General Adversary Structures in Byzantine Agreement and Multi-Party Computation with Active and Omission Corruption *

Konstantinos Brazitikos[1] and Vassilis Zikas[2]

[1] University of Edinburgh, U.K.
K.Brazitikos@sms.ed.ac.uk
[2] Purdue University, U.S.A.
vzikas@cs.purdue.edu

Abstract. Typical results in multi-party computation (in short, MPC) capture faulty parties by assuming a threshold adversary corrupting parties actively and/or fail-corrupting. These corruption types are, however, inadequate for capturing correct parties that might suffer temporary network failures and/or localized faults—these are particularly relevant for MPC over large, global scale networks. Omission faults and general adversary structures have been proposed as more suitable alternatives. However, to date, there is no characterization of the feasibility landscape combining the above ramifications of fault types and patterns.

In this work we provide a tight characterization of feasibility of MPC in the presence of general adversaries—characterized by an adversary structure—that combine omission and active corruption. To this front we first provide a tight characterization of feasibility for Byzantine agreement (BA), a key tool in MPC protocols—this BA result can be of its own separate significance. Subsequently, we demonstrate that the common techniques employed in the threshold MPC literature to deal with omission corruptions do not work in the general adversary setting, not even for proving bounds that would appear straightforward, e.g, sufficiency of the well known Q^3 condition on omission-only general adversaries. Nevertheless we provide a new protocol that implements general adversary MPC under a surprisingly complex, yet tight as we prove, bound.

As a contribution of independent interest, our work puts forth, for the first time, a formal treatment of general-adversary MPC with (active and) omission corruptions in Canetti's universal composition framework.

Keywords: Secure Multiparty Computation · General Adversary · Simulation · Active · omission corruptions.

1 Introduction

Multi-party computation (MPC) enables n parties to securely compute a function on their joint input. To capture parties' misbehavior one typically considers a central adversary corrupting parties and using them to attack the protocol. The most common corruption type for such an adversary is *active* corruption—the adversary takes full control of a corrupted party. Security against such an active adversary offers strong guarantees, but allowing the adversary to take full control of corrupted parties is an overkill to capture more benign types of misbehavior or just faults. In fact, this typically yields restrictions both in the feasibility—e.g., tolerable number of corruptions—and in terms of efficiency. As a result, different types of corruption have been investigated to capture such benign faults scenarios.

In the opposite extreme of active corruption, *fail-corruption* (aka fail-crash corruption or fail-stop corruption) allows the adversary to make a party crash (irrevocably) at any point of the protocol he chooses—without having knowledge of the party's internal state.[3] Naturally, adversaries with fail-corruption allow for

* This work was supported by Input Output (iohk.io) through their funding of the Edinburgh Blockchain Technology Lab.

[3] In the distributed computing literature, omission-corrupted parties are often considered also semi-honest. This is suitable for classical distributed computing tasks, e.g., Byzantine agreement (see below), where input privacy is a lesser issue. However, since here we are interested in MPC, we will follow the cryptographic convention of considering it separate.

better feasibility and efficiency bounds than active adversaries, but this corruption type is often criticized as too benign. As an example, fail-corruption is too weak for capturing faults caused by temporary issues on the network of otherwise *correct*—i.e., protocol abiding—parties. This gave raise to the study of the so called *omission-corruption*, which allows the adversary to selectively drop incoming and/or outgoing messages of the corrupted party, but obliviously of the message contents or the party's internal state.

On a different dimension, the two standard ways to capture the adversary's corruption patterns are via *threshold* and *general* adversaries. A threshold adversary is specified by the maximum number (threshold) of possible corruptions. This model can, again, be considered overly pessimistic, and therefore restrictive, when one considers situations in which certain combinations of faulty parties are unlikely. The concept of a *general adversary (structure)* is the alternative, fine-grained way which better captures such a situation: Rather than the maximum number of corruptions, a general adversary structure $\mathcal{Z}$ enumerates all possible combinations of corrupted parties, therefore giving more flexibility in describing the adversary's capabilities.

Tight feasibility bounds have been established for both threshold and general adversaries in the context of active corruptions and fail-corruptions, and even their combination (see Section 1.1 below for an overview). However, to our knowledge, omission corruption has not been considered for general adversaries, neither in isolation nor in conjunction with active corruption. Furthermore, all work on omission corruptions or general adversaries uses the property-based security definition of MPC, as opposed to simulation based security which is not only more general but, as we discuss is needed for completing the proofs of these works that rely on composing smartly designed sub-protocols. In a nutshell, our work provides the first characterization of the feasibility landscape of Byzantine agreement (BA)—the core primitive in fault-tolerant distributed computation and standard building block of MPC—and of secure multi-party computation (MPC) for general adversaries that might corrupt some parties actively (i.e., force byzantine faults) and, simultaneously, omission corrupt other parties. Concretely, we prove a tight feasibility bound for both synchronous consensus and broadcast in the perfect (security) setting, i.e., information theoretic security with zero error probability. We then turn to the study of MPC in this model. As we show, the correctness and privacy requirement of MPC make existing arguments and techniques from the cryptographic literature inadequate for proving even what one would consider a simple and intuitive feasibility result. Notwithstanding, we provide a tight feasibility bound for MPC in this setting by developing a new protocol for (publicly) detectable point-to-point secure communication and proving a tight bound for this task. In fact, a look at the complexity in the associated (tight!) bound (see Eq. 6) serves as a perfect demonstration of the technical challenges associated with devising such a bound, protocol, and associated tightness proof.

Of independent interest is the fact that our results are proven secure in a simulation-based composable framework. This, to our knowledge, is a first both for general (mixed) adversary MPC and for MPC with omission corruptions.

1.1 Related Literature

In this section we discuss the related literature, where we focus on perfect (security) protocols, i.e., with zero error probability, which is also the type of protocols we develop here.

(Synchronous) Byzantine Agreement (BA). BA comes in two flavors: consensus and broadcast. In consensus, n parties, each with its own input, wish to agree on a joint output, so that pre-agreement is preserved. In broadcast, only one party, the sender, has input, and the goal is to distribute it in a consistent manner to all parties, so that consistency is achieved even if some of the parties are actively corrupted (cf. Section 2.6, 2.7). The seminal results by Lamport, Shostak, and Pease[PSL80,LSP82], showed that Consensus and Broadcast are feasible if and only if at most t parties are byzantine, where $t < n/3$. Follow up work has extended the above results to various models capturing different types of synchrony, alternative networks, and setup assumptions such as a public key infrastructure.

Multi-party computation (MPC). In MPC we have n parties from a set $\mathcal{P} = \{p_1, \ldots, p_n\}$, each with a private input x_i who wish to securely compute a function on their joint input, even in the presence of faulty parties. Faulty parties are captured by assuming a central adversary that corrupts parties and uses them to orchestrate a coordinated attack to break the protocol's security, where the two main security goals are *privacy*—corrupted parties should learn nothing beyond their prescribed inputs and output, and

correctness—the adversary should not be able to affect the output of the computation in any other way than choosing his own inputs independently of that of uncorrupted parties. The typical type of corruption is active. Actively corrupted parties are often referred to as *malicious* or *byzantine* and the set containing them is denoted as A.

MPC was introduced by Yao [Yao82] where feasibility of two-party computation was shown. The seminal works of Ben-Or, Goldwasser, and Wigderson [BGW88] gave the first feasibility results for perfect security (that is, information-theoretic with zero error probability) for a threshold adversary. In particular it was shown that $t < n/3$ is both necessary and sufficient for perfectly secure MPC in the synchronous malicious adversary model.

General adversary structures. General adversaries have also been studied for both BA and MPC. Here, for the case of perfect security, Hirt and Maurer [HM97,HM00] proved that a necessary and sufficient condition, if no setup[4] is assumed, for a general adversary structure—with active corruptions—to be tolerable is that the union of no three sets in the adversary structure $\mathcal{Z}$ covers the whole player set, a condition which is often referred to as the Q^3 condition:[5]

$$CP^{(A)}_{CONS}(\mathcal{P}, \mathcal{Z}) \iff Q^3_A(\mathcal{P}, \mathcal{Z}) \iff \forall A_i, A_j, A_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \neq \mathcal{P}. \quad (1)$$

The above tight condition holds for perfectly secure BA (both consensus and broadcast) and MPC. This was later extended to the mixed setting adding fail-corruption faults in [AFM99] and a combination of fail-corruption and passive corruption [BFH+08]. (We refer to [Zik10] for a comprehensive survey of the relevant literature).

Omission Faults. The first variant of omissions was introduced to the distributed literature by Hadzilacos [Had85], where the notion of send-only omissions was introduced. There, and it was proven that $t < n$ (send-)omission faults are necessary and sufficient for BA. Full (send and receive) omission faults were proposed by Perry and Toueg [PT86], who affirmed the $t < n$ bound for that more general model. A long line of follow-ups investigated the problem. As it can be seen in [AP99], the recovery of crashed components is often considered a built-in feature of the distributed replication systems, meaning that crash failures are treated in essence as omissions, making omissions appear frequently in the literature.

Importantly, in [Had85,PT86], a weaker variant of BA with omissions was considered, where the consistency guarantee was limited to the output of the non-faulty (i.e., uncorrupted/honest) players—meaning that omission-corrupted players were treated as malicious, and were not given any output guarantees. The case where the output of both honest and omission-corrupted players should be guaranteed (whenever possible) was treated by Raynal and Parvedy in [Ray02,PR03] where it was proved that the tight bound on omissions with this requirement becomes $t < n/2$. We note in passing that this latter, more natural and challenging way is also how we treat omissions in this work.

In the cryptographic literature omission faults (also referred to as *omission corruption* and denoted by Ω) were first studied by Koo[Koo06] who proved that for a (mixed-corruption) adversary who can corrupt up to t_a parties actively and omission corrupt up to t_ω parties, $3t_a + 2t_\omega < n$ is sufficient for Consensus and $4t_a + 3t_\omega < n$ is sufficient for MPC. Follow-up work by Hauser, Maurer, and Zikas [ZHM09] provided the first tight bounds proving that $3t_a + 2t_\omega < n$ is both necessary and sufficient for BA and MPC in the perfect security (synchronous) setting. The results were extended in [Zik10] by adding fail corruption.

More recently, Eldefrawy, Loss, and Terner [ELT22] and investigated computational security for the case where send and receive omission faults have different thresholds t_s and t_r respectively. This case was also treated in [ZHM09] but for perfect security only. As demonstrated in [ELT22] the shift to computational security carries unexpected complications, which is yet another indication of the challenges associated with omission-corruption. More concretely, [ELT22] proved that in this setting $t_s + t_r + 2t_b < n$ is sufficient for MPC—where t_b is the threshold on byzantine parties. However they were not able to prove the bound tight for the general case; instead they proved tightness for a weaker adversary that performs what they termed "spotty" send-corruptions: messages from a send-(omission-)corrupted player in any round are either all delivered or none of them is.

[4] Note that "no setup" implies that we cannot use cryptographic tools such as digital signatures.

[5] Here we denote the classical Q^3 condition as Q^3_A to explicitly state that it only applies to active corruptions.

1.2 The Model

We consider n parties from a party set $\mathcal{P} = \{p_1, \ldots, p_n\}$. The parties can communicate via a complete network of bilateral point-to-point secure (i.e., authenticated and private) channels [BGW88]. (We note in passing that our BA protocol does not need privacy and can just rely on standard authenticated channels; however, privacy is necessary for perfectly secure MPC results). We assume synchronous communication as in [LSP82,BGW88,CCD88], i.e., all our protocols advance in rounds; every party is aware of the current round and can send messages to all other parties, where messages sent in any round are delivered to their intended recipients by the beginning of the following round.

For simplicity in the exposition, for protocols that build on top of broadcast we assume that each of their round is a broadcast round (i.e., a round where all parties can broadcast a message). This does not affect composition of the total counting of rounds as our broadcast protocol is deterministic and therefore we do not run into the known issues of probabilistic termination [CCGZ16]. Furthermore, to make the protocols description simpler we will assume that each sub-protocol has a dedicated *output round* where the parties do not send any messages to each other, but use messages they have received to compute their (sub-)protocol's output(s). This does add a constant overhead on sequentially composing protocols, but makes for a much cleaner abstraction and does not affect the nature of our results which is targeted to feasibility. In fact, one can easily get rid of this overhead by starting a next sub-protocol already during that output round of the previous one.

Simulation-based (composable) security. We prove our protocols secure using the synchronous adaption of Canetti's UC framework [Can01] put forth by Katz *et al.* [KMTZ13]. We assume the reader has some familiarity with UC, but we make our best effort to keep the technicalities of the framework insulated from the protocol design and functionality description. In the following we discuss the above synchrony framework and how it is utilized here.

In a nutshell, [KMTZ13] proposed a methodology for the design/embedding of synchronous protocols within the (by-default asynchronous) UC framework. In this adaptation, protocols can be designed in a synchronous manner, and [KMTZ13] defines how they can be executed assuming access to a clock functionality, which ensures that (1) all parties get a chance to speak in each round, (2) parties can become aware when the clock round switches. Proving security in such a framework means that the functionalities need to also become round aware; this is taken care of in[KMTZ13] by adding to the functionality dummy rounds which advance once every party has had a chance to ping the functionality in that round. This allows the environment to advance the ideal experiment if it wishes to, similar to what it can do in the real world. To keep the description cleaner, we abstract away this pinging of functionalities as dummy ("do-nothing") rounds in the functionalities we define, and explicitly make the functionality aware of the underlying (broadcast) round.

To make the two-fold contribution of our work (protocol/proofs level vs. model/UC-treatment level) clearer and isolate the techniques used in each of the two contribution types, we use the following methodology in proving our feasibility results: First we state and prove in separate claims key properties that our (sub-)protocols achieve; this is useful for understanding the protocol ideas that go into the construction and how these are used in the security proof. Then, we use these properties in the simulation proof to obtain our end result.

Adversary. We consider a mix of active corruption and omission-corruption characterized by general adversary structures. Concretely, the possible combinations of corruptions are described by a mixed (active/omission) general adversary structure. Such an adversary is a collection $\mathcal{Z}$ of tuples of the type $(A_i, \Omega_i) \in \mathcal{P}^2$, often referred to as *classes*. Intuitively, $\mathcal{Z}$ is intended to capture all possible scenarios of corrupted parties. In particular, a tuple/class $(A_i, \Omega_i) \in \mathcal{Z}$, displays the scenario where all parties in A_i are actively corrupted and all parties in Ω_i are omission-corrupted. We will be using the terminology: *" the adversary corrupts (class)* $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$*"* to refer to the above scenario and we denote it by using a * symbol at the exponent. This means for example that the sets $A_i^\star$ and $\Omega_i^\star$ denote the sets of actively corrupted and omission-corrupted players, respectively. Similarly, we refer to an adversary who might corrupt any of the sets in $\mathcal{Z}$ as a (general) $\mathcal{Z}$-adversary. The set of uncorrupted/honest players will be denoted by $\mathcal{H}$. Note that the class $Z^\star$ is not known to the players and appears only in our security analysis. Furthermore an omission or actively corrupted party might be allowed to send or receive all its messages, in which case he is indistinguishable

from an uncorrupted party. We refer to such a party as *correct* at a certain point in time if it was allowed to behave this way (correctly) up until this certain point in time. Essentially, an omission-corrupted party stops being correct the moment its first message is blocked. Finally, some of our protocol executions allow omission-corrupted parties to realize that they are corrupted; when this detection occurs, the party understands that it is in the discretion of the adversary whether or not it will be allowed to contributed inputs or receive outputs in the protocol. Therefore, in such cases the parties step out of the computation and inform all their peers about this decision; borrowing the terminology of [ZHM09] we will then say that this party becomes a *zombie*, in contrast to the rest of non-actively-corrupted parties that are considered *alive*.

We will make the following standard conventions on the adversary structure $\mathcal{Z}$: (1) For any $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$, for every $A' \subseteq A_i$ and $\Omega' \subseteq \Omega_i$: $Z' = (A', \Omega') \in \mathcal{Z}$. This captures the intuitive fact that if a set of parties might jointly fail in a certain way, then any subset of them failing is also a possible corruption scenario. This convention allows us to describe $\mathcal{Z}$ by enumerating only its maximal elements. (2) For any $Z_i = (A_i, \Omega_i) \in \mathcal{Z}$ we will assume that $A_i \subseteq \Omega_i$; this is simply capturing the fact that active corruption is strictly more severe (as a misbehavior strategy) than omission and can, behave as such.

Finally, we prove our statements here with respect to *static* adversaries, i.e., the set of corrupted parties (and hence the set of possible corruptions) is decided at the beginning of the protocol and cannot depend on the exchanged messages. We note that all properties we prove here will directly hold to the adaptive security setting without changing the respective bounds [CDD+01,ACS22]. However, the simulation-based treatment of adaptive security under parallel composition of, e.g., BA primitives is known to have several thorny issues which are beyond the scope of this submission [HZ10,CGZ23].

1.3 Overview of our Results and Organization of the Paper

Byzantine Agreement. As our first contribution towards our MPC feasibility we prove that the following condition on the adversary structure is necessary and sufficient for perfect synchronous BA, both broadcast and consensus:

$$C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \tag{2}$$

Without loss of generality we provide protocols for binary consensus and broadcast, i.e. the inputs and outputs of the protocol are from the field $\mathbb{F} = \{0, 1\}$. This is sufficient for arbitrary valued BA, as we can represent the inputs as bit-strings of appropriate (fixed) length and then we can invoke the bit-Consensus protocol for each of those bits.

Our feasibility result is proven in two stages. First, in Section 2.2 we show how to tackle one of the core challenges of omission-corruption, namely detection of dropped messages. In particular, the biggest thorn with omissions is that a party p_j who does not receive a message it expects does not know whether this happened because the sender or itself (p_j) is omission-corrupted. To tackle the above issue, we devise a simple protocol, called *FixReceive*, which allows the receiver to take this decision. We prove (see Claim 1) that the decision will always be correct as long as the following condition[6] is satisfied

$$C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \iff \forall Z_i, Z_j \in \mathcal{Z} : \Omega_i \cup \Omega_j \neq \mathcal{P}, \tag{3}$$

which is also proven necessary for the above task in Lemma 2.2. One can view *FixReceive* as a way to lift the underlying communication network from a plain one to one with detection. When this detection is successful and a player discovers that he suffers from omissions, he steps down—becomes a zombie—for the rest of the protocol and sends a special message to let others know.

The underlying idea of *FixReceive* is simple, and similar to the corresponding protocol from [ZHM09]: the sender sends to all parties, who relay to the receiver; then the receiver tries to "fit" the received messages into the corruption pattern. However, in the threshold case, this "fitting" is rather straightforward. This is in contrast to the general-adversary case, where the right condition (and proof) is more involved. Yet, the above simplicity of *FixReceive* stems from the fact that it makes the transmitted message public to

[6] Due to our assumption from earlier, the condition can also be written as $A_i \cup \Omega_j \neq \mathcal{P}$.

the adversary. This makes it suitable for BA but not for MPC where we need detection on top of private communication. Looking ahead, this combination turns out to be particularly challenging and the private version of *FixReceive* (which we will call detectable secure message transmission) will be one of the core contributions of our paper.

Having added *FixReceive* to our arsenal, we can now use this to improve the communication properties that are disrupted by omission corruptions. (This can be seen as "lifting" the underlying communication network by adding (partial) corruption awareness/detection.)

In particular, having improved the detection ability of communicating parties as above, we proceed to our BA construction. For this, we use the phase-king approach of Berman, Garay, and Perry [BGP89]–which was previously adapted to general adversary structures (with fail-corruption instead of omissions) by Altmann, Fitzi, and Maurer [AFM99]. Concretely, we gradually build protocols with stronger guarantees, from *Weak Consensus* (Section 2.3), to *Graded Consensus* (Section 2.4), to *King Consensus* (Section 2.5), and then iterate through different parties as kings to achieve the consistency and validity conditions of consensus (see Theorem 2).

At this point, it is worth discussing the main challenge in shifting from a combination of active corruptions and fail-corruptions (for which we know tight bound both for BA [AFM99] and for MPC [BFH$^+$08]) to active and omission corruptions for which nothing is known in the general-adversary setting: The main issue lies in the ability of an omission-corrupting adversary to create confusion by selectively dropping messages to some and not other parties and in some specific rounds. In particular, a standard method to limit the effect of fail-corruptions is to embed a heart-bit after each step (or in selective protocol rounds) that allows parties to detect whether or not some party has already crashed. This approach does not work with omission corruptions, as a party might drop messages during the protocol round, but send all messages in the heart-bit procedure as if nothing happened. Thus one needs to come up with ways to counter the ability of the adversary to create such confusions. The challenge of our above protocol design is to come up with protocols that either allow for public detection of an omission-corrupted party not sending messages, or make the party aware that it is omission-corrupted—in the latter case, this party can put itself in a crashed position (a possibility which the adversary would anyway have by blocking all communication to/from that party) to allow the other parties to complete the protocol.

Having derived a consensus protocol as above, we then turn to broadcast. Interestingly, the standard reduction of broadcast to consensus—i.e. have the sender send his input to everyone and run consensus on the received values—does not work here. The reason is that a send-omission corrupted sender p_s might fail to send his input to some but not all non-actively corrupted parties, in which case consensus might end up flipping his input, which violates our requirement on the output with a non-actively corrupted sender.

We fix this by using an additional round of consensus: To guarantee that an omission-corrupted p_s never broadcasts a wrong value (but he may broadcast $\perp$ in case he is incorrect) we extend the above generic protocol as follows: after running consensus on the received bit, we have p_s send a confirmation bit to every player, i.e. a bit $b = 1$ with the meaning that p_s agrees with his output of the consensus or $b = 0$ otherwise. The players then invoke consensus on the received bit to make sure that they have a consistent view on the confirmation-bit and based on that they accept the output of the generic broadcast protocol only if $b = 1$. In the opposite case, they output $\perp$. This ensures that if they output anything, it will be the correct bit.

Finally, we prove the tightness of $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ for BA by means of a delicate player simulation argument (see Lemma 5).

Multi-Party Computation. Having proven a tight characterization of BA in our model, we turn to multi-party computation (MPC). Here we first observe that even the seemingly straightforward bounds do not apply. In particular, it is known that in the case of active-only general adversary structure, MPC is feasible if and only if the $Q_A^3(\mathcal{P}, \mathcal{Z})$ condition (Eq. 1) holds [HM97]. One might be tempted to assume that since active corruption is more severe than omission-corruption, the natural adaptation of the above condition to the omission-only setting, i.e. the condition Q_Ω^3

$$CP_{CONS}^{(\Omega)}(\mathcal{P}, \mathcal{Z}) \iff \forall \Omega_i, \Omega_j, \Omega_k \in \mathcal{Z} : \Omega_i \cup \Omega_j \cup \Omega_k \neq \mathcal{P}, \tag{4}$$

would be at least sufficient for MPC. This however is not necessarily the case, as MPC protocols for active corruptions entirely give up the inputs and outputs of actively corrupted parties, something which we cannot do for omission corruption. Indeed, in Section 3.1 we give an example of such a structure that suffers from this issue.

In fact, as we argue, common techniques used in the threshold MPC literature to recover from corruptions, such as *player elimination* [BHR07], cannot be applied here either. A standard example of player elimination is used in the case of MPC with (threshold) byzantine corruptions with $t < n/3$. The idea is that if some p_i blames another p_j, then, as long as both p_i and p_j have had the chance to share their inputs, we can simply eliminate both of them and continue the computation with the remaining parties—and send p_i and p_j their outputs at the end; the $t < n/3$ condition will then ensure that in the $n' = n - 2$ remaining parties set, the number t' of maximum active corruptions will still satisfy $t' < n'/3$. In fact, this technique was used in [ZHM09] to prove the first, and only to date, tight condition on MPC with active and omission corruption. However, as we show, player elimination is inapplicable in our general-adversary active/omission setting. In particular, we show that natural candidates for feasibility bounds conditions are not preserved by player elimination (see 3.1). This situation calls for new protocols/techniques beyond what is used in the threshold or previous general adversary literature.

To overcome this, we devise a novel protocol that aims at facilitating detectable (i.e., which might abort with the identity of a corrupted party) perfectly secure (private and authenticated) message transmission introduced in [DDWY93] (in short, DetSMT) between any two parties. Looking ahead, this will allow to neutralize the effect of omissions in MPC.
The challenge in devising and proving security of the new detectable SMT primitive is evident by the new associated condition $C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$, which in combination with $C^{(A,\Omega)}_{BA}(\mathcal{P}, \mathcal{Z})$ gives us the condition that is proven to be tight for DetSMT with sender p_s and receiver p_r. The $C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ states that for any three $Z_i, Z_j, Z_k \in \mathcal{Z}$:

$$\begin{aligned} &\text{if } (p_s \in \Omega_i \cap \Omega_j \wedge p_r \in \Omega_k) \text{ OR } (p_r \in \Omega_i \cap \Omega_j \wedge p_s \in \Omega_k) \\ &\text{then } A_i \cup A_j \cup \Omega_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \end{aligned} \tag{5}$$

The sufficiency of the above condition for detectable SMT and its necessity for (detectable) SMT (hence also for MPC) are proven in Section 3.2 (Lemmas 6 and 7, respectively).

