Zombies and Ghosts: Optimal Byzantine Agreement in the Presence of Omission Faults

Julian Loss[1] and Gilad Stern[2]

[1] CISPA Helmholtz Center for Information Security, Saarbrücken, Germany
[2] The Hebrew University of Jerusalem, Jerusalem, Israel

Abstract. Studying the feasibility of Byzantine Agreement (BA) in realistic fault models is an important question in the area of distributed computing and cryptography. In this work, we revisit the mixed fault model with Byzantine (malicious) faults and omission faults put forth by Hauser, Maurer, and Zikas (TCC 2009), who showed that BA (and MPC) is possible with t Byzantine faults, s send faults (whose outgoing messages may be dropped) and r receive faults (whose incoming messages may be lost) if $n > 3t + r + s$. We generalize their techniques and results by showing that BA is possible if $n > 2t + r + s$, given the availability of a cryptographic setup. Our protocol is the first to match the recent lower bound of Eldefrawy, Loss, and Terner (ACNS 2022) for this setting.

Keywords: Consensus · Synchrony · Mixed-Faults

1 Introduction

Byzantine agreement (BA) is a fundamental problem in distributed computing where n parties $1, \ldots, n$ each hold an input v_i and want to agree on a common output v by running some distributed protocol Π. However, their task is complicated by some $t < n$ out of the parties deviating from the protocol description, e.g., by crashing or sending incorrect messages. BA forms the backbone of many distributed protocols and has wide-ranging applications such as multi-party computation, verifiable secret sharing, and replicated state machines. For this reason, an extensive body of literature has studied the feasibility of BA under various conditions, e.g., different types of network behaviors and/or faults. If no setup is assumed, the celebrated work of Lamport, Shostak and Pease [16] demonstrates that BA is possible if and only if the number of *malicious faults* t satisfies $t < \frac{n}{3}$. On the other hand, a protocol can tolerate any number of *crash faults* (i.e., where a party crashes). As a middle ground between these two types of faults, many previous works have also considered so-called *omission faults.* An omission fault is a party that remains honest and online, but for which some of its incoming and outgoing messages may not be delivered. This makes omission faults a very realistic, but also particularly difficult type of fault to deal with.

Toward a more general understanding of fault tolerance with omission faults, Hauser, Maurer, and Zikas [28], studied BA (and multi-party computation) in a mixed model with malicious faults and omission faults. Their work considers two types of omission faults, *receive omission faults* (a party does not receive some incoming messages) and *send omission faults* (some messages of a party are not delivered). Their result shows that BA and MPC are possible if $n > 3t + r + s$, where t is the number of malicious faults, r is the number of receive omission faults, and s is the number of send omission faults.[3] Much more recently, Eldefrawy, Loss, and Terner [11] make a first attempt at translating this result to a setting where cryptographic setup (i.e., a PKI) is available to the parties. In this setting, they show that BA is possible only if $n > 2t + r + s$. On the converse, they show a protocol that matches this bound when considering a special type of send omission fault called a *spotty fault*. Spotty faults must drop either *all or none* of their messages in any given protocol round; a feature which their protocols crucially exploit. Their work explicitly leaves open the question of finding a protocol matching the lower bound even for general send omission faults. In this work, we answer this question in the affirmative. More concretely, we show the following results:

– We begin by revisiting the protocol framework of Hauser et al. used by parties to detect and silence themselves upon becoming receive omission faulty. To overcome additional obstacles that we are presented with as a result of our more general fault regime, we extend their framework to the much more challenging case of send omission faults.
– Using our new framework, we then give the the first protocol matching the $2t + r + s < n$ lower bound of Eldefrawy et al. in the mixed model with malicious faults and general send/receive omission faults. Our definitions and protocol designs are modular and lend themselves ideally as building blocks to future works in this area.

1.1 Our Techniques

We now give a technical overview of our results. We begin with a recap of the model and necessary definitions.

The Mixed Fault Model. Let us first revisit the standard security properties of BA (i.e., without omission faults): (1) *validity*: if all honest parties input v, all honest parties output v. (2) *consistency*: if an honest party i outputs v, then all honest parties output v. From the onset, it is clear that we cannot hope to achieve this definition for omission faulty parties. For example, a receive faulty party may not receive all of the necessary messages to output in protocol Π at all. Similarly, a send omission faulty party may not be able to share its input, so any validity property that is sensitive to its input cannot be achieved. On the other hand, we *can hope* to achieve a validity property that takes into account receive faulty parties' inputs as well as to guarantee output for parties that are

[3] Parties who are both send and receive omission faulty are counted twice in this bound.

only send faulty. Indeed, the protocols of Hauser et al. and Eldefrawy et al. satisfy these properties.

Zombies. Central to the protocol design of existing works is a means for receive omission faulty parties to detect that they are not receiving messages as they should. A self-detected party can then react by ceasing to propagate potentially incorrect information in the future. To this end, parties overlay communication with a protocol that constantly checks whether they receive messages from sufficiently many parties, i.e., $n-t-s$ many of them. If not, a party i can be certain that it is not receiving messages from at least one party who is neither malicious nor send omission faulty. Hence, i concludes that it is receive faulty and shuts itself off in the protocol so as to not cause further harm to the remaining honest parties. In line with existing works, we will refer to such parties as *zombies*. To deal with $t < \frac{n}{3}$ malicious faults, Hauser et al. now obtain a weak consensus protocol as follows.[4] Suppose that from party i's view, there is a value $b \in \{0,1\}$ that is supported by at least $n-s-r > 2t$ parties, whereas $1-b$ is supported by fewer than $t+1$ many parties. Then i can decide b, as it is sure that among the (at least) $2t+1$ supporters of b, there were $t+1$ honest parties. Those parties would have also communicated their support to all other parties, who, by the same rule, would not decide $1-b$, unless they are receive faulty. Moreover, if all honest parties input b to the protocol, b will always be the decided value for all honest parties and send omission faulty parties. Since zombie parties can detect themselves and cease to send messages, Hauser et al. now manage to transform the above weak consensus, into strong consensus via standard techniques.

Additional Challenges When $\frac{n}{3} \leq t < \frac{n}{2}$. Even when signatures are available, it is unclear how the above approach would be made to work. Note that the standard strategy in this setting (i.e., when there are no omission faults) is for parties to gather *certificates* of at least $t+1$ signatures on either b or $1-b$, which they then pass on to all other parties. Upon obtaining $t+1$ certificates on a value b and no certificate on a conflicting value $1-b$, a party i deems that it is safe to output b. Much as above, P would usually infer that at least one certificate on b was sent to it by another honest party, j, who also sent it to all other parties. Hence, no other party decides $1-b$.

This strategy is not applicable when there are send omission faults, as i's $t+1$ certificates could have been sent by t malicious parties and one send faulty party j. Contrary to an honest, party, j might not be able to pass on the certificate to all other honest parties, and so they may yet decide on $1-b$. It may seem tempting for i to just wait for more certificates. But, since zombies are still needed to make the remaining steps of the construction work, this is also not possible, since i, in the worst case, will receive messages from at most $t+1$ honest parties. Hence i may not decide on a value b, even if all honest parties input b to weak consensus. The issue with send faulty parties described above is

[4] Weak consensus is a precursor to full consensus in which any honest party outputs either some $y \in \{0,1\}$ or a special symbol $\perp$ (but no honest party outputs $1-y$). Moreover, if all honest parties input y, they all output y.

sidestepped by Eldefrawy et al. who consider spotty omissions that drop all or none of the messages in a single protocol round.

Ghosts. We now introduce our new tool that allows us to deal with fully fledged send omission faults. As with zombies, our main idea is for parties to self-detect whether or not they are send faulty and take measures to prevent themselves from causing further harm in the protocol. Looking ahead, we will again let parties who self-detect as send omission faulty silence themselves in all subsequent protocol steps. Different from zombies, however, self-detected send omission faulty parties will remain as silent observers in the protocol and output along with the honest parties. Therefore, we refer to such parties as *ghosts*.

Putting ghost parties to good use turns out to be subtle. To see the issue, consider the following scenario. In some round, a party i becomes aware that at least $n-t-s > t+r$ parties (and thus at least one honest party) haven't heard from it in the previous round and haven't become zombies. This can easily be achieved by running our protocol via an overlay in which parties confirm each others messages in each round. Party i promptly concludes that it must be send omission faulty and stops speaking in all further rounds. However, the problem is that i might already have caused a problematic situation. For example, i might have sent a certificate for some value b to a party j. Party j would like to conclude that b is a safe value to decide on. However, even if j sees $t+1$ certificates (including i's) for b in a given round, and no conflicting certificate for $1-b$, it still cannot do so, as i's message may not have arrived in that round at all other honest parties. As a result some of those parties might still deem $1-b$ a value that is safe to decide on.

