) For party P_i:
   If P_i has received strictly fewer than $n - t - s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If there exist two valid proposals with different values v and v' for the same (locally) lowest VRF evaluation, P_i multicasts conflicting headers to all other parties.
   - If no equivocation is detected and if one of following conditions a) and b) is satisfied, P_i sets $\mathsf{lock}_{\mathsf{NS}} = v$ and sends $\big\langle \langle v \rangle_j,\ \mathsf{vote}_2,\ (k, j, y_j) \big\rangle_i$ to all parties:
     a) If party P_i receives strictly fewer than $n - s$ messages in which at least $n - t - s - r$ of them are $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$ for $v \neq \bot$.
     b) If party P_i receives $n - s + x$ (where $x \geq 0$) messages in which at least $n - t - s - r + x$ of them are $(v,\ \mathsf{vote}_1,\ (k, j, y_j))$ for $v \neq \bot$.

   - Otherwise, P_i sets $\mathsf{lock}_{\mathsf{NS}} = \bot$ and sends triples $\langle k, j, y_j \rangle$ for the lowest VRF evaluation y_j (if there is no such valid triples, sends $\langle \bot \rangle_i$) to all other parties.

5. **Second Lock Step.** ($\mathsf{lock}_{\mathsf{ALL}}$) For party P_i: If P_i has received strictly fewer than $n - t - s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If an equivocation (with (locally) smallest VRF evaluation) is detected by P_i, P_i multicast conflicting headers. Otherwise, if P_i has not received a valid

proposal from P_j in the proposal phase for the (locally) lowest triple of (k, j, y_j) received in the forward phase, P_i sets $\mathsf{lock_{ALL}} = \perp$ and multicasts $\perp$ to all other parties.

- If no equivocation is detected by P_i, and P_i has received the valid proposal with the same triple (k, j, y_j) with the (locally) smallest y_j, and if one of following conditions a) and b) is satisfied for party P_i, P_i sets $\mathsf{lock_{ALL}} = v$ and sends $\langle \langle v \rangle_j,\ \mathsf{vote}_3,\ (k, j, y_j) \rangle_i$ to all parties:
  a) If P_i has received strictly fewer than $n - s$ messages in which $n - t - s - r$ of them are $(v,\ \mathsf{vote}_2,\ (k, j, y_j))$ for $v \neq \perp$.
  b) If P_i has received $n-s+x$ (where $x \geq 0$) messages in which $n-t-s-r+x$ of them are $(v,\ \mathsf{vote}_2,\ (k, j, y_j))$ for $v \neq \perp$.

- Otherwise, P_i sets $\mathsf{lock_{NS}} = \perp$ and sends triples $\langle k, j, y_j \rangle$ for the lowest VRF evaluation y_j (if there is no such valid triples, sends $\langle \perp \rangle_i$) to all other parties.

6. **Certificate Generation.** For party P_i: If P_i has received strictly fewer than $n - t - s$ messages in the previous round, set $Z_i = \mathsf{True}$ and send $Z_i = \mathsf{True}$ to all other parties. Otherwise,
   - If one of the following conditions a) or b) is satisfied for party P_i, then P_i aggregates messages of the form $\langle \langle v \rangle_j,\ \mathsf{vote}_3,\ (k, j, y_j) \rangle$ (same v and same triple (k, j, y_j)) from any $t{+}1$ distinct parties into a certificate C_i using a $t{+}1$-out-of-n threshold signature scheme. P_i sets $\mathsf{Cert}_i = C_i$ and multicasts Cert_i to all other parties:
     a) P_i has received strictly fewer than $n - s$ messages, of which at least $n - t - s - r$ are $(v,\ \mathsf{vote}_3,\ (k, j, y_j))$ with $v \neq \perp$.
     b) P_i has received $n - s + x$ messages (for some $x \geq 0$), of which at least $n - t - s - r + x$ are $(v,\ \mathsf{vote}_3,\ (k, j, y_j))$ with $v \neq \perp$.
7. **Ghost-Check and Commit.** For each party P_i, P_i sets its local certificate Cert_i to be the certificate C_i^* with the highest rank.
   - If P_i did not receive a valid certificate from party P_j, P_i sends $\langle \mathsf{Cert}_i \rangle_i$ together with $\langle \mathsf{Nomessage} \rangle_i$ to party P_j.
   - If P_i has received a valid certificate from party P_j, P_i sends $\langle \mathsf{Cert}_i \rangle_i$ together with $\langle \mathsf{Received} \rangle$ to party P_j.

   At the end of the phase, for each party P_i:
   - If P_i has received messages from strictly fewer than $n - t - s$ parties, P_i sets $Z_i = \mathsf{True}$ and sends $\langle Z_i = \mathsf{True} \rangle_i$ to all other parties.
   - Otherwise, if P_i has already set $\mathsf{lock_{ALL}} = v$ and received $\mathsf{Nomessage}$ from strictly fewer than $t + r + 1$ parties, P_i commits on v.

   Each party P_i sets $k = k + 1$ and enters the next iteration.

Commit by notify. At any time, if P_i receives $\langle \mathsf{notify}, v \rangle$ from $t{+}1$ distinct parties, combine them into a $t{+}1$-out-of-n threshold signature for notify, multicast it once to all parties, and terminate.

Fig. 2: A (t, s, r)-secure Byzantine Agreement Protocol

Finally, we include the full proof of security of the BA protocol specified in Fig. 2.

Theorem 4. *If there are at most t Byzantine parties, s send-omission faulty parties, r receive-omission faulty parties, where $n > 2t + s + r$ and t is at least ϵn*

for some constant $\epsilon > 0$, then the Byzantine Agreement protocol $\Pi_{t,s,r}$ terminates with expected $O(1)$ round complexity and $O(n^2)$ communication complexity.

Proof. We show a detailed analysis of the efficiency:

- **Round Complexity.** The pre-round phase takes 1 round, each iteration of our core protocol incurs 7 rounds. According to Lemma 3, a good iteration exists in expected $O(1)$ rounds. By Theorem 2, all non-faulty parties and all send-omission faulty parties commit in any good iteration. As a result, they will send notify messages at the end of this iteration. Either all receive-omission faulty parties receive some combined form of $t+1$ notify headers and terminate, or they must become zombies in the next iteration. In conclusion, our Byzantine Agreement protocol terminates with expected $O(1)$ round complexity.
- **Communication Complexity.** In the pre-round phase, all parties send one message with their signatures. In each iteration of the core protocol:
  1. In the Status phase, each party sends a status message of size $O(1)$, where the certificate is generated by threshold signature schemes. The total communication complexity is $O(n^2)$.
  2. In the propose phase, there exists an expected constant number of non-faulty leaders, each of them sends a proposal message consisting of $t+r+1$ messages to all other parties, which incurs $O(n^2)$ expected communication complexity.
  3. Each Lock-Step has one round n-to-n communication with message size $O(1)$. The total communication complexity is $O(n^2)$.
  4. Certificate Generation has one round n-to-n communication with message size $O(1)$. The total communication complexity is $O(n^2)$.
  5. Ghost-check has two rounds of n-to-n communication with message size $O(1)$. The total communication complexity is $O(n^2)$.
  6. Each notify message is of size $O(1)$ and $t + 1$ headers of notifications are combined into one signature. The total communication complexity is $O(n^2)$.

  By Theorem 3, protocol terminates within $7.05n/t + 4$ number of rounds in expectation, the overall communication complexity is $O(n^2)$.