Finally, we put everything together to prove our last main theorem (Theorem 4) of MPC feasibility under the same combination of conditions (where the $C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ needs to hold true for all pairs $p_s, p_r \in \mathcal{P}$). More concretely, we prove that perfectly secure MPC against a general adversary with mixed active and omission corruptions is feasible if and only if the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z})$ holds, where

$$C^{(A,\Omega)}_{MPC}(\mathcal{P}, \mathcal{Z}) \iff C^{(A,\Omega)}_{BA}(\mathcal{P}, \mathcal{Z}) \ \wedge \ \forall p_s, p_r \in \mathcal{P} : \ C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z}, p_s, p_r). \tag{6}$$

The necessity of the above condition follows from the fact that both SMT and broadcast are special cases of MPC. For the sufficiency we use the following idea: We modify the general adversary protocol from [BFH+08] by first projecting it to the active-corruption-only case (recall that [BFH+08] works for a mixed active/passive/fail adversary) and then doing the following:

- All point-to-point communication between any two parties p_i and p_j is done by the above detectable SMT.
- All broadcasts are implemented by our detectable broadcast.
- All sub-protocols in [BFH+08] for computing individual circuit gates (input, addition, multiplication, and output gates) are turned to detectable counterparts, i.e., they might abort and make the identity of a corrupted party public.
- Importantly, instead of computing the actual circuit, our MPC computes a verifiable secret sharing of the circuit's output—we prove that such a robustly reconstructible sharing is feasible under our conditions. The reason for this is that before its last reconstruction round, the MPC from [BFH+08] leaks no information to the adversary. By switching the computation's output to a secret sharing instead of

actual circuit value, we ensure that no matter if or when the protocol aborts, it will leak no information on any of the non-actively corrupted player's inputs.

The above construction gives a detectable MPC protocol which either computes a verifiable secret sharing of the output of the intended circuit, or it aborts without leaking any information to the adversary while exposing a corrupted party. Such a protocol can be bootstrapped to a fully secure MPC (with guaranteed output delivery) by standard techniques: Whenever it aborts, remove the detected (corrupted) party from the player set and re-start the computation—this can be repeated at most n times as each abort exposes a new corruption. Once the protocol succeeds, use the reconstruction protocol of the verifiable secret sharing to publicly reconstruct the outputs. We note in passing that the above only computes MPC with a public output, but it can be tuned to allow for private outputs using standard techniques: every party inputs in addition to its actual output a random key which is used to blind—by one-time-pad encryption—the announced public output so that only this party can recover the plaintext [LP09].

UC Treatment. Last but not least, as discussed above, all our proofs are in the (synchronous) UC framework, which we view as a contribution in its own sense. Indeed, to our knowledge, this is the first time that a general adversary protocol is proven secure in such a composable manner. In particular, existing MPC protocols for general adversaries [HM97,BFH$^+$08,ZHM09] also follow a modular design approach—i.e., they design sub-protocols for each type of MPC gate (input/sharing, addition, multiplication, output/reconstruction)—and prove the security of each underlying sub-protocols separately in a property-based manner, i.e., prove the correctness and privacy of each of these sub-protocols. They then argue that these sub-protocols can be combined in the main MPC protocol. Although we believe this last statement to be true, an actual proof would require a composition proof (which is generally problematic with property-based definitions), or, alternatively a composable treatment of the whole construction, an approach which we take for the first time in this work. In fact, to our knowledge even without considering general adversaries, no work has considered (active and) omission corruptions in UC. As it is evident by our functionalities, embedding omission corruptions in UC requires new design choices for the relevant functionalities.

Due to limited space, the detailed proofs of many of our statements are moved to the supplementary material section which is referred to as needed throughout the text. We do, however, for sake of self-containment, include intuition of the proved statements—especially the more complicated ones—in the body.

2 Byzantine Agreement with Active and Omission Corruption

2.1 Security Conditions

In this section we present our first major result, a tight BA condition. Our results cover the case of mixed active and omission-corruption under perfect security (i.e., zero error probability).

Theorem 1. *In the model with both active and omission-corruption if no setup is assumed a set $\mathcal{P}$ of players can perfectly $\mathcal{Z}$-securely realize Consensus or Broadcast if and only if the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ holds where,*

$$C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \tag{7}$$

The proof of the theorem is in 2.7 and the condition is proven to be both sufficient and necessary for both flavors of BA, i.e. Consensus and Broadcast. The theorem follows after all the necessary tools are created (namely the primitives FixReceive, Weak, Graded and King Consensus).

2.2 Detection of Omission-Failures

Functionality $\mathcal{F}_{FR}(\mathcal{P}, \mathcal{Z}, p_i, p_j, x)$

The sender $p_i \in \mathcal{P}$ has input x and the intended receiver is p_j. The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Initially, set the output value m_{out} to $\perp$.
- In round $\rho = 1$:
  - Upon receiving a message (input, sid, x) from the sender p_i (or the adversary, if the sender is actively corrupted), set $m_{out} = x$ and send (leakage, sid, $p_i, l(x) = x$) to the adversary.
- In round $\rho = 2$:
  - Upon receiving (adv-omit, sid, $\perp$) from the adversary, if $p_i \in \Omega^\star$, set $m_{out} = \perp$.
  - Upon receiving (inform omission, p_j) from the adversary, if $p_j \in \Omega^\star$, set $m_{out} = \perp$ and output (omission, p_j) to p_j.
  - Otherwise, discard the message.
- In round $\rho = 3$:
  - Upon receiving (fetch-output, sid) from p_j, send (output, sid, m_{out}) to p_j and (fetch-output, sid, p_j) to the adversary.

In this section we start by tackling the main problem of omission-corruption, namely the fact that detecting such a corruption is not trivial. Indeed, when a player p_j does not receive a message she was expecting (because the channels are synchronous), and receives the default value $\perp$ she cannot be certain if the sender p_i is *actively corrupted* and did not send a message, if the sender is *omission-corrupted* and his message was blocked or if p_j *herself is omission-corrupted* and was not able to receive the message (or a combination of the above).
Our first goal is to implement a functionality $\mathcal{F}_{FR}$ in order to reinforce our communication network and render it able to detect omission-failures. Effectively, this functionality guarantees that either the adversary lets the message of the sender p_i reach its recipient p_j or it becomes known to everyone (first to p_j herself and then she will make it public) that p_j is omission-corrupted, forcing her to become a zombie.
If p_j becomes a zombie via the *FixReceive* protocol, she stops participating in any upcoming computations. Also, she notifies all other players about that by sending a special message (she can send it at every round to make sure that everyone receives it) saying that she "is out". Those properties are captured by the functionality $\mathcal{F}_{FR}$ fully described above. For our functionalities we follow the template of [CCGZ16] for canonical synchronous functionalities. The functionality proceeds in this way. Initially $\mathcal{F}_{FR}$ sets the output value equal to $\perp$ and then waits for input from p_i. This input is made known to the adversary through the leakage function $l(x)$. On the second round the adversary has the following choices. i) If $p_i \in \Omega^\star$ (the sender is omission-corrupted), the adversary can drop the input message and turn it to $\perp$ or let it be recorded as normal. ii) If $p_j \in \Omega^\star$ the adversary can either inform p_j of his omission status or let her receive the recorded message m_{out}.
As such, we can see that if p_j remained alive then she outputs a value m_{out}, which will be either p_i's input or $\perp$. Additionally, if p_i is correct we are granted that it is the former case.

Our protocol which realizes this functionality does the following in more detail. When p_i wants to send a message x to p_j, he sends x to all $p_k \in \mathcal{P}$ to leverage all parties. Then, every p_k who received the message forwards it to p_j. If p_k did not receive a message (he denotes that by the symbol $\perp$) he sends a special message "$\mathsf{n/v}$" $\notin \mathbb{F}$ to p_j, to let her know that *no value* was received.
After that, p_j should have received a message from all $p_k \in \mathcal{P}$. If from some player she did not, she denotes that by the default character $\perp$. At that point, if there is no way according to the adversary structure $\mathcal{Z}$ that the $\perp$ symbols she received were sent by players that could be omission-corrupted *or* actively corrupted she becomes a zombie. In other words, if there is some player who sent $\perp$ but could never be corrupted, then it is clear for p_j that she has a problem in receiving messages.
In the opposite case, if there exists a value x' which was sent to p_j by people that could not be actively corrupted, it would mean that this value cannot be an erroneous one being pushed by the adversary. As such, p_j can be certain that this is the message that p_i sent, and she outputs this value x'.
Otherwise, in the case where no such value exists, meaning that p_i was not consistent with the messages he sent or he was blocked, p_j outputs $\perp$ to indicate that p_i is not correct. The protocol *FixReceive* $(\mathcal{P}, \mathcal{Z}, p_i, p_j, x)$

starts with the sender sending x to everyone. In the second round all parties forward their messages to the receiver, who tries to figure out what the message is. A notation that we will be using often is $P_j^{(\perp)}$ to denote the set of players that sent to p_j the value $\perp$ and similarly $P_j^{(x)}, P_j^{(\text{"}\mathsf{n/v}\text{"})}$. The procedure that the protocol follows is along these lines.

1. For each $p_k \in \mathcal{P}$, p_i sends x to p_k. The value received is denoted by p_k as x_k ($x_k = \perp$ if no value was received).
2. Each $p_k \in \mathcal{P}$ sends p_j a message x'_k, where $x'_k = x_k$ if $x_k \neq \perp$ and $x'_k =$ "$\mathsf{n/v}$" otherwise. The value received from p_k is denoted by p_j as $x_k^{(j)}$ ($x_k^{(j)} = \perp$ if no value was received).
3. Output: The receiver performs the following checks to determine the correct value to output, based on the classes of the adversary structure that justify the received messages.
   (a) If there exists no $Z = (A, \Omega) \in \mathcal{Z} : P_j^{(\perp)} \subseteq (A \cup \Omega)$, where $P_j^{(\perp)} := \{p_k : x_k^{(j)} = \perp\}$ then p_j becomes a zombie and outputs ($\mathsf{omission}$, p_j).
   (b) If there exists no $Z = (A, \Omega) \in \mathcal{Z} : P_j^{(\text{"}\mathsf{n/v}\text{"})} \subseteq (A \cup \Omega)$, then p_j outputs $\perp$ (the sender is omission-corrupted).
   (c) Otherwise, p_j considers all justifying (for any x) classes [7] Z_m.
   (d) If there are no justifying classes (for either x or $\overline{x}$) with $p_j \notin \Omega_m$ then p_j becomes a zombie and outputs ($\mathsf{omission}$, p_j).
   (e) If there exists unique x' and $Z_m : A_m \supseteq P_j^{(\overline{x})}$ and $A_m \not\supseteq P_j^{(x)}$, for $\overline{x} \neq x'$ then p_j outputs x'.
   (f) Otherwise p_j outputs $\perp$.

Lemma 1. *If the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 3) holds, the protocol* FixReceive *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{FR}$.*

Proof. We are going to give a simulation-based proof, meaning that we are going to devise a simulator and prove that it creates an ideal world execution that is indistinguishable from a real world execution of our protocol. To this direction we first prove the following useful lemma about the input/output properties of the protocol.

Claim 1. *If the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 3) holds, the protocol* FixReceive *has the following properties: If p_j is alive at the end of the protocol then p_j outputs a value x', where $x' \in \{x, \perp\}$, unless $p_i \in A^\star$, and $x' = x$ if p_i is correct until the end of the protocol. Moreover, p_j might become a zombie only if p_j is omission-corrupted.*

Proof. According to the protocol, p_j becomes a zombie when for every justifying $Z = (A, \Omega) \in \mathcal{Z}$ we have that $P_j^{(\perp)} \not\subseteq A \cup \Omega$ (or when there are no justifying classes with p_j not being omission-corrupted) and in no other case. This would mean that there is no class in the adversary structure that would explain the $\perp$ that p_j receives, unless she is omission-corrupted and the adversary is blocking messages that are addressed to her. As a result, we can be certain that $p_j \in \Omega^\star$, indeed.

Now, if p_j is not actively corrupted and is alive at the end of the protocol, we will show that she will output a value $x' \in \{x, \perp\}$. Specifically, according to the claim, if $p_j \in \mathcal{P} \setminus A^\star$ and the sender p_i is correct until the end of the protocol, p_j will output $x' = x$.

Let us assume, towards contradiction that the choice of x' is not unique and that there is another candidate value $\overline{x} \neq x$. From the protocol for the value x we have that there exists[8] a justifying for x class Z_ℓ with

$$(\Omega_\ell \cup A_\ell \supseteq P_j^{(\perp)} \text{ or } p_j \in \Omega_\ell) \text{ such that } A_\ell \supseteq P_j^{(\overline{x})} \text{ for all } \overline{x} \neq x. \tag{8}$$

[7] A class $Z_m = (A_m, \Omega_m)$ is justifying for x if ($\Omega_m \supseteq P_j^{(\perp)}$ or $p_j \in \Omega_m$) AND ($\Omega_m \cup \supseteq P_j^{(\text{"}\mathsf{n/v}\text{"})}$ or $p_i \in \Omega_m$) AND $A_m \supseteq P_j^{(\overline{x})}$ AND $A_m \not\supseteq P_j^{(x)}$, for all $\overline{x} \neq x$.

[8] The actual class $Z^\star$ is always a justifying class. Also, since p_i is correct, the correct value x will reach all correct players in $\mathcal{P}$ and then p_j. So the above class always exists.

Now, if for all justifying classes Z_ℓ it was true that $p_j \in \Omega_\ell$ then p_j would become a zombie. But since p_j stayed alive, there exists $Z_m : p_j \notin \Omega_m$, hence $\Omega_m \cup A_m \supseteq P_j^{(\perp)}$ and $A_m \supseteq P_j^{(\overline{x})}$. Additionally, if we assume that p_i is correct we get that the "$\mathsf{n/v}$" values must be coming from players that are indeed at least omission-corrupted. Thus, $\Omega_m \cup A_m \supseteq P_j^{(\text{"}\mathsf{n/v}\text{"})}$.
Similarly, for $\overline{x}$ we have that there exists Z_q such that

$$(\Omega_q \cup A_q \supseteq P_j^{(\perp)} \text{ or } p_j \in \Omega_q) \text{ such that } A_q \supseteq P_j^{(x)}, \text{ since } x \neq \overline{x}. \tag{9}$$

Now, according to the values that p_j received we can write the whole player set as

$$\begin{array}{rlll} \mathcal{P} = P_j^{(x)} & \cup P_j^{(\overline{x})} & \cup P_j^{(\perp)} & \cup P_j^{(\text{"}\mathsf{n/v}\text{"})} \\ \subseteq A_q & \cup A_m & \cup \Omega_m, & \end{array} \tag{10}$$

which is a contradiction to the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ because it states that $A_q \cup \Omega_m \not\supseteq \mathcal{P}$ for any two classes q, m. As $A_m \subseteq \Omega_m$ gives us the above expression, we reach a contradiction. This proves that the choice for x' is unique when p_i is correct.

Finally, if $p_i \in \Omega^\star, p_i \notin A^\star$ the claim states that if p_j remains alive he will output $x' \in \{x, \perp\}$.
Indeed, since for any two k, m from $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ we have that $\Omega_k \cup \Omega_m \not\supseteq \mathcal{P}$ we can be certain that even if $P_j^{(\text{"}\mathsf{n/v}\text{"})} \subseteq \Omega_k$ and $P_j^{(\perp)} \subseteq \Omega_m$ there will still exist a $p_h \in \mathcal{P}$ who will receive successfully the value x from p_i and forward it to p_j. Otherwise, if p_j would only receive the wrong value $\overline{x}$ it would mean that all parties who received x are in $P_j^{(\perp)}$. This gives us

$$\mathcal{P} = P_j^{(\text{"}\mathsf{n/v}\text{"})} \cup P_j^{(\perp)} \cup P_j^{(\overline{x})} \subseteq \Omega_k \cup \Omega_m \cup A_m \subseteq \Omega_k \cup \Omega_m, \tag{11}$$

which contradicts $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$.

Now, if the adversary chose to send another value $\overline{x}$ to p_j and there existed justifying classes Z_m, Z_k for both x and $\overline{x}$ then p_j would output $\perp$, since none of them is unique. Still, there is no way that p_j could output a wrong value $\overline{x} \neq x$ if $p_i \notin A^\star$. □

Given that claim, we can now continue our simulation proof. Let $\mathcal{A}$ be an adversary attacking the protocol. We define a simulator $\mathcal{S}$ (an efficient algorithm) that interacts with the functionality $\mathcal{F}_{FR}$ and creates the same effect as the real world adversary $\mathcal{A}$, i.e. makes the environment's/distinguisher's view in the ideal experiment identically distributed as the one in the real experiment with adversary $\mathcal{A}$. A description of a respective simulator for broadcast can be found in [Zik10], on which we base the core of our simulator. We note here that our protocol is deterministic and thus the simulator in all cases learns all inputs to the protocol (this happens in general in the next protocols as well, here only p_i has input), so it is easy to perfectly simulate all the messages sent by honest players. This means that the only thing that one needs to verify is that the output of the protocol is identically distributed in the ideal and real world. To do this we make use of the Claim 1 stated above. Our simulator is built in the following way.

Description of the Simulator. Firstly, the simulator invokes $\mathcal{A}$. Given the corrupted class $Z^\star$, the adversary $\mathcal{A}$ controls the corrupted parties $p_c \in A^\star$ and which messages of any player $p_m \in \Omega^\star$ are blocked. The simulator simulates all players taking part in the computation and corrupts the same parties as $\mathcal{A}$. He initiates an interaction where he uses $\mathcal{A}$ in a straight-line black-box manner, simulating the protocol messages towards $\mathcal{A}$ and receiving from $\mathcal{A}$ the messages and/or actions (e.g., drop a message) of corrupted parties. In more detail:
During the first round of the functionality, the simulator $\mathcal{S}$ learns the input value x through the leakage function $l(x)$ and the functionality sets $m_{out} = x$. In the real world the adversary learns x from the corrupted parties that receive it. (the case where no parties are actively corrupted at all is trivial and omitted).

- If the sender is honest, during the second round, we have two cases.

- If p_j is honest, then p_j in the ideal world remains alive, and receives the output m_{out} that was recorder by $\mathcal{F}_{FR}$, which as we see in Claim 1 is the same as what happens in the protocol in this case.
- If the receiver p_j is in $\Omega^\star$ and the adversary decides to block more messages than what can be justified, i.e. for any $Z \in \mathcal{Z}$ we have that $P_j^{(\perp)} \not\subseteq (A \cup \Omega)$, then p_j becomes a zombie. The simulator observers that. To create the same effect, $\mathcal{S}$ sends (inform omission, p_j) to the functionality, so p_j becomes a zombie and the output she gets is $m_{out} = \perp$, as in the real world.
  In the opposite case, if p_j does not become a zombie, then from Claim 1 we are granted that p_j will output the correct value $x' = x$. As the simulator observes that p_j remained alive, in the ideal world the functionality receives (fetch-output) from p_j and outputs $m_{out} = x$, creating the same effect again as above, when p_j was honest.
- If p_j is actively corrupted, she is controlled by $\mathcal{A}$ and in both worlds outputs the value that $\mathcal{A}$ selects. The adversary learns no additional information in both worlds, except from the input value, which he already knew.

- If the sender p_i is in $A^\star$, in the real world the adversary $\mathcal{A}$ has full control over what the receiver p_j will get as output x'. During the first round of the real world execution, all parties receive some x_k (could be that $x_k = \perp$) and $\mathcal{A}$ already knows the value x'. During round 2 the real world adversary $\mathcal{A}$ decides to deliver a value to p_j or in the case that $p_j \in \Omega^\star$ make him a zombie. The simulator observes this and if p_j outputs x', $\mathcal{S}$ sends (adv-input, sid, x') to $\mathcal{F}_{FR}$, making $\mathcal{F}_{FR}$ set $m_{out} = x'$. So p_j outputs the same value in both worlds.
  If p_j became a zombie (he can only become a zombie if he is omission-corrupted according to Claim 1) then $\mathcal{S}$ observes this and sends the message (inform omission, p_j) to $\mathcal{F}_{FR}$, creating the same effect in both worlds.
- If the sender p_i is in $\Omega^\star$, then the adversary can block sent messages from p_i. This would make some $p_k \in \mathcal{P}$ send "n/v" to p_j.
  - If p_j outputs $\perp$ during round 2 according to the protocol because she was not able to determine the correct value for x, the simulator observers that. Then, in the ideal world the simulator sends (adv-omit, sid, $\perp$) during round 2 and the functionality sets $m_{out} = \perp$. So, when the receiver sends (fetch-output) she receives $\perp$, as per in the real world. From Claim 1 we are granted that p_j would output $\perp$ only if the sender is not correct.
  - If p_j is able to output a value x', again from the above Claim, we are certain that this was the input value of p_i. Then the simulator lets the functionality deliver m_{out} upon receiving (fetch-output) from p_j and the output is the same in both worlds.

As a result the view of the adversary and the outputs of the honest parties[9] in the real world are distributed identically as the respective view and outputs in the ideal world, making the two executions indistinguishable for any environment/distinguisher.

□

Next, we will show that this bound is actually tight, meaning that the $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition is also necessary for FixReceive.

Claim. If the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 3) does not hold, then no protocol can satisfy the properties of the *FixReceive* protocol stated in Claim 1, as well.

Proof. Assuming that the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ does not hold we get that $\exists\, Z_1 = (A_1, \Omega_1), Z_2 = (A_2, \Omega_2) \in \mathcal{Z}$ such that

$$\Omega_1 \cup \Omega_2 = \mathcal{P} \tag{12}$$

which means that the whole player set is covered by the two corruptible sets.

[9] Even though this is a multiparty protocol, only p_i and p_j have effective input/output so for simplicity we only mention that input/output can assume that all other parties output $\perp$.

Here, without loss of generality, we can assume that $\Omega_1 \cap \Omega_2 = \emptyset$. If not, we can reduce the sets to the case above by considering a strictly weaker adversary structure and still the condition holds. Now, we will use a player-simulation argument. If we create a contradiction by considering only two players, p_1, p_2, and show that the condition is broken there, then it is trivial that the proof is complete. For that reason, we consider an adversary structure $\mathcal{Z}$ as seen in the table. Towards contradiction, we will have the player p_1 send a message to player p_2 and we will examine two cases.

	p_1	p_2
Z_1	ω	
Z_2		ω

Table 1. Either p_1 or p_2 is omission-corrupted in classes Z_1, Z_2 respectively.

Scenario 1: The adversary chooses the corruptible set Z_1 with $A = \emptyset, \Omega = \{p_1\}$, where p_1 is omission-corrupted and p_2 is honest. As such, the adversary can make p_1 send nothing to p_2 when he was instructed to send his input x to him. That would make p_2 receive the empty value $\perp$, as is the case when the recipient receives nothing. As a result, p_2 should output $\perp$, denoting that he did not receive a value from p_1.

Scenario 2: The adversary chooses the corruptible set Z_2, with $A = \emptyset, \Omega = \{p_2\}$, where p_2 is omission-corrupted and the sender p_1 is honest. If in that case the adversary blocks all messages addressed to p_2 then he should become a zombie and step down from the computation process, because he cannot predict the value he ought to output, since p_1 is honest.

However, both scenarios are indistinguishable for p_2 and we have used the same set up. As a result, the output should be the same in both cases, which gives us a contradiction. We note that as an the actively corrupted party could behave exactly like an omission-corrupted one, the same proof would hold true for $A_1 \cup \Omega_2 = \mathcal{P}$, as well. □

2.3 Weak Consensus

Functionality $\mathcal{F}_{WC}(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Initially, set the input values $x_1, \ldots, x_n$, the output values $y_1, \ldots, y_n$ to $\perp$.
- In round $\rho = 1$:
  - Upon receiving a message (input, sid, v_i) from any $p_i \in \mathcal{P}$ (or the adversary, if the player is actively corrupted), set $x_i = v_i$ and send (leakage, sid, p_i, v_i) to the adversary.
- In round $\rho = 2$ and 3: Do nothing.
- In round $\rho = 4$, $\mathcal{F}_{WC}$ sets $(y_1, \ldots, y_n) = (v_t, \ldots, v_t)$, where v_t is the input value of the first honest party:
  - Upon receiving (adv-omit, sid, $p_i, \perp$) from the adversary, if $p_i \in \Omega^\star$, set $x_i = \perp$.
  - Upon receiving (set y, $(v_1, \ldots, v_n)$) from the adversary, where all $v_i \in \{v, \text{“n/v”}\}$, if the inputs of all (alive) $p_i \in \mathcal{P} \setminus A^\star$ are not the same value v then set $(y_1, \ldots, y_n) = (v_1, \ldots, v_n)$.
    Otherwise, ignore.
  - Upon receiving (inform omission, p_j) from the adversary, if $p_j \in \Omega^\star$, set $y_j = \perp$ and output (omission, p_j) to p_j.
  - Upon receiving (fetch-output, sid) from $p_i \in \mathcal{P}$, send (output, sid, y_i) to p_i and (fetch-output, sid, p_i) to the adversary.