To prevent this scenario, we introduce some extra rounds after every round of sending messages. During these rounds, parties confirm to each other that they didn't turn themselves into ghosts in any of the previous rounds. At the same time, parties are instructed to echo all of the messages they receive to all other parties and to keep confirming receipts of messages to each other in all rounds. In order to guarantee the delivery of a message, i sends the message once, and then sends it again if it hasn't become a ghost. Every party that receives the message during the first round also forwards that message in the second round. Party i knows that if it didn't become a ghost, at least one non-faulty party received its message in the first round and will forward that message to all parties. In addition, a party that receives messages from i in both rounds knows that every party will hear about the message for similar reasons (i.e., because i has not turned itself ghost). Using this technique, in addition to detecting its own failures, every party can also grade received messages according to how confident it is that every other party will receive them as well. Then, if some party receives a certificate with high confidence that every other party will do so as well, it can count it towards its tally of $t+1$ supporting certificates for a given value.

Putting Things Together: Undead Consensus. Using the above template, we use a modular approach to build protocols of increasingly strong consistency guarantees. Here, we follow a more or less standard structure of going from weak consensus to full consensus. We adapt all of our definitions to explicitly take into account *undead parties* (i.e., zombies or ghosts). This is important, because zombie parties cannot be expected to have the same consistency guarantees as the live (i.e., honest) ones, whereas ghost parties cannot be expected to have their inputs taken into account for protocols they participate in. Our definitions are tailored to modular protocol design and depart from prior works, in which these guarantees were left implicit. In particular, our definitions account for parties to be undead even before the protocol starts, which may occur as the result of running a previous subprotocol in the overall protocol stack. We believe that our modular notions will serve as important definitional pillars for future work in this area.

2 Model and Definitions

2.1 Network Model

We assume a network of n parties with point-to-point authenticated communication channels. In addition, we assume a PKI setup used for signing messages, meaning that each party i has a well-known public key pk_i associated with it and a signing key sk_i known only to it. Parties can sign a message using the Sign algorithm and verify signatures using the Verify algorithm. As is standard in this line of literature, we model the signatures as perfectly unforgeable. It is, however, straight-forward to instantiate signatures in any of our protocols with any existentially unforgeable signature signature scheme, in which case our properties hold against computationally bounded adversaries.

The network is assumed to be synchronous, meaning that there is a well-known upper bound Δ on message delay. Any message sent by an honest party at time t is delivered by time $t + \Delta$. In such systems, it is possible to define discrete communication rounds. All parties start each protocol at time 0, and then actions taken between time $(r-1)\Delta$ and $r\Delta$ are said to take place in round r. In particular, if a party sends a message at time $(r-1)\Delta$ in the beginning of round r, it is guaranteed to be delivered by time $r\Delta$, at the beginning of the next round. In the below protocols, each bullet-point defines the actions to be taken in the beginning of a specific round, unless specifically stated otherwise. When a bullet-point contains a call to a subprotocol that requires k rounds to complete, parties continue to the next bullet-point only after k rounds.

2.2 Adversary Model

The aim of this work is to deal with mixed-fault networks. The n parties can experience one of three types of faults/corruptions:

– **Send Omission Faults.** Send faulty parties follow the protocol description. For any message that a send faulty party sends, the adversary can choose not to deliver that message. We assume that there are at most s send faulty parties.

– **Receive Omission Faults.** Receive faulty parties follow the protocol description. For any message sent to a receive faulty party, the adversary can choose not to deliver that message. We assume that there are at most r receive faulty parties.
– **Byzantine/Malicious Faults.** A Byzantine/Malicious party can deviate from a protocol description arbitrarily. We assume that there are at most t Byzantine parties.

Throughout this work, we will assume that $n > 2t+s+r$. As shown by Eldefrawy, Loss, and Terner, this is the best-possible tolerance that can be achieved. The adversary is assumed to be *rushing* and *strongly adaptive*. This means that it can corrupt parties at any time throughout a protocol execution with one of the three types of corruptions explained above. In any round of a protocol execution, the adversary can observe the messages of all honest parties and then choose the messages of corrupt parties as well as what new parties to corrupt adaptively for that round. For send faulty parties, the adversary chooses which of their messages to deliver for that round, whereas for receive faulty parties, it chooses which of the (honest) messages to deliver to those parties. If a party newly becomes Byzantine and/or send faulty in a round, the adversary can erase (or replace, in case of a Byzantine corruption) any of the messages that party sent while it was honest, as long as those messages have not been delivered yet. Similarly, for a newly receive faulty party, the adversary may erase any of the messages that were sent to that party while it was still honest, as long as those messages have not been delivered. We assume the adversary to have full control over the network, subject to the constraint of delivering messages of honest and receive faulty parties within Δ time.

2.3 Zombies and Ghosts

Our overarching goal is constructing a mixed-fault tolerant consensus protocol, allowing all parties to agree on a value. As has been done in previous works, our protocols are designed in a way that an omission-faulty party should either behave correctly (or correctly enough), or find out that it is faulty and stop communicating in order not to cause harm.

Using the terminology of [28], if a party finds out that it is receive faulty, it becomes a “zombie”. This means that it sets a flag Z to the value true, stops sending messages, and eventually outputs the flag from the protocol in addition to any other value. Similarly, if a party sees that it is send faulty it becomes a “ghost” by setting a flag G to true, stops sending messages, and eventually outputs this flag as well. Parties also receive such flags as inputs, denoting whether they were zombies or ghosts in the beginning of the protocol, and output those flags in addition to any regular output in order to relay this information to any calling protocol. For simplicity, whenever a party sends a message via a protocol with a designated sender, we assume all parties participate in the protocol. In addition, whenever parties call an internal protocol, they also input their current Z, G flags to the called protocol.

We say that a party has become a zombie or a ghost during a protocol if it set its corresponding flag to the value true while executing the protocol. Similarly, we say that a party is a zombie or a ghost in the beginning of a protocol if its corresponding input flags are set to the value true. A party that has become a zombie or a ghost is said to be "undead", and a non-Byzantine party that isn't undead at a given moment is said to be alive.

2.4 Protocol Definitions

We now give basic protocol definitions used throughout this paper. Our definitions extend classical definitions in the literature with properties for undead parties. The original definitions without the notion of undead parties are provided in Appendix B.

No Living Undead Protocol. In all of the following definitions, we consider undead versions of classic protocols. That is, if parties output a value x in some protocol, in the undead version parties output x, z, g, with x being analogous to the original output, z being a flag indicating whether the party became a zombie while running the protocol, and g being a flag indicating whether it became a ghost. A general property we would like in any such protocol is that parties only become zombies or ghosts if they are indeed receive or send faulty, respectively. We formalize this notion in the following definition, indicating that the protocol does not produce any undead parties that are actually alive.

Definition 1. *Let Π be a protocol executed by parties $1, \ldots, n$, where every party outputs a triplet (x, z, g) such that z and g are boolean values. We say that Π is* (t, s, r)-no living undead (NLU) *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if a non-Byzantine party is alive in the beginning of the protocol and it outputs (x, z, g) such that $z = \mathsf{true}$ (resp. $g = \mathsf{true}$), then it is receive faulty (resp. send faulty).*

Undead Weak Multicast. An undead weak multicast protocol is a basic building block, replacing the simple multicast implementation. In the protocol, a known sender has a message to send to all parties. The sender then attempts to send the message to all parties, checking whether it should become a ghost and allowing parties to check whether they should become zombies. This primitive is weak in the sense that if a party does not become a ghost, it only knows that at least one nonfaulty party heard its message.

Definition 2. *Let Π be a protocol executed by parties $1, \ldots, n$, with a designated sender i^* starting with an input $m \neq \bot$. In addition, every party i has two values $z_i, g_i \in \{\mathsf{true}, \mathsf{false}\}$ as input. Every party outputs a triplet (x, z, g) such that x is either a possible message or $\bot$, and z, g are boolean values.*

- ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is nonfaulty or receive faulty and is alive in the beginning of the protocol,* *every non-Byzantine party either outputs (x, z, g) such that $x = m$, or becomes a zombie by the end of the protocol. In addition, if i^* is send faulty, no non-Byzantine party outputs (x, z, g) such that $x \notin \{m, \perp\}$.*

- ***Detection.*** *Π is* (t, s, r)-detecting *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is send faulty and it is alive at the end of the protocol, at least one nonfaulty party output (x, z, g) such that $x = m$.*
- ***Termination.*** *Π is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

If Π is (t, s, r)-valid*,* (t, s, r)-detecting*,* (t, s, r)-terminating*, and (t, s, r)-no living undead we say that it is a (t, s, r)-secure multicast protocol with undead parties.*

Undead Graded Multicast. Using the undead weak multicast primitive, it is possible to construct a slightly stronger multicast primitive. An undead graded multicast protocol also has a designated sender that attempts to send its message to all parties in such a way that parties can detect their own faults. In addition to outputting a message m, parties also output a grade y, intuitively indicating how confident they are that the the protocol succeeded. In such a protocol, if some party is very confident that the protocol succeeded for some non-Byzantine sender, all parties will receive a message from the sender, even if it is send faulty. This protocol is also "stronger" than the previous protocol in the sense that if a sender does not become a ghost, it knows that every party received its message.