By use of Fix Receive, we will now establish an initial, basic form of consensus, called Weak Consensus. For this, we require the following properties. *Persistency:* If all alive, not actively-corrupted parties start with the same input x then all of them should output $y = x$. *Consistency:* Additionally, there cannot be disagreement between them. To do this, we allow them to output a special character “n/v” (no value) if they are unsure. So, all of them can output either the common value y or “n/v”. However, no two correct players should have contradicting values. Our functionality $\mathcal{F}_{WC}$ above captures those requirements.
Initially, it sets everything to $\perp$ and then receives input from the players. Again, the adversary learns those values, as in FixReceive. Afterwards, once FixReceive is concluded, the adversary is allowed to affect the output. However, he is bound by the consistency and persistency properties we mentioned. As such, he can

only set the outputs to either v or "$\mathsf{n/v}$", if there is no pre-agreement on the inputs. The only other action he can perform is to make players in $\Omega^\star$ become zombies, by informing them of their omission status.

Now, our *WeakConsensus* is realized as follows, using (as do all follow-up protocols) *FixReceive* for party-to-party communication. First, we have every player send his input x_i to all players (using *FixReceive*). Then, each player looks at all the possible classes of the adversary structure for the following: 1) If there exists one $Z = (A, \Omega)$ which gives him a value $x \in \mathbb{F}$ such that this value was sent to p_j by some players who are not corrupted *and* 2) the values he received which are different from both x and $\perp$ can be justified by the set of actively corrupted players, meaning that a malicious player sent the disagreeing value. Additionally, all $\perp$ values should be coming from $p_k \in \Omega$, because FixReceive gives us the guarantee that an alive player only outputs $\perp$ if the sender is not correct. If that is the case, the player adopts this value x as his output. Otherwise, he cannot be certain and outputs the message "$\mathsf{n/v}$". The protocol follows.

Protocol *WeakConsensus*$(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

- In round $\rho = 1$: Each $p_i \in \mathcal{P}$ sends x_i, by invocation of *FixReceive*, to every $p_j \in \mathcal{P}$, who denotes the set of players who sent him 0 as $P_j^{(0)}$, respectively $P_j^{(1)}$ for 1 and those who he did not receive a value from as $P_j^{(\perp)}$.
- In round $\rho = 2$ and 3: Do nothing (waiting for *FixReceive* to conclude).
- In round $\rho = 4$: Each p_j outputs y_j, according to the following.

$$y_j =: \begin{cases} 0 \text{ if } \exists (A,\Omega) \in \mathcal{Z} \text{ s.t. } \mathcal{P} \setminus \left(P_j^{(0)} \cup P_j^{(\perp)}\right) \subseteq A \wedge P_j^{(\perp)} \subseteq \Omega \wedge \mathcal{H} \subseteq P_j^{(0)} \\ 1 \text{ if } \exists (A,\Omega) \in \mathcal{Z} \text{ s.t. } \mathcal{P} \setminus \left(P_j^{(1)} \cup P_j^{(\perp)}\right) \subseteq A \wedge P_j^{(\perp)} \subseteq \Omega \wedge \mathcal{H} \subseteq P_j^{(1)} \\ \text{"}\mathsf{n/v}\text{" otherwise} \end{cases}$$

If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Lemma 2. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol* WeakConsensus *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{WC}$.*

Proof. Instead of proving that protocol *WeakConsensus* securely realizes the functionality $\mathcal{F}_{WC}$, we will do it for the hybrid protocol $\mathbf{Hyb}_{WC}$ which instead of using protocol *FixReceive*, it makes ideal calls to functionality $\mathcal{F}_{FR}$ internally (by ideal call to $\mathcal{F}_{FR}$ we mean an invocation of $\mathcal{F}_{FR}$ as in the ideal world). From there, the statement of the lemma follows using the composition lemma of Canetti [Can01]. Before we give our simulation-based proof, we need to state the following useful claim about the input/output properties of the protocol.

Claim 2. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol* WeakConsensus *has the following properties:* (weak consistency) *There exists some $y \in \mathbb{F}$ such that every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j \in \{y, \text{"}\mathsf{n/v}\text{"}\}$.* (persistency) *If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of* WeakConsensus *has the same input x, then all alive players at the end of the protocol output $y = x$.*

Proof. First, we begin by observing that the choice of p_j for y_j is unique, meaning that there is no way that he could select both 0 and 1 as his output.
To prove that, assume towards contradiction that there exist $(A_0, \Omega_0), (A_1, \Omega_1) \in \mathcal{Z}$ s.t.

$$\mathcal{P} \setminus (P_j^{(0)} \cup P_j^{(\perp)}) \subseteq A_0 \text{ and } \mathcal{P} \setminus (P_j^{(1)} \cup P_j^{(\perp)}) \subseteq A_1,$$

as well as $P_j^{(\perp)} \subseteq \Omega_0$ and $P_j^{(\perp)} \subseteq \Omega_1$. This gives us that $P_j^{(1)} \subseteq A_0$ and $P_j^{(0)} \subseteq A_1$ together with $P_j^{(\perp)} \subseteq \Omega_0 \cap \Omega_1$, which leads us to

$$\begin{aligned} \mathcal{P} &= P_j^{(0)} \cup P_j^{(1)} \cup P_j^{(\perp)} \\ &\subseteq A_1 \cup A_0 \cup (\Omega_0 \cap \Omega_1), \end{aligned} \tag{13}$$

which is covered by $A_k \cup A_1 \cup A_0 \cup (\Omega_0 \cap \Omega_1)$ for some k and contradicts the $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition. Hence, the selection of y_j is unique.

(Weak consistency) Now, let us assume towards contradiction, that there exist (alive) players $p_k, p_m \in \mathcal{P} \setminus A^\star$ with outputs $y_k = y, y_m = y'$, respectively, and $y \neq y'$. Without loss of generality due to the observation above, we can assume that $y_k = 0$ and $y_m = 1$. We define the sets $P_k^{(0)}, P_k^{(1)}$ and $P_k^{(\perp)}$ for p_k and similarly the sets $P_m^{(0)}, P_m^{(1)}, P_m^{(\perp)}$ for p_m. Because both $p_k, p_m \notin A^\star$ are alive and follow the protocol *WeakConsensus* we get that

$$P_k^{(1)} \subseteq A_k \text{ and } P_k^{(\perp)} \subseteq \Omega_k, \text{ as well as } P_m^{(0)} \subseteq A_m \text{ and } P_m^{(\perp)} \subseteq \Omega_m, \tag{14}$$

from where we can easily deduce that

$$P_k^{(\perp)} \cap P_m^{(\perp)} \subseteq \Omega_k \cap \Omega_m, \tag{15}$$

and then we can write

$$\begin{aligned}
\mathcal{P} =& P_k^{(\perp)} \cup P_k^{(1)} \cup P_k^{(0)} && (16)\\
=& \left(P_k^{(\perp)} \cap \mathcal{P}\right) \cup P_k^{(1)} \cup \left(P_k^{(0)} \cap \mathcal{P}\right) && (17)\\
=& \left(P_k^{(\perp)} \cap \left(P_m^{(\perp)} \cup P_m^{(0)} \cup P_m^{(1)}\right)\right) \cup P_k^{(1)} \cup \left(P_k^{(0)} \cap \left(P_m^{(\perp)} \cup P_m^{(0)} \cup P_m^{(1)}\right)\right) && (18)\\
=& \left(\left(P_k^{(\perp)} \cap P_m^{(\perp)}\right) \cup \left(P_k^{(\perp)} \cap P_m^{(0)}\right) \cup \left(P_k^{(\perp)} \cap P_m^{(1)}\right)\right) \cup P_k^{(1)} \cup && (19)\\
& \cup \left(\left(P_k^{(0)} \cap P_m^{(\perp)}\right) \cup \left(P_k^{(0)} \cap P_m^{(0)}\right) \cup \left(P_k^{(0)} \cap P_m^{(1)}\right)\right) && (20)\\
\subseteq& (\Omega_k \cap \Omega_m) \cup A_m \cup A_k \cup A_k && (21)\\
& \cup A_m \cup A_m \cup A^\star && (22)\\
=& A_k \cup A_m \cup A^\star \cup (\Omega_k \cap \Omega_m), && (23)
\end{aligned}$$

where the inequalities on (21) follow along the lines of

i. $P_k^{(\perp)} \cap P_m^{(\perp)} \subseteq \Omega_k \cap \Omega_m$ from relation (15),
ii. brackets involving $P_k^{(1)}$ and $P_m^{(0)}$ come from (14),
iii. for the case where the message to one player is $\perp$ we know that this message was blocked from reaching its recipient. If it was not, and the sender was following the protocol, he would have sent the same message to p_k and p_m, giving us the above inequalities for the sets from (14), again. Otherwise, he would not be following the protocol and he would belong in $A^\star$,

giving us the final relation, which contradicts the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$.
(persistency) If all non-actively corrupted players $p_k \in \mathcal{P} \setminus A^\star$ who are alive at the beginning of *WeakConsensus* have the same input x, without loss of generality we can assume that $x = 0$. The case where $x = 1$ is clearly symmetrical. Then for all receivers $p_j \in \mathcal{P} \setminus A^\star$ who are alive, we have that $P_j^{(\perp)} \subseteq \Omega^\star$ and $P_j^{(1)} \subseteq A^\star$, according to Claim 1. This means that the condition for $y_k = 0$ at *WeakConsensus* Protocol is satisfied. Hence all non-actively corrupted players who are still alive output $x = 0$, which is unique by the weak consistency property. □

Continuing with the proof of Lemma 2, a simulator for the security proof can be created similarly to the one for *FixReceive*. Since our protocols for the whole BA section are deterministic and do not involve randomness, once the simulator learns the inputs of the players and what players are becoming zombies, he can perfectly simulate the view of the environment in the real and ideal world execution. Due to space limitations we omit the rest of the simulators of this section. □

2.4 Graded Consensus

Functionality $\mathcal{F}_{GC}(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Initially, set the input values $x_1, \ldots, x_n$, the output values $y_1, \ldots, y_n$, $g_1, \ldots, g_n$ to $\perp$.
- In round $\rho = 1$:
  - Upon receiving a message (input, sid, v_i) from $p_i \in \mathcal{P}$ (or the adversary, if the player is actively corrupted), set $x_i = v_i$ and send (leakage, sid, p_i, v_i) to the adversary.
- In round $\rho = 2$ up to 7: Do nothing. (Wait for *WeakConsensus* to conclude.)
- In round $\rho = 8$, $\mathcal{F}_{GC}$ sets $(y_1, \ldots, y_n) = (v_t, \ldots, v_t)$, where v_t is the input value of the first honest party and sets $(g_1, \ldots, g_n) = (1, \ldots, 1)$:
  - Upon receiving (adv-omit, sid, $p_i, \perp$) from the adversary, if $p_i \in \Omega^\star$, set $x_i = \perp$. Otherwise, discard the message.
  - Upon receiving (set, $\vec{z} = (z_1, \ldots, z_n)$) where either $z_i = \{v_i, b_i\}$ or $z_i = \perp$.
    - If $z_i = \perp$ for some $p_i \notin \Omega^\star$, discard the message.
    - If $\forall$ alive $p_i \in \mathcal{P} \setminus A^\star$ the functionality has recorded the same $x_i = x$ then ignore all messages (set).
    - Else if $b_k = 1$ for some alive $p_k \notin A^\star$ then ignore all v_i for $p_i \notin A^\star$ and set $(y_1, \ldots, y_n) = (v_k, \ldots, v_k)$ and $(g_1, \ldots, g_n) = (b_1, \ldots, b_n)$.
    - Else, set $(y_1, \ldots, y_n) = (v_1, \ldots, v_n)$ and $(g_1, \ldots, g_n) = (b_1, \ldots, b_n)$, for $b_i \in \{0, 1\}$. If $z_j = \perp$ for some $p_j \in \Omega^\star$, set $y_j = \perp$ and output (omission, p_j) to p_j. Discard all other messages.
  - Upon receiving (fetch-output, sid) from $p_i \in \mathcal{P}$, send (output, sid, $\{y_i, g_i\}$) to p_i and (fetch-output, sid, p_i) to the adversary.

The next step is *GradedConsensus*. Leveraging Weak Consensus to get a stronger type of agreement, the players here also output a bit grade g_i reflecting their certainty in their output. This is similar to the Gradecast primitive in [FM88]. This is a graded version of persistence—i.e., if the players who are not actively corrupted have pre-agreed on a value x, then we get that they all output x with grade $g = 1$ (*graded persistency*). Additionally, it ensures that if any non-actively corrupt party outputs $y_i = y$ with $g_i = 1$, then every non-actively corrupt alive party p_j outputs $y_j = y$ (*graded consistency*). In the opposite case, where the player is not certain about his output value, his grade of confidence is $g_i = 0$.

Our functionality $\mathcal{F}_{GC}$ above captures those properties. At its core it works in a similar manner to $\mathcal{F}_{WC}$. The difference here is that if there exists pre-agreement then all grades are set to 1 and outputs to the same value and the adversary is not allowed to change them. Else, if some grade is $g_i = 1$, then all outputs have to be the same, but the adversary can select the other grades. Otherwise, with all grades 0, the outputs are allowed to be selected by the adversary.

In more detail, the protocol first calls the *WeakConsensus* protocol in order to reach an initial step of agreement. Then all the players exchange the outputs they received, by invocation of *FixReceive*. After that, each player collects that information and decides whether to output $y_j = 0$ or $y_j = 1$. If it can be seen from the adversary structure that non-actively corrupted players have sent the value 1, then $y_j = 1$ (as in this case every non-actively corrupted player would have output $x'_i \in \{y_j, \text{“}\mathsf{n/v}\text{”}\}$, according to the previous Claim). Otherwise, if she only received 0 and “n/v” from non-actively corrupted, she sets her output (by default) to $y_j = 0$.

Next, we have to determine the grade. If at least all non-actively corrupted players have sent to p_j either the same message as her output y_j or $\perp$ (from the players in Ω) *and* at least all uncorrupted players have definitely sent y_j, then p_j sets her grade of confidence in the fact that all uncorrupted players have the same output as $g_j = 1$. Otherwise, she sets $g_j = 0$, showing that there is no agreement from her point of view, yet. The protocol follows. We only list rounds $\rho = 1, 5, 8$ because in the rest of the rounds the players are waiting for *WeakConsensus* and *FixReceive* to give their output.

Protocol *GradedConsensus*$(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

1. In round $\rho = 1$: Invoke *WeakConsensus* $(\mathcal{P}, \mathcal{Z}, \vec{x})$; Player p_i denotes his output by x'_i.
2. In round $\rho = 5$: Each $p_i \in \mathcal{P}$ sends x'_i, by invocation of *FixReceive*, to every $p_j \in \mathcal{P}$, who denotes the received value by $x_j^{(i)}$.
3. In round $\rho = 8$: Each p_j outputs (y_j, g_j) as follows:

$$y_j =: \begin{cases} 0 & \text{if } \exists (A, \Omega) \in \mathcal{Z} \text{ s.t. } \mathcal{P} \setminus \left(P_j^{(0)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)} \right) \subseteq A \ \wedge \left(P_j^{(\perp)} \subseteq \Omega \right) \\ 1 & \text{otherwise} \end{cases}$$

$$g_j =: \begin{cases} 1 & \text{if } \exists (A, \Omega) \in \mathcal{Z} \text{ s.t. } \mathcal{P} \setminus \left(P_j^{(y_j)} \cup P_j^{(\perp)} \right) \subseteq A \\ 0 & \text{otherwise} \end{cases}$$

If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Lemma 3. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 7) holds, the protocol* GradedConsensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{GC}$*.*

The claim stating the input/output properties of the protocol follows.

Claim. If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol *GradedConsensus* has the following properties: *(graded consistency)* If some $p_i \in \mathcal{P} \setminus A^\star$ outputs $(y_i, g_i) = (y, 1)$ for some $y \in \mathbb{F}$, then every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y$ with some $g_j \in \{0, 1\}$. *(graded persistency)* If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of *GradedConsensus* has input $x_i = x$, then every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $(y_j, g_j) = (x, 1)$.

Proof. To begin with, we will show that for any player $p_j \in \mathcal{P} \setminus A^\star$, $y_j = 1$ only when $\exists\ Z = (A, \Omega) \in \mathcal{Z}$ s.t.

$$\mathcal{P} \setminus \left(P_j^{(1)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)} \right) \subseteq A \ \wedge \left(P_j^{(\perp)} \subseteq A \cup \Omega \right).$$

In order to do that, let us consider under which case a player p_j would not output $y_j = 0$. This would only happen when the first condition for $y_j = 0$ would not hold true, i.e. $\forall Z = (A, \Omega) \in \mathcal{Z}$ we would have that

$$\mathcal{P} \setminus \left(P_j^{(0)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)} \right) \nsubseteq A \text{ OR } \left(P_j^{(\perp)} \nsubseteq A \cup \Omega \right),$$

which means that one of the two statements (either the first or the second) will always hold. As such, we can select any special $Z = (A, \Omega)$ with $A \supseteq A^\star$ and $\Omega \supseteq \Omega^\star$ of $Z^\star = (A^\star, \Omega^\star)$ for which we know that the second argument is false, as we know that $P_j^{(\perp)} \subseteq A^\star \cup \Omega^\star$. We are allowed to make that selection because the corruptible set $Z^\star$ that the adversary eventually chooses belong in $\mathcal{Z}$ and can be used for our analysis, despite being unknown to the players. As a result, for that Z the first argument has to hold true, i.e. $\exists\ p_k \in \mathcal{P}$ with

$$p_k \notin P_j^{(0)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)}, \text{ s.t. } p_k \notin A^\star,$$

which means that p_k is not actively corrupted. After the invocation of *WeakConsensus* p_k's output was $x'_k = 1$ and consequently he sent this value 1 to p_j. By the Weak Consistency property though, this would mean that *all* non-actively corrupted players would output either 1 or “n/v”, as well, and none of them would output 0. This leads us to

$$P_j^{(1)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)} \supseteq \mathcal{P} \setminus A, \tag{24}$$

which, together with the earlier result of $P_j^{(\perp)} \subseteq A \cup \Omega$ can be written as

$$\mathcal{P} \setminus \left(P_j^{(1)} \cup P_j^{(\text{“n/v”})} \cup P_j^{(\perp)} \right) \subseteq A \ \wedge \left(P_j^{(\perp)} \subseteq A \cup \Omega \right),$$

completing our argument.

Now we can move on to the proof of the two properties.

(Graded consistency) Let us assume, towards contradiction, that there exist (alive) $p_k, p_m \in \mathcal{P} \setminus A^\star$ with

$(y_k, g_k) = (0,1)$ and $(y_m, g_m) = (1, g_m)$, where $g_m \in \{0,1\}$, their respective outputs. We define the sets $P_k^{(0)} = \{p_i : x_k^{(i)} = 0\}$, $P_k^{(1)} = \{p_i : x_k^{(i)} = 1\}, P_k^{(\perp)} = \{p_i : x_k^{(i)} = \perp\}$ and $P_k^{(\text{“n/v”})} = \{p_i : x_k^{(i)} = \text{“n/v”}\}$ for p_k and the sets $P_m^{(0)}, P_m^{(1)}, P_m^{(\perp)}, P_m^{(\text{“n/v”})}$ for p_m in the same manner. As both p_k and p_m are not malicious and they follow the protocol we get that there exist sets (A_k, Ω_k) and (A_m, Ω_m) for which we have that

$$\mathcal{P} \setminus \left(P_k^{(0)} \cup P_k^{(\perp)}\right) \subseteq A_k \tag{25}$$

$$\mathcal{P} \setminus \left(P_m^{(1)} \cup P_m^{(\text{“n/v”})} \cup P_m^{(\perp)}\right) \subseteq A_m, \tag{26}$$

due to the fact that $g_k = 1$ and $y_m = 1$. This can be written as

$$P_k^{(1)} \cup P_k^{(\text{“n/v”})} \subseteq A_k \tag{27}$$

$$P_m^{(0)} \subseteq A_m, \tag{28}$$

as those are the only possible sets that $\mathcal{P}$ can be divided into.
Now, we are ready to write that

$$\begin{aligned}
\mathcal{P} =& P_k^{(0)} \cup P_k^{(\perp)} \cup P_k^{(1)} \cup P_k^{(\text{“n/v”})} \\
=& \{P_k^{(0)} \cap \mathcal{P}\} \cup \{P_k^{(\perp)} \cap \mathcal{P}\} \cup P_k^{(1)} \cup P_k^{(\text{“n/v”})} \\
=& \left\{[P_k^{(0)} \cap P_m^{(0)}] \cup [P_k^{(0)} \cap P_m^{(1)}] \cup [P_k^{(0)} \cap P_m^{(\text{“n/v”})}] \cup [P_k^{(0)} \cap P_m^{(\perp)}]\right\} \cup \\
& \cup \left\{[P_k^{(\perp)} \cap P_m^{(0)}] \cup [P_k^{(\perp)} \cap P_m^{(1)}] \cup [P_k^{(\perp)} \cap P_m^{(\text{“n/v”})}] \cup \underbrace{[P_k^{(\perp)} \cap P_m^{(\perp)}]}_{\subseteq\, \Omega_k \cap \Omega_m}\right\} \cup \\
& \cup \underbrace{[P_k^{(1)} \cup P_k^{(\text{“n/v”})}]}_{\subseteq A_k} \\
\subseteq & \quad A_m \quad \cup \quad A^\star \quad \cup \quad A^\star \quad \cup \quad A^\star \quad \cup \\
& \cup \quad A_m \quad \cup \quad A_k \quad \cup \quad A_k \quad \cup \quad (\Omega_k \cap \Omega_m) \quad \cup A_k \\
=& A_k \cup A_m \cup A^\star \cup (\Omega_k \cap \Omega_m),
\end{aligned} \tag{29}$$

where we have used the relations (27),(28) from above and the fact that if $p_i \in P_k^{(\alpha)} \cap P_m^{(\beta)}$ with different exponents $\alpha \neq \beta$ it means that p_i has sent different messages to the players p_k, p_m, i.e. he is actively corrupted.
Also, when a player p_k receives the value $\perp$ from a player p_i we know that this message was blocked from reaching its recipient. If it was not, and the sender was following the protocol, he would have sent the same message to p_k and p_m, giving us the above inequalities for the sets from (28). Otherwise, he would not be following the protocol and he would belong in $A^\star$, in which case we get the same result. This is the reasoning used in the blue underlined brackets.
This final result we reached contradicts the assumption of the $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition, giving us the desired result.

(Graded persistency) If we assume that every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of *GradedConsensus* has input $x_i = 0$, we can use the Weak persistency property that we have established to deduce that every non-actively corrupted p_j (who is still alive) outputs $x'_j = 0$ in Step 1 (the case of pre-agreement on 1 can be handled symmetrically).
Hence, for every p_i we have that $\mathcal{P} \setminus (P_i^{(0)} \cup P_i^{(\perp)}) \subseteq A^\star$, and $P_i^{(\perp)} \subseteq A^\star \cup \Omega^\star$ which means that p_i outputs 0 with grade 1, i.e. $(y_i, g_i) = (0,1)$. The uniqueness of this decision follows from the graded consistency property and this concludes our proof. □

2.5 King Consensus

The last step towards Consensus is *KingConsensus*. Here, we select one player and give him the special role of *(phase) king*. As before, if the players had pre-agreement on their inputs x, they all must output the same value x (persistency). On top of that, if the king is correct, the players will reach agreement, no matter what (king consistency).

The functionality $\mathcal{F}_{KC}$ that captures that is described above It keeps the persistency if it is already established. Otherwise, it could allow the adversary to change the output to a specific value v for all, subject to the king being correct. Else, the adversary is allowed to select the outputs for all players.
The protocol realizing it first uses *GradedConsensus* and then has the king send his output to all players. All players certain for their output keep their value, whereas all those who are uncertain adopt the value of the king. This way, using graded consistency for the grades and persistency we make sure that if the king is correct until the end of the protocol then all non-actively corrupted players will agree on the same output. The protocol follows.

Protocol *KingConsensus*$(\mathcal{P}, \mathcal{Z}, p_k, \vec{x} = (x_1, \ldots, x_n))$

1. In round $\rho = 1$: Invoke *GradedConsensus* $(\mathcal{P}, \mathcal{Z}, \vec{x})$; p_i denotes his output by (x'_i, g_i).
2. In round $\rho = 9$: The king p_k sends x'_k to every $p_j \in \mathcal{P}$ by invocation of *FixReceive*.
3. In round $\rho = 12$: Each $p_j \in \mathcal{P}$ outputs $y_j := \begin{cases} x'_j & \text{if } (g_j = 1) \text{ or } (p_k \text{ sent } x'_k \notin \mathbb{F}) \\ x'_k & \text{otherwise} \end{cases}$

   If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Lemma 4. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol* KingConsensus *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{KC}$.*

The claim stating the input/output properties of the protocol follows.