Definition 3. *Let Π be a protocol executed by parties $1, \dots, n$, with a designated sender i^* starting with an input $m \neq \perp$. In addition, every party i has two values $z_i, g_i \in \{\mathsf{true}, \mathsf{false}\}$ as input. Every party outputs a tuple (x, y, z, g) such that x is either a possible message or $\perp$, $y \in \{0, 1, 2\}$ and z, g are boolean values.*

- ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is nonfaulty or receive faulty and is alive in the beginning of the protocol, every non-Byzantine party either outputs (x, y, z, g) such that $x = m, y = 2$, or becomes a zombie by the end of the protocol. In addition, if i^* is send faulty, no non-Byzantine party outputs (x, y, z, g) such that $x \notin \{m, \perp\}$.*
- ***Detection.*** *Π is* (t, s, r)-detecting *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is send faulty and it is alive at the end of the protocol, every nonfaulty party output (x, y, z, g) such that $x = m$ and $y \geq 1$.*
- ***Consistency.*** *Π is* (t, s, r)-consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is non-Byzantine, for every two non-Byzantine parties j, k that output (x_j, y_j, z_j, g_j) and (x_k, y_k, z_k, g_k) respectively, either $|y_j - y_k| \leq 1$, or at least one of the parties becomes a zombie by the end of the protocol. In addition, either $x_j = \perp$ and $y_j = 0$ or $x_j = m$.*

– ***Termination.*** *$\mathit{\Pi}$ is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

If $\mathit{\Pi}$ is (t, s, r)-valid*,* (t, s, r)-detecting*,* (t, s, r)-consistent*,* (t, s, r)-terminating*, and* (t, s, r)-no living undead *we say that it is a* (t, s, r)-(secure) *graded multicast with undead parties protocol.*

Note that in the above, the output of a party is x, y, z, g with z, g being boolean flags. For the no living undead property, we consider x, y as the first element of the output and omit the parentheses for convenience, and z, g as the two flags.

Undead Weak Consensus An undead weak consensus protocol is a mixed-fault version of a weak consensus protocol. In such a protocol, every party has a binary input, 0 or 1. The goal of the protocol is for there to be some value y such that every party either outputs y or $\perp$.

Definition 4. *Let $\mathit{\Pi}$ be a protocol executed by parties $1, \dots, n$, in which every party i starts with an input $v_i \neq \perp$. In addition, every party i has two values $z_i, g_i \in \{\mathsf{true}, \mathsf{false}\}$ as input. Every party outputs a triplet (x, z, g) such that x is either a possible input message or $\perp$, and z, g are boolean values.*

– ***Validity.*** *$\mathit{\Pi}$ is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if all parties that are alive in the beginning of the protocol have the same input v, every non-Byzantine party either outputs (x, z, g) such that $x = v$, or becomes a zombie by the end of the protocol.*
– ***Consistency.*** *$\mathit{\Pi}$ is* (t, s, r)-consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if two parties are either alive or ghosts at the end of the protocol and they output (x, z, g) and (x', z', g'), then either $x = x'$, $x = \perp$ or $x' = \perp$.*
– ***Termination.*** *$\mathit{\Pi}$ is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

If $\mathit{\Pi}$ is (t, s, r)-valid*,* (t, s, r)-consistent*,* (t, s, r)-terminating*, and* (t, s, r)-no living undead *we say that it is a* (t, s, r)-secure *undead weak consensus protocol.*

Undead Consensus Similarly, an undead consensus protocol is a mixed-fault version of a consensus protocol, in which all parties output the same value.

Definition 5. *Let $\mathit{\Pi}$ be a protocol executed by parties $1, \dots, n$, in which every party i starts with an input $v_i \neq \perp$. Every party outputs a triplet (x, z, g) such that x is a possible input value or $\perp$, and z, g are boolean values.*

– ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if all parties that are alive in the beginning of the protocol have the same input v, every non-Byzantine party either outputs (x, z, g) such that $x = v$, or becomes a zombie by the end of the protocol.*
– ***Consistency.*** *Π is* (t, s, r)-consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if two parties are either alive or ghosts at the end of the protocol and they output (x, z, g) and (x', z', g'), then $x = x'$.*
– ***Termination.*** *Π is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

If Π is (t, s, r)-valid*,* (t, s, r)-consistent*,* (t, s, r)-terminating*, and* (t, s, r)-no living undead *we say that it is a* (t, s, r)-secure *undead consensus protocol.*

Undead Common Coin. As done in previous works, in order to construct a consensus protocol with a constant expected number of rounds, we use an unbiasable common-coin protocol in each iteration of the protocol. Using the same construction as the ones in [4,11], we assume a common coin protocol, CoinFlip. This protocol can be constructed using a threshold signature scheme with unique signatures and a multicast protocol which allows parties to detect their own failures and turn into zombies or ghosts. Using this construction, the protocol informally achieves the following properties:

– Until at least one non-Byzantine party calls the common coin protocol for a given iteration, the output is distributed uniformly from the point of view of the adversary.
– All parties that are not zombies at the end of the protocol output the same value.

As done in other protocols, parties output a bit b, as well as two flags Z, G indicating whether they became zombies or ghosts throughout the protocol.

As described in [4], this protocol can be viewed as an ideal functionality. Parties can input a round number k into the functionality. Upon receiving the input k from $t + 1$ parties, the functionality uniformly samples a bit $b \in \{0, 1\}$ and sends k, b to all parties signifying that the coin's value for round k is b. Note that as described above, the value b is unpredictable from the point of view of Byzantine parties before $t + 1$ parties call the functionality for round k, which must include at least one non-Byzantine party. Considering also receive-faulty parties, the adversary can have the functionality send the message $\perp$ to receive-faulty parties instead of k, b, after which they can become zombies.

One common way to instantiate such a coin is by the use of threshold signatures with unique signatures. Using such a threshold signature scheme with a threshold of $t + 1$, a simple protocol for a Byzantine coin can be constructed by having all parties sign the round number and send the signature to all parties. Parties can then generate a unique threshold signature from any set of $t + 1$

such signature, and use a randomness extractor such as a random oracle to get a coin from the signature. In our setting, parties multicast these messages using the undead weak multicast protocol. Every party that doesn't become a zombie receives the multicasts sent by the $t+1$ nonfaulty parties (in addition to other possible signatures), and thus can compute the coin correctly. Note that the unique signatures of the scheme allow taking any set of $t+1$ correct signatures and computing the same threshold signature from them, meaning that parties will have a consistent coin even when $n > 2t + s + r$.

3 Protocols

In this section we construct protocols solving the tasks defined in Section 2.4. We generally follow many of the ideas and structure of Eldefrawy et al. [11]. Importantly, we both adjust the protocols to deal with ghosts and slightly change their protocol design. We use an undead graded multicast protocol instead of a weak broadcast protocol, which does not allow Byzantine parties to equivocate, but does not output grades as well as messages. In addition, we reduce our weak consensus protocol directly to a consensus protocol while they construct a graded consensus protocol as a step in between the two protocols.

In all of the following protocols, when we say that a party calls a protocol, we assume that it also inputs its current flags Z and G to the protocols without explicitly stating that it does so. In addition parties always participate in multicast and broadcast protocols with other parties as senders without explicitly stating so.

The proofs of all theorems in this section are provided in Appendix A

3.1 Undead Weak Multicast

The simplest primitive constructed in this work is an undead weak multicast protocol. This protocol is an adaptation of the $\mathsf{FixReceive}$ protocol of [28] and the all-to-all $\mathsf{FixReceive}$ protocol of [11]. In the protocol, the sender i^* attempts to send its message m to all parties, while allowing parties to detect their own faults. This is done by i^* sending the message m along with a signature σ to all parties, which then forward any message they received, or $\perp$ if they received no message. Parties then output m' if they receive this message along with a verifying signature from any party. Every party that isn't receive faulty knows that it should receive some message (either m or $\perp$) from all of the nonfaulty and receive faulty parties, totalling in $n - t - s$ parties. Therefore, if some party receives fewer than $n - t - s$ messages, it knows that it should become a zombie. In order for i^* to be able to detect its own faults, non-zombie parties inform it if they received no message and resorted to outputting $\perp$. If i^* receives $t+1$ such messages, it knows that at least one non-Byzantine party output $\perp$, and can conclude that it should become a ghost. The full description of the protocol is provided in Fig. 1 and the proof of Theorem 1 is provided in Appendix A.1.