Claim 3. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol* KingConsensus *has the following properties:* (king consistency) *If the king p_k is correct, then every $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y$.* (persistency) *If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of* KingConsensus *has input $x_i = x$ then every (alive) p_j outputs $y_j = x$.*

Proof. (Persistency) When the non-actively corrupt players who are alive pre-agree on some y then, by the graded persistency property, they all output y with grade 1 and do not change it.
(King consistency) When the king p_k is correct, he sends the same x'_k to every player. We consider two cases: (i) If some $p_i \in \mathcal{P} \setminus A^\star$ outputs some x'_i with grade $g_j = 1$, then by graded consistency, all alive and correct players output the same x'_i, including the king $(x'_k = x'_i)$, who then sends this same value to everyone. (ii) If all players have grade 0 then they all output the value they received from the king, the same x'_k for all, completing our proof. □

2.6 Consensus

Functionality $\mathcal{F}_{CS}(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Initially, set the input values $x_1, \ldots, x_n$, the output values $y_1, \ldots, y_n$ to $\perp$.
- In round $\rho = 1$:
  - Upon receiving a message (input, sid, v_i) from $p_i \in \mathcal{P}$ (or the adversary, if the player is actively corrupted), set $x_i = v_i$ and send (leakage, sid, p_i, v_i) to the adversary.
- In round $\rho = 2$ up to $12n$ do nothing.
- In round $\rho = 12n + 1$, $\mathcal{F}_{CS}$ sets $(y_1, \ldots, y_n) = (v_t, \ldots, v_t)$, where v_t is the input value of the first honest party:

- If $\forall$ alive $p_i \in \mathcal{P} \setminus A^\star$ the functionality has recorded the same $x_i = x$ then ignore all messages (set y, $(v_1, \ldots, v_n)$).
- Else, upon receiving (set y, $(v_1, \ldots, v_n)$) from the adversary, if $v_j = \perp$ only for some $p_j \in \Omega^\star$ and $v_i = v_k$ for any two $p_i, p_k \notin A^\star$, set $(y_1, \ldots, y_n) = (v_1, \ldots, v_n)$.
  Otherwise, ignore all (set y) messages.
- If $y_j = \perp$ for some $p_j \in \Omega^\star$, output (omission, p_j) to p_j.
- Upon receiving (fetch-output, sid) from $p_i \in \mathcal{P}$, send (output, sid, y_i) to p_i and (fetch-output, sid, p_i) to the adversary.

Finally, we are now ready to present our Consensus primitive. The end goal of the parties is to terminate with the same output y. On top of that, if there was pre-agreement on input x, the common output should be $y = x$. Our functionality $\mathcal{F}_{CS}$, described above allows the adversary to change the common output value only if there was no pre-agreement. Also, he is able to make a player in $\Omega^\star$ zombie. All other messages are ignored.

The way that the $\mathcal{F}_{CS}$ is realized by the protocol *Consensus* is by repeatedly calling the *KingConsensus* protocol. We use as inputs the outputs of the previous iteration and each time the king is a different player, in turn for all players until we reach an honest one. Since every player becomes a king, we can be certain, as long as not all players are corrupted (which is not allowed by our security condition) that at least one king will be correct and, hence, we will achieve consistency on the output value. This point will be reached even sooner if the non-actively corrupted players have pre-agreed on a value x. What is more, once this agreement is achieved, by the persistency property of *KingConsensus*, we can be certain that it will not change, no matter what the king sends (in case he is not correct) thus the agreement will be maintained.

To be more formal, below is the property-based definition of Consensus for our relevant corruption types. A protocol perfectly $\mathcal{Z}$-securely realizes Consensus among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a $\mathcal{Z}$-adversary:

- *(consistency)* Every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the end of the protocol outputs the same value y.
- *(persistency)* Assuming that every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the beginning of the protocol has input x, the output is $y = x$.
- *(termination)* For every non-actively corrupted $p_i \in \mathcal{P}$ the protocol terminates after a finite number of rounds.

The description of our *Consensus* protocol follows.

Protocol *Consensus*$(\mathcal{P}, \mathcal{Z}, \vec{x} = (x_1, \ldots, x_n))$

1. In round $\rho = 1$: Each $p_i \in \mathcal{P}$ sets $x'_i := x_i$.
   Then, for $k = 1, \ldots, n$ the following steps are run:
   (a) Invoke *KingConsensus*$(\mathcal{P}, \mathcal{Z}, p_k, (x'_1, \ldots, x'_n))$; each p_j denotes his output by $z_j^{(k)}$.
   (b) Each $p_i \in \mathcal{P}$ sets $x'_i := z_i^{(k)}$.
2. In round $\rho = 12n + 1$: Each $p_j \in \mathcal{P}$ outputs $y_j := x'_j$.
   If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Theorem 2. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 7) holds, the protocol* Consensus *perfectly* $\mathcal{Z}$*-securely realizes the Consensus functionality* $\mathcal{F}_{CS}$*.*

Proof. The proof of this theorem follows from the established protocols above. If $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds true we are granted that there exists an uncorrupted player in $\mathcal{P}$, as $\mathcal{P} \setminus (A_i \cup A_j \cup A_k \cup (\Omega_j \cap \Omega_k)) \neq \emptyset$, for all i, j, k selections of the three sets.
Then, applying both properties of *KingConsensus* in succession creates and then maintains the agreement on the output for all iterations. This post-agreement can be achieved earlier still, if there is pre-agreement between the non-actively corrupted players on their values.

Finally, for the termination property, we are certain that the protocol steps (a) and (b) will be run for exactly n times, with each step being guaranteed to terminate. At this point, at least one honest player will have successfully become king, after which point the pre-agreement and the eventual reaching of a common output is reached according to the properties of *KingConsensus*. □

2.7 Broadcast

Functionality $\mathcal{F}_{BC}(\mathcal{P}, \mathcal{Z}, p, x,)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Initially, set the input value x and the output values $y_1, \ldots, y_n$ to $\perp$.
- In round $\rho = 1$:
  - Upon receiving a message (input, sid, v) from p (or the adversary, if the player is actively corrupted), set $x = v$ and send (leakage, sid, p, v) to the adversary.
- In round $\rho = 2$ up to $24n + 6$ do nothing.
- In round $\rho = 24n + 7$, $\mathcal{F}_{BC}$ sets $(y_1, \ldots, y_n) = (v, \ldots, v)$, where v is the input value of p.
  - Upon receiving (inform omission, p) from the adversary, if $p \in \Omega^\star$ set $(y_1, \ldots, y_n) = (\perp, \ldots, \perp)$ and output (omission, p) to p.
  - Upon receiving (set , $\perp, (v_1, \ldots, v_n)$) from the adversary, if $v_j = \perp$ only for some $p_j \in \Omega^\star$ and $v_i = v$ for any other $p_i \notin A^\star$, set $(y_1, \ldots, y_n) = (v_1, \ldots, v_n)$.
    Otherwise, ignore all (set) messages.
  - If $y_j = \perp$ for some $p_j \in \Omega^\star$, output (omission, p_j) to p_j.
  - Upon receiving (fetch-output, sid) from $p_i \in \mathcal{P}$, send (output, sid, y_i) to p_i and (fetch-output, sid, p_i) to the adversary.

In this section we describe our Broadcast functionality $\mathcal{F}_{BC}$ above. The idea is similar to the Consensus functionality, with the difference that only the sender p has an input x. Additionally, if he remains alive, all alive players will get the same output value $y = x$. The adversary can only make players $p_j \in \Omega^\star$ become zombie and nothing more to alter the outputs. An honest p always broadcasts the correct value and the output of broadcast is $\perp$ only if $p \in \Omega^\star$.

The formal property-based definition of *Broadcast* is as follows. A protocol perfectly $\mathcal{Z}$-securely realizes Broadcast with sender a player p whose input is x among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a $\mathcal{Z}$-adversary:

- *(consistency)* Every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the end of the protocol outputs the same value y.
- *(validity)* Assuming that the sender p is not actively corrupted, the common output y satisfies $y \in \{x, \perp\}$. Specifically, $y = x$ if p is alive and correct until the end of the protocol and $y = \perp$ if p has become a zombie.
- *(termination)* For every non-actively corrupted $p_i \in \mathcal{P}$ the protocol terminates after a finite number of rounds.

Note that, as *Broadcast* invokes *Consensus*, the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is needed for this protocol, as well. We describe below our protocol for broadcast that realizes the functionality $\mathcal{F}_{BC}$.

Protocol *Broadcast*$(\mathcal{P}, \mathcal{Z}, p, x)$

1. In round $\rho = 1$: The sender p sends x to every $p_j \in \mathcal{P}$ using *FixReceive*, who denotes the received value by x_j. (If p_j received $\perp$ he sets $x_j = 0$).
2. In round $\rho = 4$: The players invoke *Consensus*$(\mathcal{P}, \mathcal{Z}, (x_1, \ldots, x_n))$ on the received values. We denote p_j's output as y_j.

3. In round $\rho = 12n + 4$: The sender p sends a confirmation bit b to every $p_i \in \mathcal{P}$ using *FixReceive*, where $b = 1$ if the output of p after *Consensus* equals x and $b = 0$ otherwise; p_i denotes the received bit by b_i. (If p_i received $\perp$ he sets $b_i = 0$).
4. In round $\rho = 12n + 7$: Invoke *Consensus*$(\mathcal{P}, \mathcal{Z}, (b_1, \ldots, b_n))$.
5. In round $\rho = 24n + 7$: For each $p_i \in \mathcal{P}$, if p_i's output after *Consensus* is 1, he outputs y_i, otherwise he outputs $\perp$.
   If some $p_z \in \Omega^\star$ became zombie he outputs (omission, p_z).

Theorem 3. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol* Broadcast *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{BC}$.*

The claim stating the input/output properties of the protocol follows.

Claim. If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol *Broadcast* has the following properties: *(consistency)* All (alive) $p_j \in \mathcal{P} \setminus A^\star$ output the (same) value $y_j = y$. *(validity)* When $p \in \mathcal{P} \setminus A^\star$ we get that $y \in \{x, \perp\}$. Specifically, $y = x$ when p is still correct (even if he is omission-corrupted) and he is alive at the end of the protocol, and $y = \perp$ when p has become a zombie during the protocol.

The proof of the Claim can be derived from Consensus above, as a case study is enough to see that consensus grants us the desired properties we need for broadcast, as well. Something to note here is the addition of steps 3 and 4, which would initially seem surplus. The reason is that during Consensus, if there is no pre-agreement, the adversary is allowed to select which value will be output and agreed upon. This can be seen in $\mathcal{F}_{WC}$ and $\mathcal{F}_{GC}$ and it has already been mentioned in 1.3. More specifically, when the sender sends his bit to everyone, if the sender is omission-corrupted it could be the case that some non-actively corrupted parties do not receive his value. Then, according to step 1, they would set their input to 0. Hence, the second stage, i.e. Consensus, could start without pre-agreement on the correct value of the sender, if his initial x was equal to 1.
For this reason, we need the extra confirmation step of the bit send by the sender and the consensus run on this bit in order to make sure that the output of Broadcast is the correct one, by having the sender agree. Aside from those extra steps which we justified, the standard method of sending the senders bit to everyone and then running consensus on the received values gives us the desired properties for broadcast.

Necessity of Conditions for Byzantine Agreement. Next, we will show that the $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition is also necessary for Broadcast. A strawman approach would be the following: Altman, Fitzi, and Maurer [AFM99] proved that the following condition, denoted by $C_{AFM}(\mathcal{P}, \mathcal{Z})$, is necessary and sufficient for BA with general active/fail adversary structures, i.e., where the adversary characterized by a class $Z = (A, F)$ can actively corrupt every party in A and fail-crash every party in F:

$$C_{AFM}(\mathcal{P}, \mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (F_i \cap F_j \cap F_k) \neq \mathcal{P}. \tag{30}$$

Since $C_{AFM}(\mathcal{P}, \mathcal{Z})$ is necessary for active fail corruption, and fail-corruption is strictly weaker than omission one might be tempted to use the above as a way to prove impossibility in our case, too. However, $C_{AFM}(\mathcal{P}, \mathcal{Z})$ is less strict than $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$, i.e., there are structures –indeed, the structure in the counter-example of Lemma 5 is one– satisfying $C_{AFM}(\mathcal{P}, \mathcal{Z})$ but not $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$.

Hence the above approach cannot work, which calls for a completely new impossibility proof. The following lemma shows the impossibility that arises when $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is violated. The proof exploits adversarial strategies which create an ambiguity in the view of the players, which prevents them from deciding which corruptible class the adversary has actually chosen, contradicting correctness.

Lemma 5. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ does not hold, then the properties of the* Broadcast *protocol stated in Theorem 3 cannot hold, as well.*

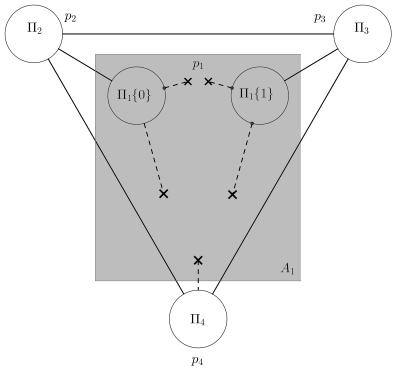

Fig. 1. p_1 is actively corrupted.

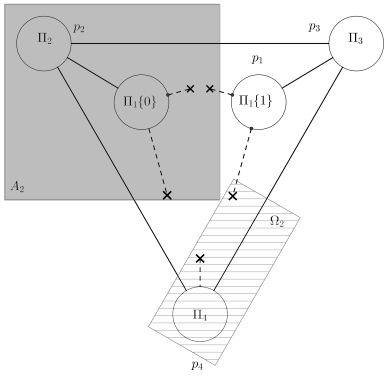

Fig. 2. p_2 is actively, p_4 is omission-corrupted.

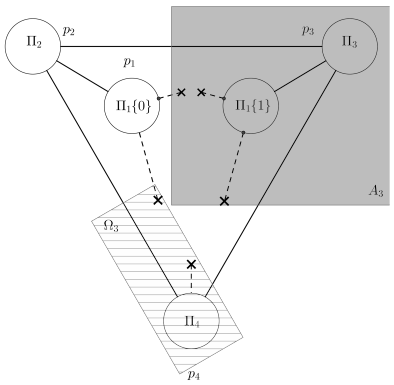

Fig. 3. p_3 is actively, p_4 is omission-corrupted.

Fig. 4. The scenarios $1, 2, 3$ are indistinguishable for the non-actively corrupted players.

Proof. Assuming that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) does not hold and that we have secure broadcast, i.e. the properties of the broadcast protocol hold true we will reach a contradiction. This means that $\exists\ Z_1, Z_2, Z_3 \in \mathcal{Z}$ such that

$$A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P},$$

which means that the whole player set is covered by these three corruptible sets. Here, without loss of generality, we can assume that the sets are not overlapping. If not, we can reduce the sets to the case above by considering a strictly weaker adversary structure and still the condition holds.

Now, we will use a player-simulation argument. If we create a contradiction by considering four players, p_1, p_2, p_3, p_4, assuming that they can securely communicate and show that the condition together with the properties cannot hold there, then it is trivial that the proof is complete. For that reason, we consider an adversary structure $\mathcal{Z}$ as seen in Table 2. Here, for $Z_1, Z_2, Z_3 \in \mathcal{Z}$ we have that $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$. We assume that p_1 is the designated sender and we want all other players to output the same value, according to the broadcast property. We consider the following scenarios. In all cases, communication from p_1 to p_4 is cut entirely.

	p_1	p_2	p_3	p_4
Z_1	α			
Z_2		α		ω
Z_3			α	ω

Table 2. The classes Z_1, Z_2, Z_3 with $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$.

Scenario 1: The adversary chooses to corrupt Z_1 (Fig. 1).

This means that p_1 is actively corrupted and all other players are honest. The adversary invokes the program π_1 that p_1 would normally run but for two different inputs, 0 and 1, and he connects the first program with p_2 and the second one with p_3, respectively. This means that p_2 believes that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the sender.

Scenario 2: The adversary chooses to corrupt Z_2. Furthermore, the sender p_1 has input 1 (Fig. 2).

In that case, p_2 is actively corrupted and p_4 is omission-corrupted. The program π_1 is invoked by the adversary and it is run with input 0. Then, he connects that program with p_2. At the same time p_1 is normally communicating his input 1 to p_3 but his communication with p_4 is blocked. This means that p_2 believes that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the sender.

Because the sender is not actively corrupted and is correct until the end of the protocol, due to validity, all players should output 1 with overwhelming probability.

Scenario 3: The adversary chooses to corrupt Z_3. Furthermore, the sender p_1 has input 0 (Fig. 3).

In that case, p_3 is actively corrupted and p_4 is omission-corrupted, as before. The program π_1 is invoked by

the adversary and it is run with input 1. Then, he connects that program with p_3. At the same time p_1 is normally communicating his input 0 to p_2 but his communication with p_4 is blocked. This means that p_2 believes that the sender has input 0, p_3 believes that the sender has input 1 and p_4 does not have any direct information from the sender.
Because the sender is not actively corrupted and is correct until the end of the protocol, due to validity, all players should output 0 with overwhelming probability.
For the players the three scenarios are indistinguishable. Hence, they should output the same value in all cases, since all scenarios have the same set up, meaning that the distribution of the outputs should be identical.
As the result is overwhelmingly different in all cases, this leads us to a contradiction. □

This proves that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ consists a tight feasibility bound, meaning that the desired properties hold true if and only if our condition holds.
In essence, this proof was done for the necessity of the *Broadcast* protocol. We claim that this is enough for the necessity of the *Consensus* protocol, as well. Indeed, if there existed a consensus protocol that would work while the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ does not hold, we could use *this* consensus protocol in the same manner we have done earlier in order to build a new broadcast protocol. This in turn would break our impossibility result, giving us a contradiction.
Hence, the necessity of the bound for Consensus is covered as well.

3 Multi-Party Computation

In this section we extend our study to multi-party computation. As a first step, in Section 3.1 we discuss the challenges of such an extension, and show that existing techniques from the threshold literature either do not work, or yield counter-intuitive results. There we discuss and prove the ineffectiveness of player elimination, a technique frequently used in the general adversary literature.

This motivates us to introduce a new condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$, which together with $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ enables us to create a Secure Message Transmission primitive in Section 3.2. This allows any two given parties to exchange securely a message s and, specifically, the protocol either aborts while detecting a corrupted party or it provides an alive receiver with the correct message of a sender, effectively creating a publicly detectable private message functionality. In other words, either the message is delivered (keeping its privacy) or it can be publicly detected which player failed/is corrupted.
With that idea we practically overcome the problem of omission corruptions and, thus, we could use any MPC protocol for active corruption in general adversaries to accomplish the rest of our task. We present the necessary tools and building blocks to do that in Section 3.3.
Next, we tackle one of the final problems, namely securely computing the gates in Section 3.4, with multiplication being the main difficulty while addition is pretty straight forward. There we present their functionalities and how we can implement them in our setting.

Finally, after establishing that, we will be able to provide a tight characterization of the perfectly secure MPC landscape (in terms of both feasibility and impossibility) in the remainder of the section. We compose all of our blocks and tools together in Section 3.5 to present our full MPC protocol and in Section 3.5 we prove that our conditions are also necessary, i.e. tight.
As a side note, we remind here that our MPC assumes that the parties can broadcast messages (elements from an appropriate arithmetic field $\mathbb{F}$). As we can easily see, our MPC condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ implies $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ which means that we can use *Broadcast* for this purpose[10].

[10] As discussed in the introduction, we can trivially turn binary *Broadcast* to a string *Broadcast* by invoking it for each bit of the string.

3.1 The ineffectiveness of player elimination

Dealing with omission corruptions can be quite difficult. So far, the only known way to accomplish that in information-theoretic MPC [ZHM09] is via *player elimination* (*cf.* Section 1.3). Recall that, player elimination is used in the threshold adversaries setting, and the idea is that if a set of parties with sufficiently many faulty parties is publicly identified—i.e., so that deleting all the detected parties does not destroy the corruption threshold—then under the condition that all the detected parties have had their chance to share their input, we can eliminate all of them, and have the remaining parties complete the computation (and later inform the eliminated parties of their outputs).

In the following we show that player elimination cannot be used in the general adversary setting, even for conditions that would seem natural to allow it. This will motivate us to focus on what we term *strong player elimination*, where if a set of parties is detected, we have a guarantee that they are *all* corrupted.

In more detail, we will provide a counter-example to using player elimination in the general adversary setting with omissions. Surprisingly, this specific example would normally work in the threshold model. Here, we show that a player set that respects the simple security condition

$$\forall Z_i, Z_j \in \mathcal{Z} : \Omega_i \cup \Omega_j \neq \mathcal{P}$$

fails to do so once we remove a set of two publicly detected players, where the guarantee is that at least one of them is (omission) corrupted. Let us assume that the adversary structure $\mathcal{Z}$ is the following, as presented in Table 3:
$Z_1 = \Omega_1 = \{p_1, p_3\}$, $Z_2 = \Omega_2 = \{p_1, p_4\}$, $Z_3 = \Omega_3 = \{p_2\}$.
Assuming that the players p_1, p_2 are the pair that has the disagreement, meaning that one of them is corrupted, if the weak player elimination was true, we could eliminate both of them and obtain the structure $\mathcal{Z}$: $Z_1 = \Omega_1 = \{p_3\}$, $Z_2 = \Omega_2 = \{p_4\}$, $Z_3 = \Omega_3 = \{\emptyset\}$. However, the residual player set consists only of p_3, p_4, meaning that $\Omega_1 \cup \Omega_2 = \mathcal{P}$, which proves our argument that weak player elimination is not working with general adversaries, because the security condition is not preserved.

	p_1	p_2	p_3	p_4
Z_1	ω		ω	
Z_2	ω			ω
Z_3		ω		

Table 3. The classes Z_1, Z_2, Z_3 that show the ineffectiveness of player elimination for general adversaries.

We note in passing that it is straight-forward to extend the above simple counter-example to other natural conditions on the adversary structure, e.g., $\forall Z_i, Z_j, Z_k \in \mathcal{Z} : \Omega_i \cup \Omega_j \cup \Omega_k \neq \mathcal{P}$ by adding another party that helps the three sets cover the player set. In fact, it is unclear which condition, if any, would allow us to use this technique.

On the other hand, due to the symmetric nature of omission corruption (where a faulty sender or receiver can have the same effect) it is unclear how one can derive a protocol with the stronger detection guarantees, which would allow for strong player elimination. To tackle this problem we came up with a new protocol idea which manages to establish a Secure Message Transmission primitive under the new condition $C^{(A,\Omega)}_{SMT}(\mathcal{P}, \mathcal{Z})$. In the following we prove that this condition together with $C^{(A,\Omega)}_{BA}(\mathcal{P}, \mathcal{Z})$, is both sufficient and necessary for MPC giving us a tight bound for the adversary structure.

3.2 Detectable Secure Message Transmission

Functionality $\mathcal{F}_{detSMT}(\mathcal{P}, \mathcal{Z}, P_s, P_r, s)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- In round $\rho = 0$: Initially, set the output value m_{out} to $\perp$.
  Upon receiving a message (input, sid, s) from the sender P_s (or the adversary, if the sender is actively corrupted), set $m_{out} = s$ and send (leakage, sid, $P_s, l(s) = |s|$) to the adversary.
- In rounds $\rho = 1$ to 7: Do nothing.
- At any point in time:
  - Upon receiving (adv-omit-output, P_r) from the adversary, if $P_r \in \Omega^\star$, set $m_{out} = \perp$ and output (omission, P_r) to P_r.

- Upon receiving (inform omission, p_b) from the adversary for some $p_b \in \mathcal{P}$, if $p_b \in \Omega^\star$, output (omission, p_b) to p_b and abort with $B = \{p_b\}$.
- Otherwise, discard the message.

– In round $\rho = 8$: Upon receiving (fetch-output, sid) from P_r, send (output, sid, m_{out}) to P_r and (fetch-output, sid, P_r) to the adversary.

The first step towards our MPC protocol is to enable any pair of parties with a sender P_s and a receiver P_r to exchange a message s securely, i.e. with the privacy and correctness of the message preserved. Furthermore, we want to accomplish that in a publicly detectable way, meaning that the protocol either succeeds or it aborts having detected a corrupted party.

The functionality that captures our goal is presented above. It starts by taking some input s from the designated sender. The adversary can input any value his chooses if the sender is actively corrupted.
Then this value is forwarded to P_r. Meanwhile, it allows the adversary to affect the output if the sender or receiver are corrupted by in a detectable way. Also, he could cause an abort, but at the cost of making publicly known the identity of a corrupted party.