Theorem 1. *The protocol described in Fig. 1 is a (t, s, r)-secure undead weak multicast protocol if $n > 2t + s + r$.*

Undead Weak Multicast, sender i^* with input m, every party j with inputs z_j, g_j

1. Every party j sets $\mathsf{Z}_j = z_j, \mathsf{G}_j = g_j$. Party i^* sends $m, \mathsf{Sign}(\mathsf{sk}_{i^*}, m)$ to every party if it is alive.
2. If a party receives m, σ from i^* such that $\mathsf{Verify}(\mathsf{pk}_{i^*}, m, \sigma) = 1$, forward m, σ to every j. Otherwise, send $\perp$ to every j.
3. Every $j \neq i^*$ that received fewer than $n - t - s$ messages becomes a zombie by setting $\mathsf{Z}_j = \mathsf{true}$ and outputs $(\perp, \mathsf{Z}_j, \mathsf{G}_j)$. If j hasn't become a zombie and it received at least one pair m, σ such that $\mathsf{Verify}(\mathsf{pk}_{i^*}, m, \sigma) = 1$, it outputs $(m, \mathsf{Z}_j, \mathsf{G}_j)$ (choose arbitrarily). If j hasn't become a zombie or output a tuple, it sends "no output" to i^* and outputs $(\perp, \mathsf{Z}_j, \mathsf{G}_j)$ (note that zombies don't send this message). In all cases, parties wait an additional round before terminating.
4. If i^* receives at least $t+1$ "no output" messages, it becomes a ghost by setting $\mathsf{G}_{i^*} = \mathsf{true}$. Regardless of whether it has become a ghost, if $\mathsf{Z}_{i^*} = \mathsf{false}$, i^* outputs $(m, \mathsf{Z}_{i^*}, \mathsf{G}_{i^*})$ and otherwise it outputs $(\perp, \mathsf{Z}_{i^*}, \mathsf{G}_{i^*})$.

Fig. 1. An undead weak multicast protocol

3.2 Undead Graded Multicast

We turn to construct a slightly stronger version of an undead weak multicast protocol. In the previous protocol, if the sender party did not become a ghost, it only knows that one nonfaulty party received its message. The undead graded multicast protocol described in Fig. 2 allows a non-ghost sender to know that all parties received its message. In addition, parties output a message x as well as a grade $y \in \{0, 1, 2\}$. Informally, the grades represent parties' confidence that all parties heard the message sent by the sender. A grade of 2 guarantees that all parties hear the message sent by a non-Byzantine sender, and a grade of 0 is only given in the case that some party heard no value and had to output $\perp$.

The protocol consists of two rounds. In the first round, i^* signs its message m and sends m, σ to all parties with the undead weak multicast protocol described above. In the second round, parties forward the received message and signature if they received such values, and $\perp, \perp$ otherwise. Parties then output $m, 2$ if they heard (m, σ) from i^* in both rounds; $m, 1$ if they heard (m, σ) from some party in the second round; and $\perp, 0$ if they heard no message with a verifying signature. Intuitively, if some party outputs $m, 2$, it knows that i^* did not become a ghost in the first round because it sent a message in the second round. This means that some nonfaulty party heard that message and forwarded it to all parties, which either receive it and output m with a grade of at least 1, or become zombies. It is important to note that we require that if some party is receive faulty, but is alive in the beginning of the protocol, all parties receive its message and output $m, 2$. In order to make sure this happens, if i^* finds out that it is a zombie after the first round, it sends its message again in the second round and only then becomes a zombie. The proof of Theorem 2 is provided in Appendix A.2.

Undead Graded Multicast, sender i^* with input m, every j with inputs z_j, g_j

1. Every party j sets $\mathsf{Z}_j = z_j, \mathsf{G}_j = g_j$. Party i^* sends $m, \mathsf{Sign}(\mathsf{sk}_{i^*}, m)$ using an undead weak multicast protocol if it is alive.
2. Every party j checks whether it became a zombie in the previous round, and if not forwards whichever message it received. As an exception, i^* sends its message again even if it became a zombie (but not if it became a ghost). More precisely, let $x_j, \mathsf{Z}'_j, \mathsf{G}'_j$ be j's output in the previous round. If $\mathsf{Z}'_j = \mathsf{true}$ for some $j \neq i^*$, j sets $\mathsf{Z}_j = \mathsf{true}$ and if $\mathsf{G}'_j = \mathsf{true}$ (for any j), j sets $\mathsf{G}_j = \mathsf{true}$. Then, if $\mathsf{Z}'_{i^*} = \mathsf{true}$, i^* first sends (m, σ) again using an undead weak multicast protocol and then sets $\mathsf{Z}_{i^*} = \mathsf{true}$. Every $j \neq i^*$ checks if $x_j = (m', \sigma')$ such that $\mathsf{Verify}(\mathsf{pk}_{i^*}, m', \sigma') = 1$. If this is the case, j sets $(m_j, \sigma_j) = (m', \sigma')$, and otherwise $(m_j, \sigma_j) = (\bot, \bot)$. Every $j \neq i^*$ that is alive sends (m_j, σ_j) using an undead weak multicast protocol.
3. For every pair of parties j, k, define $x_{j,k}, \mathsf{Z}_{j,k}, \mathsf{G}_{j,k}$ to be the value j received from k in the previous round. If j has become a ghost or a zombie in any of the calls to the undead weak multicast protocol, it becomes one in the undead graded multicast protocol as well. That is, it sets $\mathsf{G}_j = \mathsf{true}$ if $\mathsf{G}_{j,j} = \mathsf{true}$ and sets $\mathsf{Z}_j = \mathsf{true}$ if there exists a k such that $\mathsf{Z}_{j,k} = \mathsf{true}$. Then, if j has $\mathsf{Z}_j = \mathsf{true}$, it outputs $\bot, 0, \mathsf{Z}_j, \mathsf{G}_j$. If $\mathsf{Z}_j \neq \mathsf{true}$, $(m_j, \sigma_j) \neq (\bot, \bot)$ and j received $x_{j,i^*} = (m_j, \sigma_j)$ from i^* in the previous round as well, it outputs $m_j, 2, \mathsf{Z}_j, \mathsf{G}_j$. If j did not output a value yet and there exists some k such that $x_{j,k} = (m', \sigma')$ and $\mathsf{Verify}(\mathsf{pk}_{i^*}, m', \sigma') = 1$, j outputs $m', 1, \mathsf{Z}_j, \mathsf{G}_j$ and if no such k exists it outputs $\bot, 0, \mathsf{Z}_j, \mathsf{G}_j$.

Fig. 2. An undead graded multicast protocol

Theorem 2. *The protocol described in Fig. 2 is a (t, s, r)-secure undead graded multicast protocol if $n > 2t + s + r$.*

3.3 Undead Weak Consensus

Using the undead graded multicast protocol, we construct an undead weak consensus protocol with binary inputs. In this protocol, parties send each other their signed inputs. Each party i then collect all of the signed inputs it heard in a set S_i and forwards the set to all parties using the undead graded multicast protocol. These sets are used as certificates, proving that a large enough number of parties reported having a certain input. Generally, we say that S_i is a certificate for the value x if it contains signatures from $t + 1$ parties on the value x. Parties then output a value x if it is possible that all nonfaulty parties had the input x. They check whether this is possible by seeing if at least $t + 1$ parties signed a value x and at most t parties signed the opposite value $1 - x$ (this can be generalized to $t + 1$ signatures for x and at most t signatures on any other value).

The protocol, described in Fig. 3, consists of two rounds. Each party starts by signing its input and sending it to all parties. Every party then defines S_i to be the set of signed messages it receives and forwards it using an undead graded multicast protocol. If a party receives a certificate S_j for a value v with a grade of 2 from $t+1$ parties j, and it does not receive a conflicting certificate S_k for the value $1-v$ from any party k, it outputs v. Intuitively, a party that outputs v knows that at least one of the certificates that was received with grade 2 was sent by a non-Byzantine party. All parties receive that certificate with grade 1 or greater, or become zombies. Therefore, a party that chooses to output v knows that all other parties will either become zombies or hear about at least one certificate for v and thus won't output $1-v$. The proof of Theorem 3 is provided in Appendix A.3.

Undead Weak Consensus, each party i has input $x_i \in \{0,1\}$ and two inputs z_i, g_i

1. Each party i sets $\mathsf{Z}_i = z_i, \mathsf{G}_i = g_i$. Each party i that is alive at this time sends $x_i, \mathsf{Sign}(\mathsf{sk}_i, x_i)$ to all other parties.
2. Each party i defines the set S_i to contain tuples of the form (j, x, σ) such that i received the message x, σ from j and $\mathsf{Verify}(\mathsf{pk}_j, x, \sigma) = 1$ and calls the undead graded multicast protocol with input S_i.
3. For every party i, j, define $S_{j,i}, y_{j,i}, \mathsf{Z}_{j,i}, \mathsf{G}_{j,i}$ to be i's output from the undead graded multicast call with j as sender. If there exists some j such that $\mathsf{Z}_{j,i} = \mathsf{true}$, i sets $\mathsf{Z}_i = \mathsf{true}$, and if there exists some j such that $\mathsf{G}_{j,i} = \mathsf{true}$, i sets $\mathsf{G}_i = \mathsf{true}$. A set S with enough signatures on the value v is called a certificate for v. That is, S is said to be a certificate for v if $|\{k | \exists \sigma \ s.t. (k, v, \sigma) \in S \wedge \mathsf{Verify}(\mathsf{pk}_k, v, \sigma) = 1\}| \geq t+1$. Each party i checks if there is a value v_i such that there are at least $t+1$ parties j from which it received a certificate $S_{j,i}$ with grade 2, and checks that none of the $S_{j,i}$ sets it received contained a certificate for the value $1 - v_i$, before outputting v_i. More precisely if there exists some value v such that for $t+1$ parties j, $S_{j,i}$ is a certificate for v and $y_{j,i} = 2$; and for every k, $S_{k,i}$ is not a certification for $1-v$ (with any grade $y_{k,i}$); then i sets $v_i = v$. Otherwise, it sets $v_i = \perp$. Finally, i outputs $v_i, \mathsf{Z}_i, \mathsf{G}_i$.

Fig. 3. An undead weak consensus protocol

Theorem 3. *The protocol described in Fig. 3 is a (t, s, r)-secure undead weak consensus protocol if $n > 2t + s + r$.*

3.4 Undead Consensus

Finally, we use the undead weak consensus protocol above to construct an undead consensus protocol with expected constant round complexity, similarly to the construction of [11]. Parties start by setting a local variable v_i to be their input, and then perform several actions in a loop until terminating. In the beginning of each iteration, parties call an undead weak consensus protocol with v_i as input, and after completing that protocol also call a common coin protocol. If some party outputs $u \neq \perp$ from the protocol, it adopts it as its value by updating $v_i = u$. On the other hand, if $u = \perp$, i defaults to using the output from the common coin protocol as its input to the next round. If some party outputs the same value u from both the weak consensus protocol and from the common coin protocol, it signs the value and send the signature to all parties. At this point, it already knows that v is going to be the output from the protocol and could output v if desired. Parties store all signatures received throughout the protocol in a set D_v. Once parties receive $t+1$ such signatures on v, they forward D_v to all parties and output v. They then continue for one more iteration of the protocol in order to help parties complete the protocol without becoming zombies or ghosts.

The protocol works by having parties check whether they already agree on a value v using the weak consensus protocol. If all parties have the same input to the protocol, they will output it from the weak consensus protocol and always have their v_i variables equal v. Unfortunately, parties don't know that the weak consensus protocol succeeded, so they also need to choose when to complete the protocol. This is done by also flipping a common coin, and checking whether its result b equals the output from the weak consensus protocol u. Every party that sees that $b = u$ knows that every party j either sets its v_j variable to u because it output u from the weak consensus protocol, or because it output $b = u$ from the common coin protocol. From this point on, all parties have the same input u in each iteration and will output u from any subsequent call to the weak consensus protocol. Note that this event also takes place if all parties output $\perp$ from the weak consensus protocol and then adopted the value of the common coin as their inputs to the next iteration. Parties now simply need to wait for b to equal u, at which point all parties will send signatures and complete the protocol. A formal description of the protocol is provided in Fig. 4 and a proof of Theorem 4 is provided in Appendix A.4.

Theorem 4. *The protocol described in Fig. 4 is a (t, s, r)-secure undead consensus protocol if $n > 2t + s + r$.*

4 Related Work and Future Directions

Byzantine Agreement, originally defined as the Byzantine Generals problem, has been researched as a foundational task in the field of distributed computing. Early results showed that the task is only solveable in synchronous systems with t Byzantine parties and without a PKI setup if $n > 3t$ and with a PKI setup if $n > 2t$ [16]. The highly related task of authenticated broadcast has been shown to be possible as long as $n > t$ [10]. Following that, research into the task of consensus in synchronous systems with omission faults yielded protocols that are resilient to any number of omission faults if the faulty parties aren't required to output correct values [20], and in the presence of k omission faults (of either type) if $n > 2k$ when they are required to output correct values [19,21].

Undead Consensus, each party i has input $x_i \in \{0,1\}$ and two inputs z_i, g_i

- Set $v_i = x_i$, $\mathsf{Z}_i = z_i, \mathsf{G}_i = g_i$, $k = 0$ and $D_0 = D_1 = \emptyset$, then loop until terminating:
  1. Call the undead weak consensus protocol with input v_i. Define $u_i, \mathsf{Z}'_i, \mathsf{G}'_i$ to be the output from the undead weak consensus call. If $\mathsf{Z}'_i = \mathsf{true}$, set $\mathsf{Z}_i = \mathsf{true}$ and if $\mathsf{G}'_i = \mathsf{true}$, set $\mathsf{G}_i = \mathsf{true}$.
  2. Update $k = k+1$ and invoke $\mathsf{CoinFlip}(k)$. Define $b, \mathsf{Z}'_i, \mathsf{G}'_i$ to be the output of the protocol. If $\mathsf{Z}'_i = \mathsf{true}$, set $\mathsf{Z}_i = \mathsf{true}$ and if $\mathsf{G}'_i = \mathsf{true}$ set $\mathsf{G}_i = \mathsf{true}$. If $u_i \neq \perp$, update $v_i = u_i$. If also $u_i = b$, send the message $v_i, \mathsf{Sign}(\mathsf{sk}_i, v_i)$ to all parties. On the other hand, if $u_i = \perp$, update $v_i = b$.
  3. If for some $v \in \{0,1\}$, $|D_v| \geq t+1$, send v, D_v to all parties and continue for one more iteration of the loop but stop updating the flags $\mathsf{Z}_i, \mathsf{G}_i$. Output $v, \mathsf{Z}_i, \mathsf{G}_i$ and terminate at the end of the next iteration of the loop.
- If at any point in the protocol, $\mathsf{Z}_i = \mathsf{true}$, send "Zombie" to all parties, output $\perp, \mathsf{Z}_i, \mathsf{G}_i$ and terminate. If at any point in the protocol a "Zombie" message is received from some party j, act as if it forwards $\perp$ messages in every undead weak multicast protocol from this point on, or messages corresponding to receiving no values in other protocols. If at any point $\mathsf{G}_i = \mathsf{true}$, stop sending any message in the protocol, but continue processing messages.
- If at any point in the protocol, a v, σ message is received from some party j such that $\mathsf{Verify}(\mathsf{pk}_j, v, \sigma) = 1$ and there is no tuple of the form $(j, \sigma') \in D_v$, add (j, σ) to D_v.
- If at any point in the protocol, a message v, D'_v is received such that D'_v is a set of tuples of the form (j, σ), for every $(j, \sigma) \in D'_v$, $\mathsf{Verify}(\mathsf{pk}_j, \sigma, v) = 1$ and $|\{j | \exists \sigma \; s.t. \; (j, \sigma) \in D'_v\}| \geq t+1$, set $D_v = D'_v$.

Fig. 4. An undead consensus protocol

A long line of research has been done on consensus in systems with mixed faults. The earliest works dealing with mixed faults considered a mix of t Byzantine faults and k non-malicious faults, consisting of both crash and omission faults. Crash faults are relatively mild, allowing the adversary to crash parties and stop them from sending or receiving any messages in the protocol. In the below discussion, t is always considered as the number of Byznatine faults, and k is the number of non-malicious faults when the works do not specify any further or allow for a mix of non-malicious faults. When works specify the exact number of crash, receive-omission and send-omission faults we specify these numbers as c, r and s respectively.

Early works such as that of Siu, Chin and Yang [24] a synchronous consensus protocol when $n > 3f + k$. A similar work by Garay and Perry [13] constructed a similar protocol with $n > 3f + c$. Additional works such as that of Thambidurai and Park [25] and of Lincoln and Rushby [18] constructed broadcast protocols for $n > 3t + 2f + k$, when k parties may have non-malicious faults and f parties may deviate from the protocol, but must send the same message to all parties in each round. The work of Hauser, Maurer and Zikas [28] presents several protocols, including a consensus, a broadcast and a secure function evaluation protocol under the assumption that $n > 3t + r + s$, also using the notion of zombies. More recently, the works of Eldefrawy, Loss and Terner [11] and of Abraham, Dolev, Kagan and Stern [1] construct mixed-fault consensus and state machine repliacation (SMR) protocols in an authenticated setting. Abraham *et al.* construct authenticated consensus and SMR protocols when $n > 2t + c$, only allowing crash faults. On the other hand, Elderfrawy *et al.* construct an authenticated consensus protocol for $n > 2t + 2s + r$. They also improve their resilience to $n > 2t + s + r$ when dealing only with *spotty* omission faults in which either all messages sent by a party are delivered in a given round, or none of them are. Our work adapts many of the ideas of the work of Elderfrawy *et al.*, achieving a resilience of $n > 2t + s + r$ while removing the additional assumptions on fault structure.

More research has been done into mixed faults in partially synchronous systems, in which the network starts off as an unstable network with asynchronous message delivery, but after some point in time (unknown to the participating parties), all messages are delivered within a well known time bound. The Scrooge [22], Upright [7] and SBFT [14] protocols are authenticated mixed-fault protocols for partially synchronous systems which are secure as long as $n > 4t + 2c$, $n > 3t + 2k$ and $n > 3f + 2c$ respectively. Some protocols achieve increased resilience by disallowing some actions by the adversary. Some of the works [6,8,9,17,26] make use of specialized hardware to make sure that the adversary does not act in a Byzantine manner in important code sections and de-facto make it an omission or crash party that can only choose not to send incorrect messages messages. Additional approaches include protocols which assume that specific parties with special roles are not Byzantine [23] and protocols which can deal with either Byzantine or crash faults, but not with both at the same time [3].