Our protocol that realizes this functionality works in a way that resembles *FixReceive*, as discussed in Section 2.2; The sender sends to all players and at the end all players forward the message to the receiver. However, the difference is that now we have a reliable broadcast primitive and the players can use it complain if they do not receive a message they were expecting. Also, we also ensure that the privacy of the message is maintained, in contrast to *FixReceive*, which was done in public communication. This is done by the use of a secret sharing scheme. The sharing is characterized by the *sharing specification* $\mathcal{S}$, according to which the shares of the message to be kept secret are distributed to the players, effectively stopping the adversary from holding all shares. In our case we will be using a sum sharing, i.e. the secret value s is split in summands $s_1, \ldots, s_m$ with $\sum_{i=1}^{m} s_i = s$, where m is the size of the sharing specification $|\mathcal{S}|$.
For each player p_j we call the vector of summands in his possession $\langle s \rangle_j = (s_{j_1}, \ldots, s_{j_k})$ as p_j's *share* of s. The complete vector of all shares is denoted as $\langle s \rangle = (\langle s \rangle_1, \ldots, \langle s \rangle_n)$ and is called a *sharing* of s. The vector of summands of s is denoted as $[s] = (s_1, \ldots, s_m)$ and their sum is equal to s. We, also, say that such a sharing $\langle s \rangle$ is a consistent sharing of s according to $(\mathcal{P}, \mathcal{S})$, if for each $S_k \in \mathcal{S}$ all (correct) players in S_k have the same view on s_k and $s = \sum_{k=1}^{m} s_k$.
Our selection for $\mathcal{S}$ will be the natural sharing specification $\mathcal{S}_\mathcal{Z}$ associated with $\mathcal{Z}$, i.e. $(S_1, \ldots, S_m) = (\mathcal{P} \setminus A_1, \ldots, \mathcal{P} \setminus A_m)$, where $m = |\mathcal{Z}|$, so that for each corruptible class Z_i all the players not included in A_i for that class will receive the share s_i. This way the adversary never obtains all summands.

Using that secret sharing scheme, P_s creates a sharing of his message s and sends each part s_q to the compliment S_q of A_q. This process is done for each $q = 1, \ldots, m$. Then the players who did not receive it can complain through broadcast. Additionally, an extra round of cross checking and relay is added, during which all parties in S_q send to one another the values they received from P_s. Again, complaints are raised and the players try to see which classes of the adversary structure fit their view.
Finally, once the complaints are over, all players send their vector of received values to the designated receiver of the message, P_r.

To make the counting of the rounds easier, we will assume that all rounds are *Broadcast rounds* (1 BCR). This means that we give enough time from now on to what we consider a (BC) round, to allow the parties to perform the Broadcast protocol. This assumption sacrifices peer-to-peer rounds during which nothing is done [11] while the parties wait for *Broadcast* to conclude, but this does not affect our study as we are in the perfect security model. Looking ahead, once our SMT primitive is established, we will only use those two "channels" to send messages, i.e. every message will be either a Broadcast message or a SMT message. As a result, we will start counting our rounds similarly, either as Broadcast rounds (1 BCR) or SMT rounds (1 STR).

[11] This creates a linear in the number of parties overhead of rounds.

Protocol $DetSMT(\mathcal{P}, \mathcal{Z}, \mathcal{S}, P_s, P_r, s)$

- In round $\rho = 0$: The sender P_s creates a random sharing $[s] = (s_1, s_2, \ldots, s_m)$ for s according to the sharing specification $\mathcal{S}$ by randomly selecting $s_2, \ldots, s_m$ and setting $s_1 = s - \sum_{i=2}^{m} s_i$, where $m = |\mathcal{S}|$. If a player fails to share his input, a default value is used.
- Perform the following phases (in parallel) for $q = 1, \ldots, m$:

  1^{st} Phase (spread). In round $\rho = 1$: P_s sends s_q to every $p_k \in S_q \cup \{P_r\}$. Each p_k denotes the received summand as $s_q^{(k)}$ ($s_q^{(k)} =$ "$\mathsf{n/v}$" if p_k received none or an invalid value for s_q).

  2^{nd} Phase (complaints). In round $\rho = 2$: All players broadcast a special message OK if they received the messages they were anticipating or a complaint NOK if a message was dropped.
  We denote by $P^{(OK)}$ and $P^{(NOK)}$ the sets of players who broadcast OK and NOK respectively.
  - **If** there exists some $Z_k = (A_k, \Omega_k) \in \mathcal{Z}$ such that $\Omega_k \supseteq P^{(NOK)}$ **then** proceed to the next phase.
  - **Else** the protocol aborts with the set $B = \{P_s\}$ (i.e. $P_s \in \Omega^\star$).

  3^{rd} Phase (cross-check and relay). Let $S_q = \{p_{q_1}, p_{q_2}, \ldots, p_{q_\lambda}\}$.
  - In round $\rho = 3$: Every $p_{q_i} \in S_q$ sends the value $s_q^{(q_i)}$ to every $p_{q_j} \in S_q \cup \{P_r\}$, who denotes the received value as $s_q^{(q_i,q_j)}$.
  - In round $\rho = 4$: For each p_k, all players broadcast a special message OK_k if they received the message $s_q^{(k)}$ they were anticipating from p_k or a complaint NOK_k if the message was dropped. We denote by $P_k^{(OK)}$ and $P_k^{(NOK)}$ the sets of players who broadcast OK_k and NOK_k, respectively.
  - **If** at any point some $p_{q_i} \in S_q \cup \{P_r\}$ receives contradicting values $s_q^{(q_j,q_i)}, s_q^{(q_\ell,q_i)}$ with $s_q^{(q_j,q_i)} \neq \perp$ and $s_q^{(q_\ell,q_i)} \notin \{s_q^{(q_j,q_i)}, \perp\}$ for s_q, **then** p_{q_i} broadcasts CONTRAST.
  - **If** any $p_{q_i} \in S_q$ broadcasts CONTRAST or for some $p_{q_j} \in S_q \cup \{P_s\}$ we have that:
    $p_{q_j} \notin \Omega_q \ \wedge \ P_{q_j}^{(NOK)} \nsubseteq \Omega_q$ **then** P_s broadcasts s_q and P_r adopts this value (the class S_q is not justifying the complaints).
  - **Else** all parties consider the classes
    $\mathcal{Z}_q := \{Z_j \in \mathcal{Z} \text{ s.t. } \forall\, p_{q_i} \in S_q \cup \{P_s\}\colon (p_{q_i} \in \Omega_j \vee P_{q_i}^{(NOK)} \subseteq \Omega_j)\}$. **If** for some party $p_{q_i} \in S_q$ for all $Z_j = (A_j, \Omega_j) \in \mathcal{Z}_q$ we have $p_{q_i} \in \Omega_j$, **then** the protocol aborts with $B = \{p_{q_i}\}$.
    If $Z_q \notin \mathcal{Z}_q$ **then** P_s broadcasts s_q and P_r adopts this value.
    If there exists no $Z_k \in \mathcal{Z}_q$ such that the corresponding Ω_k covers $P^{(NOK)}$ **then** the protocol aborts with $B = \{P_s\}$.

  4^{th} Phase (output of P_r for each $S_q \in \mathcal{S}$). In round $\rho = 5$: Each player $p_k \in S_q$ sends its unique received value $s_q^{(k)}$ over all $s_q^{(q_i,k)}$ (or "$\mathsf{n/v}$" if p_k has received no value for s_q) to P_r.
  - **If** there exists no $Z = (A, \Omega) \in \mathcal{Z}_q : P_r^{(\perp)} \subseteq \Omega$, where $P_r^{(\perp)} := \{p_k : s_q^{(k)} = \perp\}$, **then** P_r becomes a zombie and outputs ($\mathsf{omission}$, P_r, $\perp$).
  - **If** P_r receives contradicting values $s_q^{(q_j)} \neq \perp$ and $s_q^{(q_\ell)} \notin \{s_q^{(q_j)}, \perp\}$ for s_q, **then** P_r broadcasts CONTRAST. Upon that, the sender P_s broadcasts s_q and P_r adopts this value (in rounds $\rho = 6, 7$).
  - **If** P_r has received no (valid) value at all for s_q from any $p_k \in S_q$ OR if for all $Z_j \in \mathcal{Z}_j$ we have that $A_j \supseteq P_r^{(s_q)}$ **then** P_r broadcasts a special message REQUEST, upon which the sender P_s broadcasts s_q and P_r adopts this value (in rounds $\rho = 6, 7$).
  - **Else,** P_r sets $s_q^{(r)}$ as the unique value in the set $\{s_q^{(q_1,r)}, s_q^{(q_2,r)}, \ldots, s_q^{(q_\lambda,r)}\}$ of received values (and wait until round 7 finishes).
- In round $\rho = 8$: Finally, **if** there exists some justifying for all q class $Z_j = (A_j, \Omega_j) \in \mathcal{Z}_j$ such that for every $S_q \in \mathcal{S}$ we have some unique $s_q^{(r)} \in \mathbb{F}$ for $q = 1, \ldots, m$, **then** P_r outputs $s = \sum_{q=1}^{m} s_q^{(r)}$;
  else P_r outputs $s = 0$ (P_s is actively corrupted).

- **Protocol-Wide Detection:** The following procedure is followed if at any point in the protocol a party broadcasts an invalid value or $\perp$: **If** any player p_f fails to broadcast a valid message when expected to do so, **then** the protocol immediately aborts with $B = \{p_f\}$ (if there is more than one such players, then take the one with the smallest index.).

Lemma 6. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, the protocol* DetSMT *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{detSMT}$.*

Proof (Sketch). First we need to state the claim about the input/output properties of the protocol.

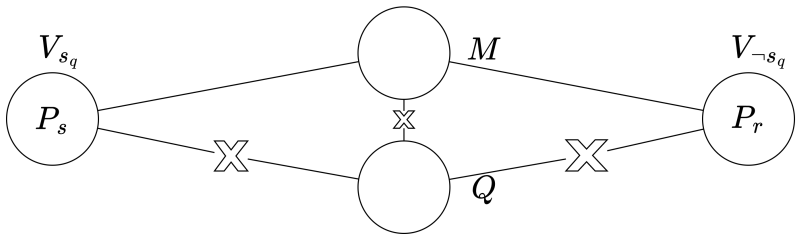

Fig. 5. An SMT execution with sender in V_{s_q} and receiver in $V_{\neg s_q}$, where any direct channel between them is blocked. No message is received from a channel between a player in V_{s_q} and a player in $V_{\neg s_q}$. Same for sets M and Q. All parties in K possess a value s_q and parties is L a value $\widehat{s_q}$.

Claim 4. *If the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P},\mathcal{Z})$ (see Eq. 6) holds, the protocol* DetSMT *satisfies the following. Either the protocol aborts with some set B of corrupted parties or it terminates and we have the properties: a) If the receiver P_r is alive at the end of the protocol then he outputs a value $s_p \in \mathbb{F}$ where $s_p = s$ unless $P_s \in A^\star$. b) Also, P_r might become a zombie only if he is omission-corrupted. c) Furthermore, no information on s is leaked to the adversary.*

Proof. First claim: If the protocol aborts, the identified set B contains only omission-corrupted parties. There are three cases where the protocol aborts. The first case is if the check during phase 2 fails and the protocol aborts with $B = \{P_s\}$. This means that there is no class Z_k such that all complaints are coming from players in Ω_k. Towards contradiction, we assume that the sender is correct. This implies that all the messages that P_s sends are correctly received by all correct parties, hence all complaints, i.e. all players in $P^{(NOK)}$ are in $\Omega^\star$, giving us that $P^{(NOK)} \subseteq \Omega^\star$. However, this contradicts the fact that no Ω_k covers $P^{(NOK)}$, meaning that P_s is indeed in $\Omega^\star$.
The second case is at the end of phase 3, where the protocol aborts with $B = \{p_{q_i}\}$ if some party p_{q_i} belongs in Ω_j for all $Z_j \in \mathcal{Z}_q$. Since $\mathcal{Z}_q$ contains all classes that justify the view of the players and the chosen class $Z^\star$ is also justifying that view (in reality it is causing it), we get that $Z^\star \in \mathcal{Z}_q$. Thus $p_{q_i} \in \Omega^\star$.
The third and final case that that protocol aborts is with a set $B = \{p_f\}$ from the protocol-wide detection. In that case, p_f failed to broadcast a valid message. By the properties of Broadcast, this directly implies that $p_f \in \Omega^\star$.

Second Claim: A player (specifically P_r) becomes zombie only if he is omission-corrupted. According to the protocol, the only instance of a player becoming a zombie is at phase 4 when P_r checks the $\perp$ he received. If there exists no class from the justifying set $\mathcal{Z}_j$ that covers $P_r^{(\perp)}$, meaning that the set of players that did not send a value to P_r is not covered by Ω, we can deduce that P_r received a $\perp$ due to his own fault, hence he is not correct. As a result, P_r becomes a zombie only if he is omission-corrupted.

Third Claim: If is P_r is alive and P_r outputs some $y \neq \perp$ then $y = s$ (unless $P_s \in \Omega^\star$).
Towards contradiction, let us assume that for some set S_q, the receiver P_r receives and adopts some $\widehat{s_q} \neq s_q$ (by definition of the protocol this means that through the protocol for this S_q and the corresponding share s_q the receiver P_r has only received $\widehat{s_q}$ and not the correct summand s_q at any round. Otherwise, he would have CONTRASTed). In this case we will show that the player set should be split as shown in Figure 5, which is essentially similar to the counterexample for $C^{(A,\Omega)}_{SMT}(\mathcal{P},\mathcal{Z})$ of Lemma 7. Specifically, P_s will belong in a set V_{s_q} and P_r in a set $V_{\neg s_q}$ with all communication cut between them, i.e. no message is received from a channel between a player in V_{s_q} and a player in $V_{\neg s_q}$. Similarly for the sets M and Q. This will then lead us to a contradiction of $C^{(A,\Omega)}_{SMT}(\mathcal{P},\mathcal{Z})$.

Let us define the sets V_{s_q} as the set of (non-actively corrupted) parties in S_q that at some point saw s_q and never something different and $V_{\neg s_q}$ the rest of the non-actively corrupted parties in S_q. Formally, we write:

$V_{s_q} := \{p_k : p_k \in S_q \setminus A^\star\}$ where p_k that at some point saw s_q and never something different and
$V_{\neg s_q} := \{p_k : p_k \in S_q \setminus A^\star\}$ and p_k never saw s_q throughout the protocol (either saw $\widehat{s_q}$ or no value for s_q). Since no CONTRAST was broadcast at any point, it means that no $p_k \in S_q \setminus A^\star$ received both s_q and $\widehat{s_q}$, i.e. $V_{s_q} \cap V_{\neg s_q} = \emptyset$. By definition we also have that $P_s \in V_{s_q}$ and $P_r \in V_{\neg s_q}$. Also, it is easy to see that every communication between those two sets is blocked, because in the opposite case some player able to communicate with the other set would have broadcast CONTRAST. In particular, we note that P_r must have received $\perp$ from all players in V_{s_q}. To explain this situation we must have either $V_{s_q} \subseteq \Omega^\star$ or $V_{\neg s_q} \subseteq \Omega^\star$. The other two sets that exist in the player set are the set $Q = A_q$ of all parties not participating in this iteration for q by definition and finally the set M of actively corrupted parties. In the specific case that we are studying, all communication from Q to $V_{s_q}, V_{\neg s_q}$ is blocked.
From here we deduce that there are only two types of justifying classes, which are public and known to all due to the public nature of the complaints.

- The justifying class of type 1) are the Z_k with Ω_k such that it covers the set $V_{\neg s_q}$ and P_r (i.e. $\Omega_k \supseteq V_{\neg s_q}$).
- The justifying class of type 2) are the Z_m with Ω_m such that it covers the set V_{s_q} and P_s (i.e. $\Omega_m \supseteq V_{s_q}$).

Those classes can be divided further in two whether their respective A is the set $Q = A_q$ or no. Let us denote the former as type i) and the latter as type ii).
Only those types of classes survive the elimination of impossible classes during the protocol and are considered justifying.

Now we observe the following.

- If no type 1) class exists then P_s becomes a zombie.
- If no type 2) class exists then P_r becomes a zombie, because V_{s_q} is the set of players that P_r received $\perp$ from.
- If no type i) class exists then P_s broadcasts s_q (it would mean that Z_q is not justifying, hence the protocol would stop and broadcast s_q at the end of phase 3).

Hence we have the following surviving types of classes and only those, conditioned on the fact that no broadcast of s_q occurred.
$Z_{k_i} \in \mathcal{Z}_q := \{A_k = A_q \wedge V_{s_q} \subseteq \Omega_k\}$ of type 1.i),
$Z_{m_i} \in \mathcal{Z}_q := \{A_m = A_q \wedge V_{\neg s_q} \subseteq \Omega_m\}$ of type 2.i), and
$Z_m, Z_k \in \mathcal{Z}_q$ of type 1,2 respectively.

Now, with those surviving classes we must try to explain the view of the players in the case that we study. In particular, since a wrong value $\widehat{s_q}$ was created and forwarded to $V_{\neg s_q}$, it means that there exists a class $Z_b = (A_b, \Omega_b)$ that is either $Z_{k_{ii}}$ or $Z_{m_{ii}}$ of either type 1 or 2 with $A_b = M$. Those are the only two classes that we know (in the analysis of the protocol, the players do not know that) that can cause the case we are studying. This means that some $p_c \in M$ either created the value $\widehat{s_q}$ and relayed it to some p_ℓ who then forwarded it to P_r, or that p_c directly forwarded it to P_r. In other words, for any $s_q^{(i,j)} = \widehat{s_q}$ that P_r received we have that either $p_i \in M \subseteq A^\star$ or $p_j \in M \subseteq A^\star$ (we only care for the case where P_s is not malicious, so any $\widehat{s_q}$ messages are not his creation). At this point, using a player simulation argument, those sets $V_{s_q}, M, V_{\neg s_q}$ together with A_q create the same player set and corruptible classes as in the counter example in 3.2.

Let us deal with the first case, i.e. the adversary actually controls some $Z_b = (A_b, \Omega_b)$ that is of type 1), $Z_{k_{ii}}$. This gives us that

$$\Omega_b \supseteq V_{\neg s_q} \text{ and } A_b \supseteq M, \tag{31}$$

which dictates that only type i) class available would be some $Z_q = (A_q, \Omega_q)$ of type 1 with

$$\Omega_q \supseteq V_{\neg s_q} \text{ and } P_r \in \Omega_q, \tag{32}$$

Otherwise, if there existed some type 2.i) class, we would have that $\Omega_q \supseteq V_{s_q}$ and this would imply

$$A_q \cup \Omega_q \cup A_b \cup \Omega_b \supseteq A_q \cup V_{s_q} \cup M \cup V_{\neg s_q} = \mathcal{P}, \tag{33}$$

which contradicts $C^{(A,\Omega)}_{FIXR}(\mathcal{P},\mathcal{Z})$. Finally, in order to complete the problematic scenario which would confuse the receiver, we know that a type 2 class Z_ℓ must exist, with $\Omega_\ell \supseteq V_{s_q}$.
Since we cannot allow any two $\Omega \cup A$'s to cover the whole player set (would contradict $C^{(A,\Omega)}_{FIXR}(\mathcal{P},\mathcal{Z})$), A_ℓ cannot cover M due to the way that Z_q is built. So this case ends up containing the following three classes:
Z_b with $\Omega_b \supseteq V_{\neg s_q}$, $A_b \supseteq M$ and $P_r \in \Omega_b$.
Z_q with $\Omega_q \supseteq V_{\neg s_q}$, A_q and $P_r \in \Omega_q$
Z_ℓ with $\Omega_\ell \supseteq V_{s_q}$ and $P_s \in \Omega_\ell$.
However, this contradicts the condition $C^{(A,\Omega)}_{SMT}(\mathcal{P},\mathcal{Z})$, due to

$$\begin{aligned}\Omega_\ell \cup A_b \cup A_q \cup (\Omega_b \cap \Omega_q) \supseteq \\ V_{s_q} \cup M \cup A_q \cup (V_{\neg s_q}) = \mathcal{P},\end{aligned} \tag{34}$$

with $P_r \in \Omega_b$, $P_r \in \Omega_q$ and $P_s \in \Omega_\ell$, proving that this case cannot be occurring. The second and final case with Z_b being of type 2.i) is completely symmetrical and can be omitted.
From this contradiction we deduce that our claim is true, meaning that if P_r is alive and P_r outputs some $y \neq \perp$ then $y = s$ (unless $P_s \in \Omega^\star$).

Fourth Claim: No information is leaked to the adversary. Moving to proof of privacy, we will show that no information on s is leaked to the adversary. By construction of the natural sharing specification $\mathcal{S} = (\mathcal{P} \setminus A_1, \ldots, \mathcal{P} \setminus A_m)$ and the fact that each s_q is sent to the players in S_q and not A_q, we see that there exists one class $Z^\star = Z_c$ such that the adversary does not obtain the summand s_c in the spread phase.
There are four operations that convey information on s_q (indeed, all other communication consists of broadcasting complaint messages): (1) parties in S_q sending s_q to each other for consistency checks in Phase 2; (2),(3),(4) sender P_s broadcasting s_q as a result of a CONTRAST complaint in Phase 3 and 4, or due to $Z_q \notin \mathcal{Z}_q$ in Phase 3 or, finally, due to a REQUEST from P_r in Phase 4.
In case (1) no information is conveyed as the messages are between parties of S_q. This means that if any party in S_q is actively corrupted the adversary already knows s_q, otherwise he does not get to see any of these messages.
For cases (2), (3) and (4), we argue that privacy is kept even after some s_q is broadcast from P_s according to the protocol. Indeed, we will show that P_s broadcasts s_q only when the adversary already knows it, so no information is leaked.
In case (2) a player in $S_q \cup \{P_r\}$ receives contrasting values during Phase 3 or 4.If the sender was correct, then all values for s_q would be the same for all players. This means that since some p_k reached CONTRAST, either the sender is actively corrupted or some actively corrupted party in S_q relayed the contrasting value or p_k is actively corrupted. In all cases, the adversary has learned the summand s_q already, so broadcasting it does not compromise the privacy of s.
Case (3) occurs when the value of s_q is broadcast because Z_q is not in the set $\mathcal{Z}_q$ of justifying classes for s_q. Since by definition $Z^\star \in \mathcal{Z}_q$[12], this directly implies that $Z_q \neq Z^\star$. Hence, the adversary already knows s_q.
Case (4) is when P_r will broadcast REQUEST during the P_r output Phase 4, if no value at all is received. In that scenario, all the messages from the players in $P^{(OK)}$ (who have received a value) towards P_r must have been blocked. We will prove that this means that Z_q cannot be the class that the adversary selected, meaning that P_s can safely broadcast s_q. Let us assume towards contradiction that $Z_q = Z^\star$ (i.e. $A_q = A^\star$) and that P_r remains alive and gets no value at all. The sender P_s is either correct or incorrect.
If P_s is correct, then $P^{(NOK)} \subseteq \Omega^\star$ and together with $A_q = A^\star$ we get that no Ω_m can cover $P^{(OK)}$, in order not to violate $C^{(A,\Omega)}_{FIXR}(\mathcal{P},\mathcal{Z})$, stating that for any two $k, m : \Omega_k \cup \Omega_m \not\supseteq \mathcal{P}$. This means that some honest party exists in $P^{(OK)}$ and after phase 3 is completed all parties that are not omission-corrupted will get some value and will forward it to P_r, who either receives it or becomes a zombie (since $\Omega_m \not\supseteq P^{(OK)}$ for all Ω_m).
If P_s is not correct, then from the check during phase 2 we know that there exists some Z_k such that

[12] Remember that $\mathcal{Z}_q$ is the set of classes that justify the view of the parties and Z_q has created this view, so it obviously justifies them.

$\Omega_k \supseteq P^{(NOK)}$ (otherwise the protocol aborts, having detected P_s). Given that $A_q = A^\star$, from $C^{(A,\Omega)}_{FIXR}(\mathcal{P}, \mathcal{Z})$ we get that $\Omega^\star \not\supseteq P^{(OK)}$, meaning that some honest party receives the correct value from P_s. Due to the relay and cross-check, at this point we are essentially back in the previous case, where the "sender" is correct, as this honest party will forward s_q to all other parties in $S_q \cup \{P_s, P_r\}$. This grants us that again P_r will either receive some value or become zombie. As a conclusion, if $Z_q = Z^\star$ we are certain that an alive P_r will always receive some value. By this, we are allowed to safely ask for the broadcast of s_q if for some q we have that P_r received no value at all. □

Now we are ready to give a sketch of the simulation proof. The simulator observes the real world execution of the protocol. If the protocol aborts, according to the properties of Claim 4, we are guaranteed that it does so while identifying some corrupted party $p_c \in B$ (sender included). As such, the simulator can create the same effect in the ideal world by sending (inform omission, p_c) to the functionality.
If the receiver becomes zombie from the same claim we are granted that he is indeed omission-corrupted. As a result, the simulator sends (adv-omit-output, P_r) to the functionality to create the same effect.
If the receiver remains alive we know from the claim that he will output a value s_p, where $s_p = s$ unless $P_s \in A^\star$. We will show that this is indistinguishable from the ideal world. To do it, we break down the simulation for each $S_q \in \mathcal{S}$, where s_q is sent to S_q and $\sum_{q=1}^{m} s_q = s$.