Future Directions This work intends to simply show that authenticated consensus is possible in synchronous systems when $n > 2t + s + r$, matching the known lower bound. Seeing as this is simply a possibility result, this work adapted techniques and structures while making small adjustments for clarity and simplicity. Some of these adjustments included removing unnecessary layers in the protocol stack (mentioned in Section 3), which had the nice side effect of making the protocol slightly more efficient. A natural future line of work deals with improving the efficiency of such protocols. For example, it could be possible to improve the round complexity of the protocol by breaking down the modular structure of the protocols and bundling several protocol messages together. On the other hand, it is very likely that improvements can be made in the number of messages or bits sent by parties by using stronger cryptographic primitives such as threshold signatures, as seen in modern Byzantine consensus protocols such as HotStuff [27] and VABA[2].

Another avenue of research is forgoing the use of threshold cryptography in the common coin protocol. This can be done either by having a $t + r + s + 1$-iteration consensus protocol with a rotating leader, as done in [28], or by generating randomness in a different way. For example, classic Byzantine agreement protocols such as those of Feldman and Micali [12] and Katz and Koo [15] use verifiable secret sharing in order to generate randomness, and constructing such protocols for the authenticated mixed-fault setting can be of independent interest.

Finally, using this protocol, it could be possible to construct more complex primitives such as state machine replication and secure multiparty computation protocols in mixed-fault settings with $n > 2t + s + r$. In a state machine replication protocol, parties agree on a (possibly infinite) log of values, with a numbered slot for each value. Well-known constructions, such as the PBFT protocol of Catsro and Liskov [5] use many instances of byzantine agreement protocols, and the recent work of Abraham et al. [1] constructed such protocols to settings where parties may either be Byzantine or crash. In addition, MPC protocols have been constructed using consensus protocols, allowing parties to securely compute arbitrary functions (or interactive functionalities). Hauser et al. construct such a protocol in an unauthenticated mixed-fault setting with $n > 3t + s + r$ [28] using their consensus protocol, and it could be possible to adapt synchronous cryptographic MPC protocols to mixed-fault protocols using a consensus protocol as an important building block.

Acknowledgements

Julian Loss was supported by the European Union, ERC-2023-STG, Project ID: 101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them. Gilad Stern was supported by the HUJI Federmann Cyber Security Research Center in conjunction with the Israel National Cyber Directorate (INCD) in the Prime Minister's Office.

References

1. 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`
2. Abraham, I., Malkhi, D., Spiegelman, A.: Validated asynchronous byzantine agreement with optimal resilience and asymptotically optimal time and word communication. CoRR **abs/1811.01332** (2018), `http://arxiv.org/abs/1811.01332`

3. Bessani, A.N., Sousa, J., Alchieri, E.A.P.: State machine replication for the masses with BFT-SMART. In: 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014. pp. 355–362. IEEE Computer Society (2014). https://doi.org/10.1109/DSN.2014.43, https://doi.org/10.1109/DSN.2014.43
4. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: Hofheinz, D., Rosen, A. (eds.) Theory of Cryptography - 17th International Conference, TCC 2019, Nuremberg, Germany, December 1-5, 2019, Proceedings, Part I. Lecture Notes in Computer Science, vol. 11891, pp. 131–150. Springer (2019). https://doi.org/10.1007/978-3-030-36030-6_6, https://doi.org/10.1007/978-3-030-36030-6_6
5. Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Seltzer, M.I., Leach, P.J. (eds.) Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999. pp. 173–186. USENIX Association (1999), https://dl.acm.org/citation.cfm?id=296824
6. Chun, B., Maniatis, P., Shenker, S., Kubiatowicz, J.: Attested append-only memory: making adversaries stick to their word. In: Bressoud, T.C., Kaashoek, M.F. (eds.) Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007. pp. 189–204. ACM (2007). https://doi.org/10.1145/1294261.1294280, https://doi.org/10.1145/1294261.1294280
7. Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riché, T.: Upright cluster services. In: Matthews, J.N., Anderson, T.E. (eds.) Proceedings of the 22nd ACM Symposium on Operating Systems Principles 2009, SOSP 2009, Big Sky, Montana, USA, October 11-14, 2009. pp. 277–290. ACM (2009). https://doi.org/10.1145/1629575.1629602, https://doi.org/10.1145/1629575.1629602
8. Correia, M., Lung, L.C., Neves, N.F., Veríssimo, P.: Efficient byzantine-resilient reliable multicast on a hybrid failure model. In: 21st Symposium on Reliable Distributed Systems (SRDS 2002), 13-16 October 2002, Osaka, Japan. pp. 2–11. IEEE Computer Society (2002). https://doi.org/10.1109/RELDIS.2002.1180168, https://doi.org/10.1109/RELDIS.2002.1180168
9. Correia, M., Neves, N.F., Veríssimo, P.: How to tolerate half less one byzantine nodes in practical distributed systems. In: 23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil. pp. 174–183. IEEE Computer Society (2004). https://doi.org/10.1109/RELDIS.2004.1353018, https://doi.org/10.1109/RELDIS.2004.1353018
10. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045, https://doi.org/10.1137/0212045
11. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy, June 20–23, 2022, Proceedings. p. 684–704. Springer-Verlag, Berlin, Heidelberg (2022). https://doi.org/10.1007/978-3-031-09234-3_34, https://doi.org/10.1007/978-3-031-09234-3_34
12. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA. pp. 148–161. ACM (1988). https://doi.org/10.1145/62212.62225, https://doi.org/10.1145/62212.62225

13. 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
14. Golan-Gueta, G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M.K., Seredinschi, D., Tamir, O., Tomescu, A.: SBFT: A scalable and decentralized trust infrastructure. In: 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2019, Portland, OR, USA, June 24-27, 2019. pp. 568–580. IEEE (2019). https://doi.org/10.1109/DSN.2019.00063, https://doi.org/10.1109/DSN.2019.00063
15. Katz, J., Koo, C.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (ed.) Advances in Cryptology - CRYPTO 2006, 26th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4117, pp. 445–462. Springer (2006). https://doi.org/10.1007/11818175_27, https://doi.org/10.1007/11818175_27
16. 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
17. Levin, D., Douceur, J.R., Lorch, J.R., Moscibroda, T.: Trinc: Small trusted hardware for large distributed systems. In: Rexford, J., Sirer, E.G. (eds.) Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA. pp. 1–14. USENIX Association (2009), http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf
18. Lincoln, P., Rushby, J.M.: A formally verified algorithm for interactive consistency under a hybrid fault model. In: Digest of Papers: FTCS-23, The Twenty-Third Annual International Symposium on Fault-Tolerant Computing, Toulouse, France, June 22-24, 1993. pp. 402–411. IEEE Computer Society (1993). https://doi.org/10.1109/FTCS.1993.627343, https://doi.org/10.1109/FTCS.1993.627343
19. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings. p. 212. IEEE Computer Society (2003). https://doi.org/10.1109/IPDPS.2003.1213388, https://doi.org/10.1109/IPDPS.2003.1213388
20. 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
21. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: 9th Pacific Rim International Symposium on Dependable Computing (PRDC 2002), 16-18 December 2002, Tsukuba-City, Ibarski, Japan. pp. 221–228. IEEE Computer Society (2002). https://doi.org/10.1109/PRDC.2002.1185641, https://doi.org/10.1109/PRDC.2002.1185641
22. Serafini, M., Bokor, P., Dobre, D., Majuntke, M., Suri, N.: Scrooge: Reducing the costs of fast byzantine replication in presence of unresponsive replicas. In: Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2010, Chicago, IL, USA, June 28 - July 1 2010. pp. 353–362.

IEEE Computer Society (2010). https://doi.org/10.1109/DSN.2010.5544295, https://doi.org/10.1109/DSN.2010.5544295

23. Serafini, M., Suri, N.: The fail-heterogeneous architectural model. In: 26th IEEE Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, October 10-12, 2007. pp. 103–113. IEEE Computer Society (2007). https://doi.org/10.1109/SRDS.2007.33, https://doi.org/10.1109/SRDS.2007.33
24. 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
25. Thambidurai, P.M., Park, Y.: Interactive consistency with multiple failure modes. In: Seventh Symposium on Reliable Distributed Systems, SRDS 1988, Columbus, Ohio, USA, October 10-12, 1988, Proceedings. pp. 93–100. IEEE Computer Society (1988). https://doi.org/10.1109/RELDIS.1988.25784, https://doi.org/10.1109/RELDIS.1988.25784
26. Veronese, G.S., Correia, M., Bessani, A.N., Lung, L.C., Veríssimo, P.: Efficient byzantine fault-tolerance. IEEE Trans. Computers **62**(1), 16–30 (2013). https://doi.org/10.1109/TC.2011.221, https://doi.org/10.1109/TC.2011.221
27. Yin, M., Malkhi, D., Reiter, M.K., Golan-Gueta, G., Abraham, I.: Hotstuff: BFT consensus with linearity and responsiveness. In: Robinson, P., Ellen, F. (eds.) Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019. pp. 347–356. ACM (2019). https://doi.org/10.1145/3293611.3331591, https://doi.org/10.1145/3293611.3331591
28. Zikas, V., Hauser, S., Maurer, U.M.: Realistic failures in secure multi-party computation. In: Reingold, O. (ed.) Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5444, pp. 274–293. Springer (2009). https://doi.org/10.1007/978-3-642-00457-5_17, https://doi.org/10.1007/978-3-642-00457-5_17

A Security Proofs

This section includes security proofs not included in the body of the paper.

A.1 Undead Weak Multicast

Theorem 1. *The protocol described in Fig. 1 is a (t, s, r)-secure undead weak multicast protocol if $n > 2t + s + r$.*

Proof. **Validity.** If i^* is nonfaulty or is receive faulty and alive in the beginning of the protocol, it sends m, σ to every party and all nonfaulty parties receive it and forward it. Every party that isn't receive faulty receives those messages. Since i^* is non-Byzantine, it only sends one verifying signature and thus this is the only received value, which will then be output. In addition, if a receive faulty party j receives fewer than $n - t - s$ messages, it becomes a zombie. Otherwise, it receives at least $n - t - s \geq 2t + s + r + 1 - t - s = t + r + 1$ messages, and at least one of those messages was sent by a party that is neither Byzantine nor receive faulty. Those parties receive the message m, σ from i^* and forward it, and thus every $j \neq i^*$ outputs m. In addition, if i^* isn't a zombie by the end of the protocol, it outputs $(m, \mathsf{Z}_{i^*}, \mathsf{G}_{i^*})$ as well. Finally, if i^* is send faulty, it only signs the message m. This means that no nonfaulty party receives any other signed message from i^*, and thus output either m or $\bot$.

Detection. If i^* did not become a ghost, then it received "no output" from fewer than $t + 1$ parties. It receives all messages from all $t + 1$ nonfaulty parties, so at least one of those parties did not send "no output", and has thus output $m, \mathsf{Z}, \mathsf{G}$.

Termination. All parties terminate after exactly 3 rounds.

No Living Undead. Any party that isn't receive faulty receives messages from at least $n - t - s$ parties in round 2. Therefore, only receive faulty parties become zombies. If i^* is nonfaulty or receive faulty, then all nonfaulty parties receive m, σ in round 1 and forward it. Every non-Byzantine party that hasn't become a zombie received $n - t - s$ messages, i.e. at least $t + r + 1$. Therefore, at least one of those messages was from a nonfaulty or send faulty party. As stated above, it received m, σ and thus did not send "no output". This means that i^* receives at most t messages of the form "no output" from Byzantine parties and does not become a ghost.

A.2 Undead Graded Multicast

Theorem 2. *The protocol described in Fig. 2 is a (t, s, r)-secure undead graded multicast protocol if $n > 2t + s + r$.*

Proof. **Validity.** If i^* is nonfaulty or is receive faulty and alive in the beginning of the protocol, it sends m, σ using an undead weak multicast protocol. The protocol has no living undead, and thus i^* does not become a ghost, and sends the same message again in the second round. From the Validity property of that protocol, every non-Byzantine party either outputs $(m, \sigma), \mathsf{Z}, \mathsf{G}$ in both rounds, or becomes a zombie. Therefore, every party that didn't become a zombie received both messages and output $m, 2, \mathsf{Z}, \mathsf{G}$. In addition, if i^* is send faulty, it only signs m and thus no party receives a message $m' \neq m$ with a verifying signature. Therefore, every party outputs $x, y, \mathsf{Z}, \mathsf{G}$ such that $x = m$ or $x = \perp$.

Detection. If i^* is send faulty and alive at the end of the protocol, it did not become a ghost after sending (m, σ) for the first time. From the Detection property of the undead weak multicast protocol, at least one nonfaulty party j received that message and forwarded it using the undead weak multicast protocol in the second round. From the Validity property of the protocol, every party that doesn't become a zombie in the protocol receives (m, σ) from j and outputs $m, 1, \mathsf{Z}, \mathsf{G}$ if it hasn't received (m, σ) from i^* and output $m, 2, \mathsf{Z}, \mathsf{G}$ instead.

Consistency. Assume i^* is not Byzantine and that some party j output $m, 2, \mathsf{Z}, \mathsf{G}$ with $\mathsf{Z} = \mathsf{false}$. This means that j received the same message (m, σ) from i^* in both calls to the undead weak multicast protocol. From the Validity property, i^* must have sent those messages, and a non-Byzantine party only sends the second message if it does not become a ghost during the first call to the protocol. As argued above, at least one nonfaulty party receives that message and forwards it to all parties, either because of the Validity property if i^* is nonfaulty or receive faulty, or from the Detection property if it is send faulty. Every party either becomes a zombie or receives the message, and thus outputs $m, 1, \mathsf{Z}, \mathsf{G}$ if it hasn't also received (m, σ) from i^* in the second round, causing it to output $m, 2, \mathsf{Z}, \mathsf{G}$ instead. This means that for every nonfaulty j, k that don't become zombies, $y_j, y_k \in \{1, 2\}$ and thus $|y_j - y_k| \leq 1$. Note that if no nonfaulty party outputs the grade $y = 2$, then for every j, k the grades y_j, y_k are either 0 or 1 and thus $|y_j - y_k| \leq 1$ also holds. Finally, note that if $y_j \neq 0$, then j output $m', y_j, \mathsf{Z}, \mathsf{G}$ after receiving a pair m', σ' such that $\mathsf{Verify}(\mathsf{pk}_{i^*}, m', \sigma') = 1$. As stated above, i^* only signs m, so $m' = m$.

Termination. All parties terminate after exactly 2 rounds.

No Living Undead. Parties only become ghosts or zombies after doing so in the undead weak multicast protocol, and thus are either send or receive omission respectively since that protocol has no living undead as well.

A.3 Undead Weak Consensus

Theorem 3. *The protocol described in Fig. 3 is a (t, s, r)-secure undead weak consensus protocol if $n > 2t + s + r$.*

Proof. **Validity.** Assume all parties that are alive in the beginning of the protocol have the same input x. Each alive i sends x, σ_i to all other parties. Each nonfaulty party receives at least $t + 1$ such messages from all nonfaulty parties, and no non-Byzantine party receives more than t messages x', σ' such that $x' \neq x$. Every nonfaulty party then calls the undead graded multicast protocol with the set of values it received. From the Validity property of the protocol every non-Byzantine party that isn't a zombie at this time receives the S_j set sent by every nonfaulty party. Therefore, every party i that isn't a zombie by that time received at least $t+1$ certificates S_j for the input x sent by the nonfaulty parties, and no set contains more than t signatures on $1-v$. Therefore every party i that isn't a zombie at that time outputs $x, \mathsf{Z}_i, \mathsf{G}_i$.

Consistency. Assume some non-Byzantine party outputs $v, \mathsf{Z}, \mathsf{G}$ such that $v \neq \bot$. By construction, this means that this party is not a zombie at that time. Since it output $v \neq \bot$, there are $t+1$ parties j from which it received a certificate $S_{j,i}$ for v with $y_{j,i} = 2$. At least one of these parties is non-Byzantine, and let that party be j. From the Validity and Consistency properties of the undead graded multicast protocol, all parties that aren't zombies received the same set $S_{j,i}$ with either grade 1 or 2. This set is a certificate for v and thus every party that isn't a zombie won't output $1-v$. In addition, parties that do become zombies output $\bot$. Therefore, no non-Byzantine party outputs $1-v$ from the protocol.

Termination. All parties complete the protocol after exactly 2 rounds.

No Living Undead. Parties only become zombies or ghosts in the protocol after seeing they do so in the undead graded multicast protocol. The protocol has no living undead, so parties only do so if they are indeed send or receive faulty respectively.

A.4 Undead Consensus

Theorem 4. *The protocol described in Fig. 4 is a (t,s,r)-secure undead consensus protocol if $n > 2t+s+r$.*

Proof. **Validity.** Assume all parties that are alive in the beginning of the protocol have the same input v. All parties set $v_i = v$ and start the first iteration of the protocol. We will now prove inductively that every party i that hasn't terminated has $v_i = u_i = v$ in all iterations of the protocol. From the Validity property of the undead weak consensus protocol, every party that isn't a zombie by the end of the protocol outputs $u_i = v$ and sets $v_i = v$. Any party that is a zombie by the end of the protocol outputs $\bot, \mathsf{Z}, \mathsf{G}$. Following that, all parties that are not zombies start the next iteration with $v_i = v$. Note that parties only send messages of the form x, σ with $x = v_i$ and thus at most t parties sign any value other than v in the protocol. This means that at all times, there are signatures from at most t different parties in D_{1-v} and thus no nonfaulty party outputs $1-v$. In order to complete the proof, it is required to show that all parties eventually terminate, which will be shown in the Termination property of the protocol.