- If the sender is correct we have two cases:
  - If the adversary controls no party in $S_q \cup \{P_r\}$ the simulator just runs the protocol, as the adversary learns no information about s_q.
  - If the adversary controls some party in $S_q \cup \{P_r\}$ the simulator samples a uniformly random r_q for s_q and forwards it to the adversary. This way the r_q variables follow the same distribution with the s_q.
- If the sender is actively corrupted, the simulator must use the value for s_q that the real adversary used, in order to make the two executions indistinguishable. Once the simulator learns this s_q value, he can send (input, sid, s_q) to the functionality to set the value of the summand to the desired one (he can only do that because P_s is actively corrupted).
- If the receiver is actively corrupted, the simulator makes certain that the sum of all s_q is equal to s by setting the last value that he has to sample accordingly.

Aside from that, everything is deterministic with no randomness affecting the protocol and there is no private input that the simulator needs to extract. As a result, the simulator can simply run the protocol.

We will examine the following two cases, namely if the adversary controls some party $p_c \in S_q$ or not.

In the first case where the adversary has no such control and does not learn s_q, the simulator just replicates the broadcast of complaints as he observes them in the real world. This is sufficient because the adversary sees nothing except from the public complaints, which he can choose.

In the second case where the adversary already knows s_q, our argument about the real world execution of the protocol and the ideal world being indistinguishable by any environment is based on the fact that the adversary only learns a fraction of the summands and not all of them. Specifically, initially the simulator observes if a party p_b becomes zombie in the real world. From the properties of Claim 4 we know that p_b is indeed omission-corrupted which allows the simulator to send (inform omission, p_b) to the functionality, creating the same effect. Then, the simulator samples a uniformly random value and forwards it to the adversary for each summand that the adversary receives in the real world. Since they are random and the adversary does not have access to all of them by construction of the adversary structure, the effect is the same as in the ideal world, where uniformly random values are handed to the adversary.
To complete the argument that the two worlds are indistinguishable, the simulator also takes advantage of the fact that the complaints are public and the actions of the real world adversary can be detected according to Claim 4, together with the fact that each summand is uniformly random and independent of the rest, subject to their sum being set. From the same claim we also get that an alive P_r will output the correct value s_q in the real world if the protocol does not abort. In the same manner, the functionality delivers the correct s_q to an alive P_r in the ideal world. As a result, the simulator is able to send the appropriate messages to $\mathcal{F}_{detSMT}$ and the adversary to create the same view in the ideal world and in the real world execution of the protocol. □

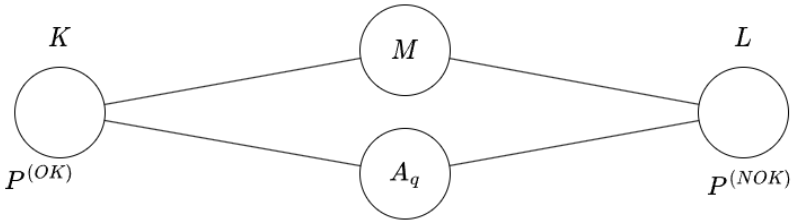

Fig. 6. Impossibility of SMT with sender in K and receiver in L, when there exists no direct channel between them.

Necessity of SMT condition. In this section, we prove that the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_1, p_2)$ is actually necessary for the $\mathcal{F}_{DetSMT}$ functionality between a sender p_1 and a receiver p_2, making our result tight (both sufficient and necessary). To show the impossibility of SMT when the condition does not hold, we distinguish the following two cases. In the first case the sender is omission-corrupted and the receiver is honest (should output "$\mathsf{n/v}$"). In the second case the sender is honest and the receiver is omission-corrupted (should output $\perp$ – become a zombie). Note that consistently with our previous functionalities (and the natural MPC functionality), the above two scenarios have distinct outputs.

Indeed, in an MPC evaluation, a correct receiver should never output $\perp$ when the sender is correct; and if the receiver is omission-corrupted and correct (but not actively corrupted), then he should never output a value other than the one that the sender intended to send (or a special value in case it is clear the sender failed to send his value). More formally, this allows to explicitly distinguish the scenario $p_1 \in \Omega^\star$, $p_2 \in \mathcal{H}$ with output "$\mathsf{n/v}$" from the scenario $p_1 \in \mathcal{H}, p_2 \in \Omega^\star$ with output $\perp$ from p_2.

	p_1	p_2	p_3	p_4
Z_1	ω		α	
Z_2	ω			α
Z_3		ω		

Table 4. p_1 wants to securely send a message to p_2.

Our impossibility proof shows that in order to ensure that the above scenarios can be faithfully followed, the parties (sender and receiver) need to reveal information about the transmitted message to the adversary which contradicts the security (specifically privacy) of SMT.

Lemma 7. *If the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_1, p_2)$ does not hold, then the functionality $\mathcal{F}_{DetSMT}(\mathcal{P}, \mathcal{Z}, p_1, p_2, m)$ cannot be securely realized.*

Proof. Let us assume towards contradiction that there exist classes Z_1, Z_2, Z_3 with

$$A_1 \cup A_2 \cup \Omega_3 \cup (\Omega_1 \cap \Omega_2) = \mathcal{P} \qquad \text{AND} \qquad p_1 \in \Omega_1 \cap \Omega_2 \wedge p_2 \in \Omega_3, \tag{35}$$

then we will prove that we cannot perform a secure message transmission from p_1 to p_2.
We focus on the smallest instance violating this condition, consisting of 4 players p_1, p_2, p_3, p_4, with the following adversary structure: $(A_1, \Omega_1) = (\{p_3\}, \{p_1, p_3\})$, $(A_2, \Omega_2) = (\{p_4\}, \{p_1, p_4\})$, and finally $(A_3, \Omega_3) = (\emptyset, \{p_2\})$, as shown in Table 4. Using a standard player simulation argument [HM97], the proof can be directly extended to the general n-party case. In fact, we will prove that a restriction of the above adversary is not tolerable, which implies that the above is also not tolerable. This restriction is as seen in the Figure 6):
Note that in all three classes the adversary controls one of p_1 or p_2, hence the adversary can always block the communication between p_1 and p_2 through their direct channel, meaning that all communication between them has to rely on p_3 or p_4.

To prove our impossibility, let us assume that p_1 wants to send some message $m \in \mathbb{F}(\neq$ "$\mathsf{n/v}$") to p_2 and let's assume that there exists a protocol Π that securely accomplishes that. We consider the following three scenarios:

– Scenario 1: The adversary corrupts Z_1. This means that communication between p_1 and p_2 is blocked because the sender is omission-corrupted. Also, p_3 is actively corrupted, so the adversary has full control

over what messages p_3 sends to p_2. Since p_2 is honest, he should output either the correct value m or "$\mathsf{n/v}$" to indicate that p_1 is faulty.
- Scenario 2: The adversary corrupts Z_2. As before, communication between p_1 and p_2 due to p_1's fault. Also, p_4 is actively corrupted, so the adversary has full control over what messages p_4 sends to p_2. Since p_2 is honest, he should output either the correct value m or "$\mathsf{n/v}$" to indicate that p_1 is faulty.
- Scenario 3: The adversary corrupts Z_3. This time, communication between p_1 and p_2 due to p_2's fault, as only p_2 is (omission) corrupted. Since the sender is correct, p_2 should output either the correct value m or $\perp$ to indicate that the problem lies with the him, the receiver.

Since the adversary can perfectly replicate an honest execution of the protocol by his actively corrupted party, all three scenarios are perfectly indistinguishable from the point of view of p_2.
As a result, it is necessary that the output of p_2 in all three scenarios is the same value, in common in all three cases, meaning that p_2 should output $v = m$.

Let us now focus on the first 2 scenarios and see how the necessity of the output being $v = m$ creates a problem. In the first scenario, the adversary has full control over p_3 and this enables him to create any transcript that he desires and gives him full information on all messages that go through p_3. As a protocol that securely realizes SMT should leak no information to the adversary, we deduce that any possible transcript of the message history between p_3 and p_2, *denoted as* $T_{3,2}$, should contain no information at all about m. In a similar way, $T_{4,2}$ should also be completely independent of m.

This leads us to the following observation. For any given m' and for every $T_{4,2}$, there exists some $T'_{3,2}$ such that $\Pi_2(T'_{3,2}, T_{4,2}) = m'$, i.e. p_2 outputs m' given those transcripts. To see why this is true, if for any given m' and $T_{4,2}$, there existed no $T'_{3,2}$ such that $\Pi(T'_{3,2}, T_{4,2}) = m'$, then the adversary could try all possible combinations of transcripts and conclude that $m \neq m'$. This means that this transcript $T_{4,2}$ would leak information about the message, specifically that it is not equal to m'.

Now, to complete our argument, let us assume that we are in the first scenario, where the adversary controls p_3, and p_1 wants to send m to p_2. Let the transcript of p_4 be some $T_{4,2}$, generated honestly by following Π. In the case where the adversary replaces the program of p_3 with a protocol that generates random messages towards p_2, there is the chance (which could be negligible, but still strictly positive) that the transcript $T'_{3,2}$ is generated.

This means that p_2 will output $\Pi_2(T'_{3,2}, T_{4,2}) = m'$, which is the wrong value, not the one that p_1 intended to send, contradicting our assumption that there exists some Π that securely realizes $\mathcal{F}_{DetSMT}$. □

3.3 Building Blocks and Tools for MPC

Having established the *DetSMT* primitive to replace the network of point-to-point channels for the communication, we will now carry on with the construction of an MPC protocol by following the classic idea of creating a secure MPC protocol in the presence of a general adversary using only active corruption. Given any arithmetic circuit C—recall that this is a complete model of computation—the protocol evaluates the circuit in a gate-by-gate fashion, where the invariant is that the inputs and outputs of each gate of C are kept secret shared, see below, so that no information leaks to the adversary. Importantly, the protocols that process each gate, which we construct, might abort; however, when this happens: (1) no information leaks to the adversary, and (2) a corrupted party p is identified. This means that we can exclude p, and reset the computation without it. As we prove, the relevant sufficient condition, $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$, is preserved when eliminating such a corrupted party, which will ensure security in the reduced setting. As soon as an iteration of the above processing of the gates of C terminates without an abort–which is bound to happen after at most n resets—we invoke a reconstruction protocol to have every party (still alive) receive the output. We note that without loss of generality, we assume that the function which is computed by C has one public output. Using standard techniques, we can use a protocol for any such function to compute functions with multiple and/or private outputs [LP09].

In the following, we start by describing and proving the security of sub-protocols that are used as building blocks and then describe how these can be stitched together in an MPC protocol.

Heartbeat. A very important part of our results is based on the fact that if the adversary blocks enough messages addressed to a player to make him reach a wrong conclusion, the player could be able to perceive this loss of messages. Then, he could step down from the calculation by becoming a zombie, as he is (omission) corrupted. The functionality $\mathcal{F}_{Hb}$ is taking as input by the player a bit $b = 1$ indicating that he is alive. The adversary is able make a player in $\Omega^\star$ aware of his omission status, effectively setting $b = 0$. Then this value is communicated to all parties.The functionality is provided in detail below.

Functionality $\mathcal{F}_{Hb}(\mathcal{P}, \mathcal{Z}, z, P_s)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- In round $\rho = 0$:
  - Initially, set the input value z and the output values $y_1, \ldots, y_n$ to $\perp$.
  - Upon receiving a message (input, sid, v) from P_s (or the adversary, if the player is actively corrupted), set $z = v$, $v \in \{0, 1\}$, and send (leakage, sid, P_s, v) to the adversary.
  - Upon receiving (inform omission, P_s) from the adversary, if $P_s \in \Omega^\star$, set $z = 0$ and output (omission, P_s) to P_s.
- In round $\rho = 1BCR$:
  - If $z = 1$, $\mathcal{F}_{Hb}$ sets $(y_1, \ldots, y_n) = (1, \ldots, 1)$.
    Otherwise, it sets $(y_1, \ldots, y_n) = (0, \ldots, 0)$.
  - Upon receiving (fetch-output, sid) from some $p_i \in \mathcal{P}$, send (output, sid, y_i) to all p_i and (fetch-output, sid, p_i) to the adversary.

We implement this through the broadcast of the bit b by the player. If $b = 1$ then all agree that the player is alive. Otherwise, if a player fails to broadcast this bit to other players or broadcasts $b = 0$ it becomes apparent to all that he is a zombie, as he is corrupted. According to the output of broadcast everyone agrees whether p is alive or not.
It should be noted that omission-corrupted players who have not yet detected their problem can learn that they are zombies from the output of the consensus protocol.

Protocol *Heartbeat*$(\mathcal{P}, \mathcal{Z}, p)$

- In round $\rho = 0$: If p is alive (not a zombie), p sends a bit $b = 1$ to every $p_j \in \mathcal{P}$, by invoking *Broadcast*$(\mathcal{P}, \mathcal{Z}, p, b)$.
- In round $\rho = 1BCR$: Every $p_j \in \mathcal{P}$ sets $b_j := 1$ if the output of *Broadcast* was a 1-bit and $b_j := 0$ otherwise. Every $p_j \in \mathcal{P}$ outputs "alive" when $b_j = 1$ and "zombie" otherwise.
  Additionally, if the output is 0, p becomes a zombie.

Lemma 8. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 7) holds, the protocol* Heartbeat *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{Hb}$*.*

Proof. The essence of the functionality above is a broadcast of an "alive indicator" bit. As such, the proof of the lemma is less involved compared to the previous lemmas and we will only include a sketch of the intuition of the simulator. This will follow after we state and prove the corresponding properties of the protocol in the claim below.

Claim. If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 7) holds, the protocol *Heartbeat* has the following properties: *(consistency)* All (correct) players agree on the output of the protocol. *(correctness)* If p has become a zombie before the invocation of *Heartbeat*, then every (correct) player learns it and outputs "zombie". Finally, a player p might become a zombie only if he is omission-corrupted.

Proof. To begin with, the consistency is granted directly from the consistency property of broadcast 2.7 on the values b.

Next, for the correctness, it is clear that if a player has noticed before the invocation of *Heartbeat* that he is omission-corrupted, then he will follow the protocol instructions and broadcast the value $b = 0$ to all players (or equivalently when it is time for him to carry out the protocol he will not send anything.) Again, from the validity property of broadcast, we directly get the required correctness.

Finally, we will prove that if p_c is not omission-corrupted, the output of the protocol will be "alive". This part is straight forward, as a sender who is not omission-corrupted is certain to succeed in the invocation of Broadcast. Since p_c is correct, when he broadcasts $b = 1$ all correct players p_j will output $b_j = 1$, due to the properties of the broadcast, hence, the output of the protocol is 1 for "alive". □

Carrying on with the simulation proof, we provide a sketch of the simulator below: A simulator that would make a real-world execution of the protocol indistinguishable from the ideal world would be described as below.

- If the prescribed player P_s that invokes *Heartbeat* is correct, the simulator does nothing.
- If P_s is omission-corrupted and not correct, the simulator observes the real world execution. If P_s fails to broadcast a value or broadcasts $b = 0$, then the simulator sends $(\mathsf{inform\ omission}, P_s)$ to the functionality.
- If P_s is actively corrupted, the simulator observers the real world execution and sets the respective value (0 or 1) to the functionality.

By the properties of Claim 3.3 stated above together with the properties granted in the hybrid $\mathbf{Hyb}_{BC}$ world from the ideal Broadcast, we get that a correct sender is always able to successfully communicate his alive bit $b = 1$ to all other parties, who all agree that P_s is "alive".
In the opposite case, we have that an omission-corrupted sender that is not correct will either successfully broadcast $b = 0$ according to the protocol or he will fail to broadcast a value, which is interpreted by others as "zombie". In both cases, the simulator observes that and sends the corresponding message to the functionality, setting $z = 0$. As a result the functionality sets $y_i = 0$ for all i and all players will agree on the same output "zombie" for P_s. □

Secret Sharing.

Functionality $\mathcal{F}_{share}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p_d, s)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- In round $\rho = 0$:
  - Initially, set the input value s and the output values $y_1, \ldots, y_n$ to $\perp$.
  - Upon receiving a message (input, sid, v) from p_d (or the adversary, if the player is actively corrupted), set $s = v$.
- (Sharing phase): If $p_d \notin A^\star$, for all $S_k \in \mathcal{S}$ (where $|\mathcal{S}| = m$) sample uniformly random $s_1, \ldots, s_m$ such that $\sum_{k=1}^{m} s_k = s$. Then for all s_k such that $S_k \cap A^\star \neq \emptyset$, send s_k to the adversary. Otherwise, if $p_d \in A^\star$, upon receiving $(\mathsf{set\ rd\text{-}sum}, (v_1, \ldots, v_m))$ from the adversary, set $s_1, \ldots, s_m = v_1, \ldots, v_m$.
- At any point in time:
  - Upon receiving $(\mathsf{inform\ omission}, p_d)$ from the adversary, if $p_d \in \Omega^\star$, set $s = \perp$, output (omission, p_d) to p_d and abort with $B = \{p_d\}$.
  - Upon receiving $(\mathsf{inform\ omission}, p_j)$ for some $p_j \in \mathcal{P}$ from the adversary, if $p_j \in \Omega^\star$ output (omission, p_j) to p_j and abort with $B = \{p_j\}$. Otherwise ignore.
- In round $\rho = 1STR$: Upon receiving (fetch-share, sid) from some $p_i \in S_k$, where $k \in \{1, \ldots, m\}$ send $(\mathsf{share}, \mathsf{sid}, s_k)$ to all $p_i \in S_k$ and $(\mathsf{fetch\text{-}output}, \mathsf{sid}, p_i)$ to the adversary.

A very important primitive that is essential in keeping the privacy of the players' input is Secret Sharing. Having split the message s in random summands s_k using a sum share, we can then send each one of them through *DetSMT* to the corresponding set S_k. This idea was first developed in [ISN89] and has since been used in many MPC protocols.

The functionality that we want to instantiate is presented in detail above. To give a brief description, it takes as input a value s that needs to be kept secret. Then, uniformly random shares $s_1, s_2, \ldots, s_m$ where $m = |\mathcal{S}|$, are created such that $s = \sum_{k=1}^{m} s_k$. Each one of those s_k is sent to the respective set S_k (which is the complement of A_k). This way no matter which class $A^\star$ the adversary corrupts, there exists a share $s^\star$ of the set $S^\star = \mathcal{P} \setminus A^\star$ that the adversary does not obtain. Hence the privacy of s is preserved.
In the case where the dealer is actively corrupted, the adversary is allowed to select the shares of s. If the dealer p_d is omission-corrupted, the adversary selects if the Sharing will succeed as normal or if it will abort and p_d will be identified as omission-corrupted.

What makes our implementation simple at this point is the existence of the SMT channel. Instead of sending the messages using the existing network of point-to-point channels, our protocol sends them by invocation of the Protocol *DetSMT* we built earlier. This grants us the detectability and privacy properties directly. Finally, the players invoke a Heartbeat to communicate to all if someone became a zombie.

Protocol *Share*$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p_d, s)$

- In round $\rho = 0$: The dealer p_d chooses the summands $s_2, \ldots, s_m$ randomly, where $m = |\mathcal{S}|$ and sets $s_1 := s - \sum_{k=2}^{m} s_k$.
- (Sharing phase): For each set S_k where $k \in \{1, \ldots, m\}$, the dealer p_d sends s_k by invocation of the protocol $DetSMT(\mathcal{P}, \mathcal{Z}, p_d, p_i, s_k)$ to all players $p_i \in S_k$.
- At any point in time:
  - If any of the invoked sub-protocols aborts with B, *Share* also aborts with B.
  - If p_d broadcasts $\perp$ at any point then Share aborts with $B = \{p_d\}$.
- In round $\rho = 1STR$: All players invoke the protocol *Heartbeat* to communicate if someone has become a zombie. If a player p_z is detected as a zombie the protocol aborts with $B = \{p_z\}$.

Lemma 9. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds and additionally we have that for all $Z_i, Z_j \in \mathcal{Z}$ and for all $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, the protocol* Share *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{share}$.*

Proof. Before providing the simulation proof, we state below the input/output properties of the protocol, necessary for the lemma above.

Claim. If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds and additionally we have that for all $Z_i, Z_j \in \mathcal{Z}$ and for all $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, the protocol *Share* has the following properties: *(correctness)* Share either outputs a consistent sharing $\langle \hat{s} \rangle$ of some $\hat{s}$, where $\hat{s} = s$ unless the dealer p_d is actively corrupted, or it aborts with a set B of corrupted parties. *(secrecy)* No information on s leaks to the adversary.

Proof. First of all, we should point out that the second condition stating that each S_k is not being covered by the union of two actively corrupted and the intersection of their respective omission-corrupted sets is implied by the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ of $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ when we use the natural sharing specification $\mathcal{S}_{\mathcal{Z}}$. This becomes apparent when we consider the BA condition, keeping in mind that for each S_k it holds $S_k = \mathcal{P} \setminus A_k$.
For the correctness property, we can see that if the dealer p_d is correct and the recipients p_i of the summand s_k are alive, then from the *DetSMT* protocol (Claim 4) we are granted the correctness of the value s_k received. In the opposite case where the dealer p_d is not correct, if he broadcasts $\perp$ or another player becomes a zombie and causes a sub-protocol to abort, then the protocol aborts with B. Otherwise, if p_d is incorrect and no protocol aborts and he broadcasts the correct values, then *Share* works correctly (by the guarantees of *DetSMT*) and outputs a consistent sharing of $\langle \hat{s} \rangle$.

After that, the secrecy part is directly inherited from the properties of *DetSMT* and the fact that all s_k are random, subject to their sum being equal to s. From there, the natural sharing specification $\mathcal{S}$ ensures that the adversary never learns all of the summands s_k, giving us privacy on s. □

Now we can continue with the simulation proof of the lemma above. Indeed, we can create a simulator that makes an ideal execution of $\mathcal{F}_{share}$ indistinguishable from a real world execution of *Share* in the hybrid

world where invocations of the *DetSMT* protocol are replaced by ideal calls to $\mathcal{F}_{DetSMT}$, as follows.
• If p_d is correct and not actively corrupted, the simulator does nothing.
• If p_d is incorrect (omission-corrupted), the simulator observes if any of the calls to $\mathcal{F}_{DetSMT}$ fails or if p_d becomes a zombie. In that case, the simulator sends (inform omission, p_d) to $\mathcal{F}_{share}$.
• If p_d is actively corrupted, the real world adversary sends his input s_k for every k to $\mathcal{F}_{DetSMT}$ so the simulator is handed all of these values. Then the simulator sends (set rd-sum, $(s_1, \ldots, s_m)$) to $\mathcal{F}_{share}$ as p_d's input and computes $s = \sum_{k=1}^{m} s_k$ as the input value.
• The simulator observes the real world execution. If at any point a (sub)-protocol aborts with some B, the simulator sends (inform omission, p_z) to $\mathcal{F}_{share}$ for the corresponding player $p_z \in \Omega^\star$.
• Besides that, the simulator sends to the adversary whatever the functionality outputs.

As in both worlds the adversary (when the dealer is not actively corrupted) receives uniformly random shares but not all of them, the view of the adversary in the real world is identically distributed to the view in the ideal world, making the two executions indistinguishable. □

Announce and Reconstruct.

Functionality $\mathcal{F}_{ann}(\mathcal{P}, \mathcal{Z}, S_k, s_k)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- In round $\rho = 0$:
  - Initially, set the input values $x_1, x_2, \ldots, x_n$ and the output values $y_1, y_2, \ldots, y_n$ to $\perp$.
  - Upon receiving a message (input, sid, v_i) from some $p_i \in S_k$ (or the adversary, if the player is actively corrupted), set $x_i = v_i$ and send (leakage, sid, $p_i, l(x_i) = x_i$) to the adversary.
- In round $\rho = 1BCR$:
  - Set $y_1, y_2, \ldots = x_1, x_2, \ldots$, where the number of values is equal to the number of players in S_k.
  - Upon receiving (fetch-share, sid) from some $p_i \in S_k$, send (share, sid, $(y_1, y_2 \ldots)$) to all $p_i \in S_k$ and (fetch-output, sid, $p_i, (y_1, y_2 \ldots)$) to the adversary.
- At any point in time: Upon receiving (inform omission, p_j) from the adversary, if $p_j \in \Omega^\star$, output (omission, p_j) to p_j.