Consistency. Assume some nonfaulty party output some value. It does so after seeing that there are signatures from at least $t+1$ parties in either D_0 or D_1. Let i be the first non-Byzantine party to send a v, σ message for any value v. If it does so, it is alive at that time, and both output $u_i = v$ from the undead weak consensus protocol and saw that $b = v$ in the same iteration. In that case, every party j that is alive at that point in the protocol outputs either $u_j = v$ or $u_j = \bot$ because of the Consistency property of the undead weak consensus protocol and outputs $b = v$ from $\mathsf{CoinFlip}$ in that iteration. Then, every alive

j either updates v_j to u_j if $u_j \neq \perp$ (and thus $u_j = v_j = v$), or updates v_j to $b = v$. Following the same arguments as the ones in the Validity property, no non-Byzantine party ever sends a signature on $1 - v$, and thus every party that completes the protocol and isn't a zombie outputs v.

Termination. First note that if some nonfaulty party completes the protocol after outputting v, it sends v, D_v to all parties. Every party that isn't receive faulty receives that message and terminates in a finite number of rounds, and every party that is receive faulty will become a zombie after all other parties stop participating and don't send it messages, causing it to become a zombie if it hasn't terminated earlier. This means that it is enough to show that some nonfaulty party completes the protocol in a finite amount of time. We will show that this even takes place with probability 1 and that the expected number of rounds until this happens is constant.

Observe a single iteration of the protocol in which no nonfaulty party terminated, and assume without loss of generality that all parties that are alive at the end of the call to the undead weak consensus protocol either output v or $\perp$ for some $v \in \{0, 1\}$. With probability $\frac{1}{2}$, all parties output $b = v$ from the $\mathsf{CoinFlip}$ protocol as well in that iteration. Therefore, every party i that is alive in the next iteration has $v_i = v$. As shown in the proof of the Validity property, from this point on, all parties have $v_i = u_i = v$ in all subsequent iterations of the protocol. From the consistency property of the undead weak consensus protocol, all nonfaulty parties output v from the protocol in each iteration from this point on. In each one of those iterations, with probability $\frac{1}{2}$, every nonfaulty party also outputs $b = v$ from the $\mathsf{CoinFlip}$ protocol. At that point, each nonfaulty party sends a v, σ message to all other parties. Every nonfaulty party receives those messages, adds the a tuple (j, σ) to D_v for every nonfaulty j and outputs v in the end of the iteration. From the above discussion, all parties complete the protocol in a constant number of rounds after that.

Note that the number of iterations required for each of the above events is a geometric random variable with probability $\frac{1}{2}$ of succeeding in each iteration, and thus the expected number of rounds is bounded by 4. In addition, for any ℓ, the probability that it takes more than ℓ iterations for each of the events to take place is no greater than $\frac{1}{2^\ell}$, and thus the probability that some party does not terminate in at most 2ℓ iterations is no greater than $\frac{2}{2^\ell} = \frac{1}{2^{\ell-1}}$, which decreases exponentially as ℓ grows with a 0 probability of an infinite run.

No Living Undead. Parties only become zombies or ghosts in the protocol if they do so in the undead weak consensus and $\mathsf{CoinFlip}$ protocols. Both of these protocols have no living undead, so parties do so if they are receive or send faulty respectively. Note that if a party is alive and it completes the protocol after outputting v, it sends its D_v set to all parties and continues participating in another iteration of the loop. This means that any party that is alive at the time the first nonfaulty party completes the protocol either receives the message or is receive faulty. It then outputs $v, \mathsf{Z}, \mathsf{G}$ in the next iteration of the loop. Since all parties are still participating in the protocol, the undead weak consensus protocol has no living undead at that point in time. Any receive-omission party that hasn't terminated by that point in time will either become a zombie because it receives no messages or eventually output v. In addition, note that if parties become zombies, all parties are informed and act as if the zombies sent messages consistent with receiving no messages in all protocols, which is a possible behaviour of theirs in a regular run of the protocols, and thus the protocols still have no living undead.

B Original Protocol Definitions

As stated in Section 2.4, the definitions used in this paper are variations on well-known protocols, which also add the notion of undead parties. Below are the original definitions, which do not use undead parties. Note that in the undead versions of the protocols, parties also output the flags Z and G, which are not part of the output in the original setting. This means that the undead versions technically do not solve the original tasks, but can be easily adjusted to solve them by simply omitting these flags from the output. Note that in the below protocols we actually consider uniform versions of the protocols. In these versions, even omission faulty parties are required to output the correct values or $\perp$ if they can't (specifically, if they are receive faulty). This reflects the above constructions of the undead versions of the protocols, which also have these guarantees for faulty outputs. Weaker versions of such protocols also exist in which no restrictions are made on the output of faulty parties, but we do not define such protocols below.

Multicast. This notion is usually not defined separately, and is often just used as a simple instruction to send a message to all parties. One could define a multicast task in which a known sender has a message to send to all parties, and all parties output a message or $\perp$.

Definition 6. *Let Π be a protocol executed by parties $1, \ldots, n$, with a designated sender i^* starting with an input $m \neq \perp$. Every party outputs a value x such that x is either a possible message or $\perp$.*

- ***Validity.*** *Π is (t, s, r)-*valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is nonfaulty or receive faulty, every party that is neither Byzantine nor receive faulty outputs m. In addition, if i^* is send faulty, no non-Byzantine party outputs x such that $x \notin \{m, \perp\}$.*
- ***Termination.*** *Π is (t, s, r)-*terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

*If Π is (t, s, r)-*valid *and (t, s, r)-*terminating *we say that it is a (t, s, r)-secure multicast protocol.*

Graded Multicast. A graded multicast protocol also has a designated sender that attempts to send its message to all parties. In addition to outputting a message x, parties also output a grade y, indicating how confident they are that the the protocol succeeded.

Definition 7. *Let Π be a protocol executed by parties $1, \ldots, n$, with a designated sender i^* starting with an input $m \neq \perp$. Every party outputs a pair of values (x, y) such that x is either a possible message or $\perp$ and $y \in \{0, 1, 2\}$.*

- ***Validity.*** *Π is (t, s, r)-*valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is nonfaulty or receive faulty, every party that is neither Byzantine nor receive faulty outputs x, y such that $x = m, y = 2$. In addition, if i^* is send faulty, no non-Byzantine party outputs x, y such that $x \notin \{m, \perp\}$.*
- ***Consistency.*** *Π is (t, s, r)-*consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if i^* is non-Byzantine and j, k are neither Byzantine nor receive faulty, and j, k output x_j, y_j and x_k, y_k respectively, then $|y_j - y_k| \leq 1$. In addition, if j is non-Byzantine (but is possibly receive faulty) then either $x_j = \perp$ and $y_j = 0$ or $x_j = m$.*
- ***Termination.*** *Π is (t, s, r)-*terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

*If Π is (t, s, r)-*valid*, (t, s, r)-*consistent *and (t, s, r)-*terminating *we say that it is a (t, s, r)-*(secure) *graded multicast protocol.*

Weak Consensus In a weak consensus protocol, every party has a binary input, 0 or 1. The goal of the protocol is for there to be some value y such that every party either outputs y or $\perp$.

Definition 8. *Let Π be a protocol executed by parties $1, \ldots, n$, in which every party i starts with an input $v_i \neq \perp$. Every party outputs a value x such that x is either a possible input message or $\perp$.*

- ***Validity.*** *Π is (t, s, r)-*valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if all nonfaulty parties have the same input v, every party that is neither Byzantine nor receive faulty outputs v.*
- ***Consistency.*** *Π is (t, s, r)-*consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if two non-Byzantine parties output x and x' respectively, then either $x = x'$, $x = \perp$ or $x' = \perp$.*
- ***Termination.*** *Π is (t, s, r)-*terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

*If Π is (t, s, r)-*valid*, (t, s, r)-*consistent *and (t, s, r)-*terminating *we say that it is a (t, s, r)-*secure *weak consensus protocol.*

Consensus In a consensus protocol, all parties output the same value. In addition, if all parties have the same input, they must output that value.

Definition 9. *Let Π be a protocol executed by parties $1, \ldots, n$, in which every party i starts with an input $v_i \neq \perp$. Every party outputs a possible output value x or possibly $\perp$ if it is faulty.*

- ***Validity.*** *Π is* (t, s, r)-valid *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if all nonfaulty parties have the same input v, every party that is neither Byzantine nor receive faulty outputs v. In addition, receive faulty parties either output v or $\perp$.*
- ***Consistency.*** *Π is* (t, s, r)-consistent *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: if two parties non-Byzantine and they output x, x', then either $x = x'$ or one of them is receive faulty and it outputs $\perp$.*
- ***Termination.*** *Π is* (t, s, r)-terminating *if the following holds whenever at most t parties are Byzantine, s parties are send faulty and r parties are receive faulty: all parties complete the protocol and output a value.*

If Π is (t, s, r)-valid*,* (t, s, r)-consistent *and* (t, s, r)-terminating *we say that it is a* (t, s, r)-secure *consensus protocol.*