The functionalities $\mathcal{F}_{ann}$ and $\mathcal{F}_{recn}$ for the Announce and Reconstruct primitives are given above. The protocols *Announce* and *Reconstruct* are closely related as the latter is essentially built on the former. The first one is used to publicly announce the value of a specific summand (using *Broadcast*) and the second one to publicly reconstruct a sharing of a value (using *PublicAnnounce* for all summands), respectively. Both of those protocols are robust, meaning that if our condition holds true and the sharing of the values was successful, those protocols cannot abort and they always succeed.

Protocol *PublicAnnounce*$(\mathcal{P}, \mathcal{Z}, S_k, s_k)$

- In round $\rho = 0$: Every $p_i \in S_k$ publishes his value for s_k, denoted as $s_k^{(i)}$, by using *Broadcast*$(\mathcal{P}, \mathcal{Z}, p_i, s_k)$.
- In round $\rho = 1BCR$: Every $p_j \in \mathcal{P}$, using the broadcast values, determines the set $V \subseteq \mathbb{F}$ of values that are "explainable" by some adversary class in $\mathcal{Z}$, i.e. set $V := \{v = s_k^{(i)}\}$ if there exists $Z = (A, \Omega) \in \mathcal{Z}$ such that $\{p_i \in S_k : s_k^{(i)} = \perp\} \subseteq \Omega$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$.
  - If p_j cannot determine such a set, he becomes a zombie ($p_j \in \Omega^\star$).
- Every $p_j \in \mathcal{P}$ outputs $v \in V$ if $|V| = 1$. Otherwise, output a default value $v = 0$.

Lemma 10. *If the condition* $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 6) holds, assuming that* s_k *is a summand of a consistent sharing of a value* s*, the protocol* PublicAnnounce *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{ann}$*.*

Proof. Before providing the sketch of the simulation proof, we state the claim for the input/output properties of the protocol below.

Claim. If the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P},\mathcal{Z})$ (see Eq. 6) holds and additionally we have that for all $Z_i, Z_j \in \mathcal{Z}$ and for all $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, assuming that s_k is a summand of a consistent sharing of a value s, the protocol *PublicAnnounce* publicly announces the value of the summand s_k.

Proof. For the proof of the claim, we will examine the possible cases for $|V|$. Since we assume a consistent sharing that successful and it was done with respect to the natural sharing specification $\mathcal{S}_{\mathcal{Z}}$, a correct summand s_k will be given to at least one honest party. This is because no S_k can be covered by some Ω_i. Indeed, assuming towards contradiction that this was the case, we would have that

$$\Omega_i \supseteq S_k \implies \Omega_i \supseteq \mathcal{P} \setminus A_k \implies \Omega_i \cup A_k \supseteq \mathcal{P}, \tag{36}$$

by the definition of S_k, contradicting $C^{(A,\Omega)}_{FIXR}(\mathcal{P},\mathcal{Z})$ and therefore $C^{(A,\Omega)}_{MPC}(\mathcal{P},\mathcal{Z})$. Hence, for each p_j the set V contains this summand. This happens because the check for the values in the set V is obviously satisfied by s_k and the honest party that has s_k will correctly broadcast it. If $|V| = 0$, according to the above it means that p_j has no access to the outputs of Broadcast and to the right value s_k and he becomes a zombie.

If $|V| = 1$ the protocol publicly announces the value of s_k correctly, as there exists only one value which is explainable and the correct value s_k is guaranteed to be in V.

If $|V| \geqslant 2$ we will show that our security condition could not hold. Since $|V| \geqslant 2$ there exist v_1, v_2 with $v_1 \neq v_2$ both satisfying the condition for belonging in V, where only one of them could be the correct one (because the and the protocol aborts. Let us assume that v_1 is the correct one. This means that there exist $Z_1 = (A_1, \Omega_1)$, $Z_2 = (A_2, \Omega_2)$, with $A_1 \subseteq A^\star, \Omega_1 \subseteq \Omega^\star$ such that all $p_i \in S_k$ with $s_k^{(i)} = \perp$ satisfy both $p_i \in \Omega_1 \subseteq \Omega^\star$ and $p_i \in \Omega_2$. Additionally, $\{p_i \in S_k : s_k^{(i)} \notin \{v_1, \perp\}\} \subseteq A_1 \subseteq A^\star$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v_2, \perp\}\} \subseteq A_2$. Putting all these together we get that all values from players in S_k are covered by the above sets or more formally

$$A^\star \cup A_2 \cup (\Omega^\star \cap \Omega_2) \supseteq S_k, \tag{37}$$

which violates the assumption $A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$ of the claim, a contradiction. This means that assuming that the sharing of s was wrong, meaning that s was shared inconsistently by some actively-corrupted party.
As such, the parties can output a default value $v = 0$ for the summand s_k without compromising the correctness of the output. □

Now we are ready to give the description of the simulator. As per the previous proofs, we are working in the $\mathbf{Hyb}_{BC}$ Hybrid world were parties can make ideal calls to $\mathcal{F}_{BC}$. Since the adversary is deterministic and does not use any randomness, the only thing that the simulator needs to know before being able to perfectly replicate a real world execution of the protocol is what omission corrupted players are blocked from using $\mathcal{F}_{BC}$ and which/how actively corrupted parties deviate from following the protocol. In our case, the protocol is just a call of $\mathcal{F}_{BC}$by each party in S_k, which makes the simulation more simple.

- The simulator observes if some party p_f broadcasts $\perp$ or fails to broadcast in the real world execution. Then, he sends the corresponding message $(\mathsf{inform\ omission}, p_f)$ to $\mathcal{F}_{ann}$.
- The simulator observes the broadcast values v_c of every actively corrupted party p_c and sends to $\mathcal{F}_{ann}$ the message (input, sid, v_c) to set the corresponding inputs.
- The simulator outputs to the real world adversary externally whatever output was generated by the program of the adversary that the simulator run internally.

As a result, the simulator can create a view for the real world adversary that is indistinguishable from the ideal world, hence proving the lemma. □

Since *PublicAnnounce* is robust and does not abort, it becomes apparent that *Reconstruct* is also robust and if the protocols called up to that point have succeeded, it correctly reconstructs the desired value s from its summands that are announced one by one.

Functionality $\mathcal{F}_{recn}(\mathcal{P}, \mathcal{Z}, S, \langle s \rangle)$

The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- In round $\rho = 0$:
  - Initially, set the input values $\{x_1, \ldots, x_m\}$, where $x_i = \{x_{i_1}, \ldots, x_{i_\ell}\}$, and the output values $s, y_1, \ldots, y_m$ to $\perp$.
  - Upon receiving a message (input, sid, $\langle s \rangle_i$) with p_i's share $\langle s \rangle_i = (s_{i1}, s_{i2}, \ldots, s_{i\ell})$ from some $p_i \in S_k$ (or the adversary, if the player is actively corrupted), set $x_k = v_i$ and send (leakage, sid, $p_i, l(x_k) = x_k$) to the adversary.
- In round $\rho = 1BCR$:
  - Set $y_1, \ldots, y_m = x_1, \ldots, x_m$ and $s = \sum_{k=1}^{m} s_k$.
  - Upon receiving (fetch-share, sid) from some $p_i \in \mathcal{P}$, send (share, sid, $(s, y_1, \ldots, y_m)$) to all p_i and (fetch-output, sid, $p_i, (y_1, y_2 \ldots)$) to the adversary.
- At any point in time: Upon receiving (inform omission, p_j) from the adversary, if $p_j \in \Omega^\star$, output (omission, p_j) to p_j.

Protocol *Reconstruct*$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle)$

- In round $\rho = 0$: For every $S_k \in \mathcal{S}$, protocol *PublicAnnounce*$(\mathcal{P}, \mathcal{Z}, S_k, s_k)$ is invoked (can be done in parallel) to announce to all players the correct summand s_k.
- In round $\rho = 1BCR$: Every $p_i \in \mathcal{P}$ locally computes $s = \sum_{k=1}^{|\mathcal{S}|} s_k$ and outputs s.

Lemma 11. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, assuming that s_k is a summand of the correct sharing of a value s, the protocol* Reconstruct *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{recn}$.*

Proof. Following is the claim for the input/output properties of the protocol that we will need for the proof of the lemma.

Claim. If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (Eq. 6) holds and additionally we have that for all $Z_i, Z_j \in \mathcal{Z}$ and for all $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, assuming that s_k is a summand of a consistent sharing of a value s, the protocol *Reconstruct* publicly reconstructs s.

Proof. If the conditions of the claim hold true, we are granted that the properties of *PublicAnnounce* from Claim 3.3 are also true. This means that no protocol *PublicAnnounce* aborts since they are robust and for each S_k we have that the summand s_k is correctly publicly announced. After that, all players p_j who are alive hold in their possession all summands s_k, $k = 1, \ldots, |\mathcal{S}|$ and due to the Broadcast properties, all of them agree on those values. At this point, all that remains for each p_j is to locally compute $s := \sum_{k=1}^{|\mathcal{S}|} s_k$ in order to obtain the correct value for s. □

To continue our simulation proof, we can use the $\mathbf{Hyb}_{ann}$ Hybrid world. This gives us that each ideal call of $\mathcal{F}_{ann}$ outputs the same value for s_k to all alive parties. The adversary cannot cause the protocol to abort and cannot block any messages, because the protocol is only calling $\mathcal{F}_{ann}$ and then locally computing the sum of the outputs. As a result, the simulator needs only to run internally the program of the real world adversary and output to the actual adversary externally whatever the program outputs. This creates an identical view between any real world and ideal world execution for the adversary. □

3.4 Computing the gates

Addition. The first type of gate that we need to implement is the Addition gate. At each such gate, given the sharing $\langle s \rangle$ and $\langle t \rangle$ of s, t the players need to compute a sharing of their sum $s + t$. The simplest way to create this new sharing is to have each party p_j locally compute the sum of his shares of the two values and set $\langle s + t \rangle_j = \langle s \rangle_j + \langle t \rangle_j$. This is way we create a sharing that is random (as the sum of two random

summands), hides the value of $s+t$ as it did for s,t and is consistent, as long as the initial sharings were consistent.

Functionality $\mathcal{F}_{add}(\mathcal{P},\mathcal{Z},\mathcal{S},\langle s\rangle,\langle t\rangle)$

- In round $\rho=0$:
  - Initially, for every $p_j\in\mathcal{P}$ set the input vectors $\langle s\rangle_j$ and $\langle t\rangle_j$, as well as the output vector $\langle x\rangle_j$ to $\perp$. [a]
  - Upon receiving $\langle s\rangle_j$ and $\langle t\rangle_j$ from p_j, set $\langle x\rangle_j=\langle s\rangle_j+\langle t\rangle_j$.
  - Upon receiving (fetch-output, sid) from $p_j\in\mathcal{P}$, send (output, sid, $\langle x\rangle_j$) to p_j.

[a] For each p_j we have that $\langle s\rangle_j:=(s_{j_1},\ldots,s_{j_\ell})$ is p_j's share, consisting of the summands of s held by p_j.

Protocol $Add(\mathcal{P},\mathcal{Z},\mathcal{S},\langle s\rangle,\langle t\rangle)$

- In round $\rho=0$:
  - Input: A sharing $\langle s\rangle$ of the value s and $\langle t\rangle$ of the value t, previously shared to the parties. Every party $p_j\in\mathcal{P}$ holds his shares $\langle s\rangle_j$ and $\langle t\rangle_j$.
  - Each party p_j in $\mathcal{P}$ (locally) computes his share of the sum $s+t$ as the sum of his respective shares of s and t.
    Formally, set $\langle x\rangle_j=\langle s\rangle_j+\langle t\rangle_j$, i.e. add for each S_k add the component of s with the respective component of t.
  - Output: a sharing of $s+t$, where each party $p_j\in\mathcal{P}$ outputs his share $\langle x\rangle_j=\langle s+t\rangle_j$.

Lemma 12. *If the condition $C^{(A,\Omega)}_{MPC}(\mathcal{P},\mathcal{Z})$ (see Eq. 6) holds, the protocol* Add *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{add}$.*

Proof. The proof for this specific protocol is trivial as no information is exchanged with other parties. Each player locally performs the calculation that the trusted party would perform in the ideal world. Hence, the protocol *Add* perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{add}$. □

Multiplication.

Functionality $\mathcal{F}_{mult}(\mathcal{P},\mathcal{Z},\mathcal{S},\langle s\rangle,\langle t\rangle)$

The adversary corrupts $Z^\star=(A^\star,\Omega^\star)$.

- Initially, set the input[8] values $\{s_1,\ldots,s_m\}$ and $\{t_1,\ldots,t_m\}$, where $s_i=\{s_{i_1},s_{i_2},\ldots\}$, and the output values $[y]=(y_1,\ldots,y_m),[st]=(s_1t_1,s_1t_2,\ldots,s_mt_m)$ and $\langle y\rangle=(\langle y\rangle_1,\langle y\rangle_2,\ldots,\langle y\rangle_n)$ to $\perp$.
- In round $\rho=0$:
  - Upon receiving a message (input, sid,v) from p_k (or the adversary, if the player is actively corrupted) with a vector $v=(v_{k_1},v_{k_2},\ldots)$ containing p_k's share of s, set the respective $s_{\ell_k}=v_{k_\ell}$. Similarly, receive input for t.
  - If $s_{\ell i}=s_{\ell j}=v$ for every two $p_i,p_j\in S_\ell\setminus A^\star$ then $\mathcal{F}_{mult}$ sets $s_\ell=v$. Otherwise adopt $s_{\ell i}$ from first honest p_i. Similarly for t. Then $\mathcal{F}_{mult}$ sets $x_{\ell,q}:=s_\ell\cdot t_q$.
  - In round $\rho=1STR$: For all $(S_k,S_\ell)\in\mathcal{S}\times\mathcal{S}$ repeat the following:
    1. If $(S_k\cap S_\ell)\cap A^\star=\emptyset$, for all $q\in\{1,\ldots,m\}$ with $Z_q=(A_q,\Omega_q)\in\mathcal{Z}$ such that $S_q\cap A^\star\neq\emptyset$, sample uniformly random $v_1,\ldots,v_m$ summing up to $x_{k,\ell}$ and forward v_q to the adversary. Repeat for all $p_r\in S_k\cap S_\ell$.
       Then $\mathcal{F}_{mult}$ sets $[x_{k,\ell}]=(v_1,\ldots,v_m)$.

2. Else, if $(S_k \cap S_\ell) \cap A^\star \neq \emptyset$, additionally to the previous step, upon receiving (adv-share, $(z_1, \ldots, z_m)$) from the adversary check the following:
If $\sum_{i=1}^{m} z_i = x_{k,\ell}$, then set $[x_{k,\ell}] = (z_1, \ldots, z_m)$.
Else if their sum is not $x_{k,\ell}$, publicly announce s_k as s_{k_g} and t_ℓ as t_{ℓ_g} from the share of the first honest player p_g in $S_k \cap S_\ell$ and set
$[x_{k,\ell}] := [s_k t_\ell] = (s_{k_g} \cdot t_{\ell_g}, 0, \ldots, 0)$ as the way that $x_{k,\ell} = s_k t_\ell$ is split into m summands.

– Phase 3: Do nothing.
– In round $\rho = 1STR + 3BCR$: For the output $[y] = (y_1, \ldots, y_m)$ $\mathcal{F}_{mult}$ sets
$y_1 = \left[\sum_{k,\ell=1}^{m} x_{k,\ell}\right]_1, \ldots, y_m = \left[\sum_{k,\ell=1}^{m} x_{k,\ell}\right]_m$ (for each y_q we sum the q-th summand of all $x_{k,\ell}$).
  - For all $k = 1, \ldots, m$, upon receiving (fetch-output, sid) from $p_j \in S_k$, send (output, sid, y_k) to p_j and (fetch-output, sid, p_j) to the adversary.
– Upon receiving (inform omission, p_j) from the adversary for some $p_j \in \mathcal{P}$, if $p_j \in \Omega^\star$, output (omission, p_j) to p_j and abort with $B = p_j$.

Our next goal is to to securely compute a sharing of the product of two shared values. Its properties are that, as long as our conditions hold, given two consistent sharings $\langle s \rangle, \langle t \rangle$ it securely creates a consistent sharing of $\langle s \cdot t \rangle$, or it aborts after detecting a set B of incorrect/corrupted players. Those properties are captured by the functionality $\mathcal{F}_{mult}$ above.

Initially, the functionality receives input in the form of sharings, where each player p_i inputs his shares $\langle s \rangle_i$ and $\langle t \rangle_i$ for s and t, respectively. The adversary can select the shares for the player he controls. After that, the functionality checks whether the input of all non-actively corrupted parties for every summand s_k is the same, i.e. checks whether the sharing is consistent. If it is, s_k is fixed to this value (similarly for every t_ℓ). Otherwise, the values of the first honest player are adopted. Then, the product $x_{k,\ell}$ of any two summands s_k, t_ℓ is calculated. Next each such product needs to be shared to all parties according to $\mathcal{S}$. This is performed by all players holding $x_{k,\ell}$. Once the sharing of all those products is completed, all parties can locally add their shares of $x_{k,\ell}$ over all combinations of k, ℓ to obtain a share of the final product $y = st$.

We note that if the adversary controls a party that can compute $x_{k,\ell}$, he is able to select how this product is shared, i.e. how it is split into summands $[x_{k,\ell}] = (z_1, \ldots, z_m)$ and importantly, he can impose this choice to the honest players, subject to the summands adding up to $x_{k,\ell}$. This was observed and dealt with in detail in the work of Asharov, Lindell and Rabin [ALR11]. Alternatively, the adversary is able to completely deviate from creating a sharing of the correct value and select summands that do not add up to $x_{k,\ell}$, but in this scenario the functionality detects that and adopts the values for s_k and t_ℓ from an honest player. Then, a default sharing of $x_{k,\ell} = s_k t_\ell$ is created as $[x_{k,\ell}] = (s_k t_\ell, 0, \ldots, 0)$. To argue that there always exists a non-actively corrupted player having both s_k and t_ℓ we can transform our security condition from Broadcast as follows.

$$
\begin{aligned}
C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \iff A_k \cup A_\ell \cup A_m \cup (\Omega_k \cap \Omega_\ell) \not\supseteq \mathcal{P} \\
A_k \cup A_\ell \not\supseteq \mathcal{P} \setminus (A_m \cup (\Omega_k \cap \Omega_\ell)) \\
\mathcal{P} \setminus (A_k \cup A_\ell) \not\subseteq A_m \cup (\Omega_k \cap \Omega_\ell) \\
(\mathcal{P} \setminus A_k) \cap (\mathcal{P} \setminus A_\ell) \not\subseteq A_m \cup (\Omega_k \cap \Omega_\ell) \\
(S_k \cap S_\ell) \not\subseteq A_m \cup (\Omega_k \cap \Omega_\ell),
\end{aligned}
$$

which gives us that $(S_k \cap S_\ell)$ cannot be covered by $A_m \cup (\Omega_k \cap \Omega_\ell)$, and by extension (since that holds for any m) cannot be covered by $A^\star$. Since there always exists an non-actively corrupted party in $S_k \cap S_\ell$, the adversary cannot tamper with the value of $x_{k,\ell}$ and in this case both s_k and t_ℓ are publicly announced to all players so that all adopt the correct values.
If at some point the adversary decides to make a player aware of his omission status, the player is informed and publicly steps down, while the functionality aborts having detected a corrupted party.

Our implementation of that functionality is protocol *Mult* and it is based on the respective protocols of [Mau02,BFH+08]. The idea of the protocol is the following: As s and t are shared according to $\mathcal{S}$, we can

use the summands $s_1, \ldots, s_{|\mathcal{S}|}$ and $t_1, \ldots, t_{|\mathcal{S}|}$ to compute the product st as the sum of the products of all those s_i, t_j, i.e.

$$st := \sum_{k=1}^{|\mathcal{S}|} \sum_{\ell=1}^{|\mathcal{S}|} s_k t_\ell = \sum_{k,\ell=1}^{|\mathcal{S}|} s_k t_\ell. \tag{38}$$

Each term $x_{k,\ell} = s_k t_\ell$ is shared by every player in $S_k \cap S_\ell$. After that the players try to see if they agree on the shared summands, by computing and reconstructing all the differences of the $x_{k,\ell}$ shared. If they do not agree, either the sharing was not consistent (due to the adversary inputting wrong values earlier on) or the adversary controls some party in $S_k \cap S_\ell$. In either case, it is safe to publicly announce both s_k and t_ℓ so that everyone agrees on the value of those summands and adopt a default sharing for their product $x_{k,\ell}$.

After doing this for all combinations of k, ℓ, the players compute the sum of the shared terms $x_{k,\ell}$, which results in a sharing of st, as desired.

Protocol *Mult*$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$

- For every $(S_k, S_\ell) \in \mathcal{S} \times \mathcal{S}$, the following steps are executed:
  - i) In round $\rho = 0$:
    Every $p_i \in (S_k \cap S_\ell)$ computes the products $x_{k,\ell} := s_k t_\ell$ and invokes *Share*$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p_i, x_{k,\ell})$; denote the resulting sharing as $\left\langle x_{k,\ell}^{(i)} \right\rangle$.
  - ii) In round $\rho = 1STR$:
    Let p_i denote the player with the smallest index in $(S_k \cap S_\ell)$. For every $p_j \in (S_k \cap S_\ell)$, the difference $\left\langle x_{k,\ell}^{(j)} \right\rangle - \left\langle x_{k,\ell}^{(i)} \right\rangle$ is computed and is reconstructed by invoking the protocol *Reconstruct*.
  - iii) In round $\rho = 1STR + 1BCR$:
    If all differences are 0, then the sharing $\left\langle x_{k,\ell}^{(i)} \right\rangle$ of p_i is adopted as sharing of $x_{k,\ell}$, i.e., $\langle x_{k,\ell} \rangle := \left\langle x_{k,\ell}^{(i)} \right\rangle$.
    Otherwise (i.e., some difference is non-zero), *Public Announce* is invoked to have both s_k and t_ℓ announced, and a default sharing $\langle x_{k,\ell} \rangle$ of $x_{k,\ell} = s_k t_\ell$ is created (e.g., the first summand is set to $x_{k,\ell}$ and the other summands are set to 0).
- In round $\rho = 1STR + 2BCR$:
  All players invoke the protocol *Heartbeat* for all to see if someone has become a zombie. If a player p_z is detected as a zombie, *Mult* aborts with $B = \{p_z\}$.
- In round $\rho = 1STR + 3BCR$:
  - Each player in $\mathcal{P}$ (locally) computes his share of the product $s \cdot t$ as the sum of his shares of all terms $x_{k,\ell}$.
  - Output: a sharing of st, where each party $p_j \in \mathcal{P}$ outputs his share $\langle st \rangle_j$.
- If any of the invoked sub-protocols aborts with B, then *Mult* aborts with B.

Lemma 13. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, $\langle s \rangle$ and $\langle t \rangle$ are consistent sharings according to $\mathcal{S}$ and the following properties hold: for all $Z_i = (A_i, \Omega_i), Z_j = (A_j, \Omega_j) \in \mathcal{Z}$ and $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, as well as for all $S_k, S_\ell \in \mathcal{S}$ and for all $Z_i = (A_i, \Omega_i) \in \mathcal{Z} : S_k \cap S_\ell \not\subseteq A_i$, the protocol* Mult *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{mult}$.*

Proof. Instead of proving that protocol *Mult* securely realizes the functionality $\mathcal{F}_{mult}$, we will do it for the hybrid protocol $\mathbf{Hyb}_{mult}$ which instead of using protocol *Share* and *PublicAnnounce*, it makes ideal calls to the functionalities $\mathcal{F}_{share}$ and $\mathcal{F}_{ann}$ internally. From there, the statement of the lemma follows using the composition theorem of [Can01]. Before we give our simulation-based proof, we need to state the following useful claim about the input/output properties of the protocol.

Claim. If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, $\langle s \rangle$ and $\langle t \rangle$ are consistent sharings according to $\mathcal{S}$ and the following properties hold: for all $Z_i = (A_i, \Omega_i), Z_j = (A_j, \Omega_j) \in \mathcal{Z}$ and $S_k \in \mathcal{S} : A_i \cup A_j \cup (\Omega_i \cap \Omega_j) \not\supseteq S_k$, as well as for all $S_k, S_\ell \in \mathcal{S}$ and for all $Z_i = (A_i, \Omega_i) \in \mathcal{Z} : S_k \cap S_\ell \not\subseteq A_i$ the protocol *Mult*$(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s \rangle, \langle t \rangle)$ has the following properties: *(correctness)* It either outputs a sharing of st according to $\mathcal{S}$ or it aborts with a non-empty set B of incorrect players. *(secrecy)* No information leaks to the adversary.

Proof. To begin with, by selecting the $\mathcal{S}_{\mathcal{Z}}$ as our sharing specification we get that both other conditions are implied by $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$. We have already proven that for the first one in Claim 3.3. We will show that the second condition is implied, as well.
For all k by definition we have $S_k = \mathcal{P} \setminus A_k$, so we get that $S_k \cap S_\ell = \mathcal{P} \setminus (A_k \cup A_\ell)$. Now it is easy to see that $S_k \cap S_\ell \nsubseteq A_i$, for all A_i is equivalent to $\mathcal{P} \setminus (A_k \cup A_\ell) \nsubseteq A_i$, which eventually can be written as

$$A_i \cup A_k \cup A_\ell \neq \mathcal{P},$$

which is easily implied by $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$.

For proving the correctness we will show that for each pair s_k, t_ℓ there exist a player who is not actively corrupted that has access to both values. Indeed, the condition stating that for all k, ℓ, i we have $S_k \cap S_\ell \nsubseteq A_i$ ensures that any pair s_k, t_ℓ is known to at least one player who is not actively corrupted. As a result, if no invocation of *Share* aborts and all differences are zero, then the shared values are correct and we have a successful sharing of all $x_{k,\ell} = s_k t_\ell$. Since the product st is defined as

$$st := \sum_{q=1}^{m} \sum_{r=1}^{m} s_q t_r,$$

when $\langle s \rangle = (s_1, \ldots, s_m)$ and $\langle t \rangle = (t_1, \ldots, t_m)$ (i.e. $s = \sum_{q=1}^{m} s_q$ and $t = \sum_{r=1}^{m} t_r$) each player is able to locally compute a sharing of the product st according to $\mathcal{S}$ by adding the values of the sharing of $x_{k,\ell}$ that he possesses.

For the secrecy of the protocol, from the secrecy of Claim 3.3 we get that no information is leaked from an invocation of *Share*.
Next, the reconstruction of the differences $\left\langle x_{k,\ell}^{(j)} \right\rangle - \left\langle x_{k,\ell}^{(i)} \right\rangle$ either outputs 0 and no information is leaked, or it does not output 0, meaning that the adversary controls a (actively corrupted) player $p_b \in S_k \cap S_\ell$ and he already has knowledge of s_k, t_ℓ.
Furthermore, *PublicAnnounce* is invoked to announce the summands s_k, t_ℓ only when there are two players in $S_k \cap S_\ell$ who contradict each other. This, again, means that the adversary is actively corrupting one of them and has already access to the values to be announced. Hence, no information is leaked to the adversary, completing our proof. □

Now we can continue our simulation proof. Assuming that the sharings of s and t are consistent, according to the properties of the Claim 3.4 that we proved above, we know that the protocol *Mult* either aborts while detecting a corrupted player or it outputs a consistent sharing. As a result, if in the real world the protocol aborts at any time with a public set B of corrupted parties, then the simulator observes this and can create the same effect in the ideal world by sending the corresponding (inform omission) message to $\mathcal{F}_{mult}$.

The different cases that can occur with consistent sharings for s and t are the following two:
Case 1: If $(S_k \cap S_\ell) \cap A^\star = \emptyset$. This case is the easier one because the adversary does not participate actively in the step 3.4 so it is trivially simulated (i.e. the simulator does nothing). Indeed, only players in $(S_k \cap S_\ell)$ create and share their values, while the adversary does not control a player in $S_k \cap S_\ell$ and does not know $x_{k,\ell} = s_k t_\ell$. Hence, in the real world the adversary observes only the output of the corrupted parties after *Share* (or rather $\mathcal{F}_{share}$ when we are in the Hybrid world) is invoked by all parties in $S_k \cap S_\ell$, i.e. he obtains a part of a sharing of $x_{k,\ell}$ (only the shares that corrupted parties receive). After that, each party locally (with no communication involved) computes its share of $\left\langle x_{k,\ell}^{(i)} \right\rangle - \left\langle x_{k,\ell}^{(j)} \right\rangle$. Afterwards, the adversary learns the reconstruction of those differences for every j. However, this reveals no information as the reconstructed differences should all be 0 for a consistent sharing, since the security condition holds and they are created by non actively corrupted parties (i.e. all p_q in $S_k \cap S_\ell$ agree on the value of the summands s_k and t_ℓ, thus they agree on $x_{k,\ell}$ as well). Furthermore, what is reconstructed are differences of two random summands $x_{k,\ell}^{(i)}$ and $x_{k,\ell}^{(j)}$, none of which the adversary knows. So he does not obtain extra information.
Also, according to the properties of $\mathcal{F}_{share}$, all the shares that the adversary receives about those reconstructed summands should be indistinguishable from random, due to the adversary being able to only see a

fraction of the summands.
Furthermore, during steps 2, 3 and 4 no information (except from the alive status) is exchanged between parties. Now, the players have a sharing of $x_{k,\ell}$ for all k, ℓ according to the sharing of some p_i. For the output, at step 4 all parties calculate their share of st as the sum of the local shares of each $x_{k,\ell}$ that they received, creating a sharing of st across all parties. Each summand of $x_{k,\ell}$ was uniformly random, thus the sum of those summands for every player creates a uniformly random summand for the sharing of st. The adversary only gets the values that actively corrupted parties have access to and not the whole product.

Respectively, in the ideal world all players will input the same value v_s for s_k and v_t for t_ℓ, since the sharings are consistent. Hence, the functionality will set $s_k = v_s$ and $t_\ell = v_t$ and then, it will compute and set $x_{k,\ell} := s_k \cdot t_\ell$. Next, the adversary receives his shares (that are as many as in the real world) from randomly sampled values $v_1, \ldots, v_m$ summing up to $x_{k,\ell}$. Obviously, no information is leaked here as the adversary does not receive all of the v_is.

Finally, the functionality outputs a sharing of $y = st$ in the form of $[y] = (y_1, \ldots, y_m)$ for the summands y_q, where for each $q = 1, \ldots, m$ it sums all the random summands of $x_{k,\ell}$ at the q-th position, i.e. $[y]_q = y_q = \sum_{k,\ell=1}^{m} [x_{k,\ell}]_q$.
As the sum of uniformly random values, this is a uniformly random sharing of st, as required.

Thus, in both worlds, the messages that the adversary receives follow the same uniform distribution.

Case 2: If the adversary controls a player in $S_k \cap S_\ell$, in the real world he is able to wait until all $x_{i,j}$ except for the last one have been established and then select the summands that he will share for $x_{k,\ell}$ such that each of them defines the respective summand of $s_k t_\ell$ (all but one) to a value of his choice. Indeed, if the corrupted party is the first player in $S_k \cap S_\ell$ it could be the case that his shares are the ones adopted by all other players. This could create a big security problem and open up the possibility for an attack. However, the adversary is still bound to have the summands add up to the correct value, since otherwise the reconstructed differences will be non-zero. Fortunately, the simulator can observe the shares that the adversary chooses and then he is able to create the same effect in the ideal world by sending (adv-share, $(z_1, \ldots, z_m)$) to $\mathcal{F}_{mult}$, so the two worlds can not be distinguished at that point, either.

Alternatively, the adversary can also actively alter the value of the sharing of the difference $\left\langle x_{k,\ell}^{(i)} \right\rangle - \left\langle x_{k,\ell}^{(j)} \right\rangle$ by using arbitrary values as his summands. Still, the only effect that this will have is to create a non-zero difference between the sharing that the adversary created and the sharing that some non actively corrupted party created (since not all players can be actively corrupted, this party always exists).

When the simulator observes this he can create the same effect in the ideal world by sending (adv-share, $(z_1, \ldots, z_m)$) for z_i that don't sum up to the correct $x_{k,\ell}$.

Back in the real world, as a result of the differences not being 0, the values of s_k and t_ℓ will be reconstructed publicly and all parties will adopt the same value, hence the same value for $x_{k,\ell} = s_k t_\ell$. The adversary already knows them so no information is leaked. The same occurs in the ideal world, where the values for s_k, t_ℓ are adopted from the first hones player.
Then a sharing of $x_{k,\ell}$ is assumed as $[x_{k,\ell}] := [s_k t_\ell] = (s_{kg} \cdot t_{\ell g}, 0, \ldots, 0)$. From then on, the same procedure as in case 1 follows and the output stage is reached.

Since in both worlds the outputs of honest parties and the view of the adversary are following the same distribution, the two worlds cannot be distinguished by any environment/distinguisher, making our protocol secure. □

3.5 The MPC protocol

We next proceed to the construction of our *MPC* protocol, which securely realizes the functionality $\mathcal{F}_{MPC}$. The function to be computed will be represented by a circuit $\mathcal{C}$. Our protocol will compute the desired circuit on the inputs of the players. If none of the sub-protocols aborts, the protocol will succeed and give the correct output. In the opposite case, where the adversary has misbehaved and caused a protocol to abort we will identify a set B of corrupted parties. Then we will restart the computation of the protocol from the beginning with a smaller structure, setting $\mathcal{P} := \mathcal{P} \setminus B$, using *strong player elimination* (see 3.1), as the players in B are all problematic. Importantly, this action preserves the monotonicity of the condition, namely

that the MPC condition is also true in the new updated adversary structure. We should also point out that even in the case of such an abortion no information about the players' input is leaked to the adversary. This is because all calculations are done with sharings of the inputs, hence the actual values are hidden. The only time where a value is actually revealed is after the *Reconstruct* protocol. However, our *Reconstruct* protocol is robust, meaning that it cannot abort and if the protocol has reached this point, it is guaranteed to succeed. Additionally, after having identified some corrupted players, we no longer care for all the adversary classes $Z = (A, \Omega)$ that we initially had. Instead, we are only interested in adversary classes Z which actually contain the identified players in their Ω set. For writing convenience we will introduce the following notation, where we will write $\mathcal{Z}|^{B \subseteq \Omega}$ to denote the restriction of the adversary structure $\mathcal{Z}$ to contain only the classes $Z = (A, \Omega)$ where $B \subseteq \Omega$. In plain words, we consider the restriction of the adversary structure where only classes that assume the set B to be omission-corrupted are preserved.

Here, we should take a moment to discuss the issue of removing players from our player set. Indeed, we will showcase the difference between the classic (weak) player elimination and our proposed strong variant. Our main concern is to make sure that after every such removal of a set B, no information is leaked to the adversary and, additionally, given that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ was true for the initial structure and player set, then it will hold true for the new updated player set, as well. The argument is as follows. We know that by definition $B \subseteq \Omega$ holds for all Ω in $\mathcal{Z}|^{B \subseteq \Omega}$. Since the adversary has made a choice of the class $Z^\star = (A^\star, \Omega^\star)$ that she will corrupt and since for the set of corrupted players that we are going to discard it holds that $B \subseteq \Omega^\star$ (because the players in B are indeed corrupted), for all $Z_i, Z_j \in \mathcal{Z}|^{B \subseteq \Omega}$ we have that $B \subseteq \Omega_i \cap \Omega_j$. Furthermore, for all relevant classes $Z \in \mathcal{Z}|^{B \subseteq \Omega}$ the element (player) p that makes the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ hold true does not belong to $A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j)$ before removing B. This means that it does not belong in $\Omega_i \cap \Omega_j$, and since $B \subseteq \Omega_i \cap \Omega_j$ we can safely remove B and the element p will still exist for all classes, making the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ hold true for the updated $\mathcal{P}$ and $\mathcal{Z}$. This ensures us that we can safely remove any corrupted players and still get the condition to hold with our new player set and adversary structure.

As a result, we can restart the *MPC* protocol for as many times necessary, updating $\mathcal{P}$ and $\mathcal{Z}$ each time it aborts and re-running it again with a smaller set, until it succeeds. Since the player set is decreasing every time we repeat that and since the sets A, Ω are finite we can be certain that we will reach a point where this loop will terminate and the invocation to *MPC* will succeed (after no more than n iterations). At this point, all alive players will have received a sharing of the output, and given that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds true, this will be a valid sharing of the actual output. This means that it can be reconstructed and provide to the players the intended output.
As before, we use the natural sharing specification $\mathcal{S}_\mathcal{Z}$ associated with $\mathcal{Z}$, setting $S_\mathcal{Z} = (\mathcal{P} \setminus A_1, \ldots, \mathcal{P} \setminus A_m)$, to make sure that at least one summand of the sharing will not be available to the adversary and to guarantee a private reconstruction of the sharing.

Functionality $\mathcal{F}_{MPC}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \vec{x} = (x_1, \ldots, x_n))$

Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is denoted as $f(\cdot)$. The adversary can decide which of the omission-corrupted players receive output from the functionality after receiving the outputs of actively corrupted players. The adversary corrupts $Z^\star = (A^\star, \Omega^\star)$.

- Input stage:
  - Set the input values $x_1, \ldots, x_n$ and the output values $y_1, \ldots, y_n$ to $\perp$.
  - Upon receiving a message (input, sid, v) from p_i (or the adversary, if the player is actively corrupted), set $x_i = v$ and send (leakage, sid, $p_i, |v|$) to the adversary.
- At any point in time: Upon receiving (inform omission, p_j) from the adversary, if $p_j \in \Omega^\star$ set x_j to a default value (e.g. 0) and output (omission, p_j) to p_j.
- Computation stage: For every $p_j \in \Omega^\star$ that received (omission, p_j) in the step above set $y_j = \perp$.
- Output stage: $\mathcal{F}_{MPC}$ computes $f(x_1, \ldots, x_n) = (y_1, \ldots, y_n)$.

Upon receiving (fetch-output, sid) from $p_i \in \mathcal{P}$ send (output, sid, y_i) to p_i and (fetch-output, sid, p_i) to the adversary.

Moving on to the description of the protocol, it involves three stages, the input, the computation and the output stage.
For the input stage, we have all players share their inputs according to the sharing specification $\mathcal{S}$. This is done to make sure that the inputs remain private, while still being able to perform computations with them. In the case that a player fails to share her input, e.g. if she is corrupted and the adversary blocks her messages, all players adopt a default pre-agreed sharing for her input value. For the evaluation stage, the procedure is the following. Depending on the gate of $\mathcal{C}$ that we want to evaluate, the players do the following. If they need to evaluate an addition gate, each player locally computes the sum of his shares of the two values, so the output is a sharing of the sum.
If they need to evaluate a multiplication gate for two values s, t, the players invoke the protocol $Mult(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s\rangle, \langle t\rangle)$ for the sharings $\langle s\rangle, \langle t\rangle$ and the output is a sharing $\langle st\rangle$ of the product.
If they need to evaluate a random gate, each player sends a random value as input and the output is a sharing of the sum of those values. Lastly, for the output stage where the players want to eventually get the actual value of the output v, they invoke the protocol $Reconstruct(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle v\rangle)$ in order to publicly robustly reconstruct one by one the summands of v and after that each one gets the desired value by summing all summands.

Protocol $MPC(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \mathcal{C}, \vec{x} = (x_1, \ldots, x_n))$

- Input stage:
  - For every input gate in $\mathcal{C}$, $Share(\mathcal{P}, \mathcal{Z}, \mathcal{S}, p_i, x_i)$ is invoked to have the input player p_i share his input x_i according to $\mathcal{S}$ (or the adversary, if the player is actively corrupted). If a player fails to share his input, a default value is adopted by all players.
  - If the protocol *Share* (or any sub-protocol) aborts during some input gate with a set B of corrupted players, set $\mathcal{P} := \mathcal{P} \setminus B$ and $\mathcal{Z} := \mathcal{Z}|^{B \subseteq \Omega}$ and restart the Input Stage.
- Computation stage: The gates in $\mathcal{C}$ are evaluated as follows:
  - Addition gate: Every $p_i \in \mathcal{P}$ locally computes the sum of his respective shares.
  - Multiplication gate: To multiply (shared) values s and t, invoke $Mult(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s\rangle, \langle t\rangle)$ to compute a sharing $\langle st\rangle$ of the product according to $\mathcal{S}$.
  - If the protocol *Mult* (or any sub-protocol) aborts during the computation stage with a set B of corrupted players, set $\mathcal{P} := \mathcal{P} \setminus B$ and $\mathcal{Z} := \mathcal{Z}|^{B \subseteq \Omega}$ and restart from the Input Stage.
- Output stage: Invoke $Reconstruct(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle y\rangle)$ for the sharing $\langle y\rangle$ of the public output.

Theorem 4. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, the protocol* MPC *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{MPC}$.*

Proof. To begin with, we formally state the input/output properties of our protocol below. Then we will carry on with the simulation proof.
Let $\mathcal{C}$ be an arithmetic circuit where each non-byzantine party p_i has input x_i. For omission-corrupted but non-byzantine parties $p_j \in \Omega^\star$ the adversary decides independently of the inputs of non-byzantine parties if the correct input x_j will be used and p_j will receive the correct output or a default value $\perp$ will be used for p_j's input and their output will be $\perp$. For every byzantine p_k the adversary chooses some x_k'' as input independently of the inputs of non-byzantine parties.

Theorem 5. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 6) holds, then the protocol* MPC *securely computes $\mathcal{C}$ on inputs x_i from every $p_i \notin A^\star$ (except from those $p_j \in \Omega^\star$ that the adversary decided to set $x_j = \perp$). In particular, the protocol satisfies the following properties:*
(Correctness): *Every uncorrupted p_i receives his correct output from the above evaluation and every omission but non-byzantine party receives either his correct output or "*$\mathsf{n/v}$*".* (Privacy): *The protocols leaks no information to the adversary, i.e., the adversary learns nothing beyond what he can compute from the specified*

inputs and outputs of the byzantine parties. (Termination): *The protocol terminates with the above outputs after a finite number of rounds.*

Proof. Since the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds, all the protocols invoked inside the protocol *MPC* are guaranteed to be secure, i.e. they securely realize the respective functionalities.
As a result, we only need to prove the properties of Protocol *MPC* in the hybrid world where every invocation of some underlying protocol is replaced by an ideal call of the respective functionality. After that, we get the desired result from the composition theorem from [Can00]. As each stage consists of calls to functionalities that we have already proven correct and terminating, the *termination* of *MPC* is a derivative of that. Also, the maximum number of times that our protocol may restart due to an abort of some sub-protocol is less than n and additionally the Output stage is robust (i.e. never aborts).
The *privacy* property is proven by the fact that *Share* (or rather $\mathcal{F}_{share}$), as well as Addition and Multiplication gates leak no information. Also, the outcome of those protocols/gates is a sharing of the intended output, rather than the actual computed value, so no information is leaked about the values and consequently the input values remain hidden. If there is an abort during the Computation stagethen the protocol restarts at the Input Stage, where input shares are calculated anew and this keeps the privacy of the inputs of the players. On top of that, as mentioned above, *Reconstruct* is robust and does not abort, meaning that once the output stage is reached the adversary cannot disrupt the conclusion of the protocol and all parties receive their outputs. This would be either the correct value or the value $\perp$ if the adversary decides to exclude this omission-corrupted party from the computation.
Along the same lines, for the *correctness* proof we have proven that $\mathcal{F}_{share}$ (*Share*) either creates a consistent sharing of the input (for non-actively corrupted dealers) or aborts. Given that, during the computation stage (if no protocol aborts) the gates are guaranteed to compute the correct result, by their corresponding correctness properties. If some party $p_z \in \Omega^\star$ is detected as zombie and after the underlying Heartbeat the sub-protocol aborts with $B = \{p_z\}$, the player is removed from the player set, $\mathcal{P}$ is updated and the computation is restarted from the beginning, with new shares for the inputs.
Finally, if no sub-protocol aborts and the Output stage is reached, assuming that the sharings up to this point are correct, we are guaranteed by *Announce* and *Reconstruct* through $\mathcal{F}_{ann}$ and $\mathcal{F}_{recn}$ that the correct output is robustly recreated towards all players (except from omission-corrupted players, if the adversary decides to completely block them, but make them aware of their status). This completes the proof of the theorem's properties. □

Moving on to sketch proof of Theorem 4 below we provide a description of the simulator.
Once more, we will use the hybrid world where every invocation of a sub-protocol is replaced with an ideal call to the ideal functionalities that the protocol realizes.

The environment gives input to the honest parties and the simulator begins his execution. The adversary is trying to distinguish if he is interacting with the simulator in the ideal world or with the real world protocol and the parties.
At the Input stage, the adversary needs to provide his input to the *Share* protocol, but since we are in the hybrid world, we instead gives his input to the simulator. This enables the simulator to call the share functionality $\mathcal{F}_{share}$ with that input and at the same time run internally the real world adversary in a protocol execution with that input. Whatever output this gives to the simulator he can use it to interact externally with the functionality and create an execution in the ideal world indistinguishable from the hybrid world.

By this, the share functionality $\mathcal{F}_{share}$ is able to create a sharing of the inputs of all players (honest ones are provided by the environment). Up to this point, the simulator observes the real world. If the adversary makes some party $p_j \in \Omega^\star$ realize that it is omission-corrupted, p_j let's anyone know that during the Heartbeat stage of the *Share* protocol. Then, the sub-protocol would abort and *MPC* would restart while removing p_j from $\mathcal{P}$.
At the same time, the simulator sends $(\mathsf{inform\ omission}, p_j)$ to $\mathcal{F}_{MPC}$ to set p_j's input to the default value and his output to $\perp$, creating the same effect.
Since the distribution of the sharings is the same in the real and ideal world, this means that the view of

the adversary is identically distributed in real and ideal world, up to this point. Due to that, the adversary cannot distinguish between the two and proceeds with the execution.

Next, we have the Computation stage. The addition gates are trivial due to being locally computed. For the Multiplication gates the simulator is calling the functionality $\mathcal{F}_{mult}$ with inputs the shared values that have been already computed. According to the observation made in [ALR11], the adversary is able to select the summands of the shares that he possesses. The adversary sends those to the simulator who sets the corresponding summands in $\mathcal{F}_{mult}$. Then, the functionality outputs a sharing of the desired product, identically distributed with the real world sharing of the output, which is given to the adversary.
Again, the simulator observes any abort with some $B = \{p_j\}$ and sends the corresponding $(\mathsf{inform\ omission}, p_j)$ to $\mathcal{F}_{MPC}$, defaulting p_j's input. This will result in the functionality performing the computation with this default value and outputting $\perp$ to p_j.

Once every gate is computed, at the Output stage the sharing of the final output needs to be reconstructed. At this point, $\mathcal{F}_{recn}$ is called by the simulator with inputs the shared values in order to robustly reconstruct the output. By the properties of $\mathcal{F}_{recn}$ we know that in the hybrid world the correct output value is reconstructed towards all parties from the sharing of the values computed.

In the ideal world, the functionality $\mathcal{F}_{MPC}$ computes the desired function on the inputs of the alive parties and the input that the adversary has provided for the parties that he controls, while for zombie parties the default value is used.

After that, the functionality just sends to each honest party and to each omission-corrupted party p_i that is still alive the corresponding output value y_i. By the properties of the *MPC* protocol stated above, we get that the view of the adversary is indistinguishable between the real and the ideal execution, completing our proof. □

Necessity of condition for MPC Finally, in this last section we show that the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is necessary to securely achieve MPC.

Lemma 14. *If $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is violated, then there exist n-party functions which cannot be securely evaluated while tolerating (corruptions caused by) a $\mathcal{Z}$-adversary.*

Proof. (sketch) As we have already proven in 2.7 the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is necessary for broadcast. Also, in Section 3.2 we proved that the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_i, p_j)$ is necessary to securely exchange a message between a sender p_i and p_j. As our condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ considers all such pairs of players, and due to the fact that MPC implies both Broadcast and SMT, we get that our condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is necessary for MPC. □

4 Conclusion and Open Problems

We put forth the study of Byzantine agreement (BA) and multi-party computation (MPC) in the presence of (an adversary causing) omission and byzantine faults, (who is) described by a general adversary structure. We provided a complete characterization for BA tolerating such an adversary. We also provide a full characterization of MPC in this model, where we showed that existing techniques fall short in providing feasibility results. Notwithstanding, by introducing a new condition for Secure Message Transmission we were able to prove the first tight characterization for MPC in this setting. The above results makes an important step forward in understanding the relevant landscape and opens the floor to important questions that have long been resolved in the threshold adversary setting, e.g., allowing setup, error probability, and computationally bounded adversaries.

References

ACS22. Gilad Asharov, Ran Cohen, and Oren Shochat. Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In Dana Dachman-Soled, editor, *3rd Conference on Information-*

Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA, volume 230 of *LIPIcs*, pages 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.

AFM99. Bernd Altmann, Matthias Fitzi, and Ueli M. Maurer. Byzantine agreement secure against general adversaries in the dual failure model. In Prasad Jayanti, editor, *Distributed Computing, 13th International Symposium, Bratislava, Slovak Republic, September 27-29, 1999, Proceedings*, volume 1693 of *Lecture Notes in Computer Science*, pages 123–137. Springer, 1999.

ALR11. Gilad Asharov, Yehuda Lindell, and Tal Rabin. Perfectly-secure multiplication for any $t < n/3$. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 240–258. Springer, 2011.

AP99. Maha Abdallah and Philippe Pucheral. A low-cost non-blocking atomic commitment protocol for asynchronous systems. In *International Conference on Parallel and Distributed Systems, 1999, Proceedings*. IEEE, 1999.

BFH+08. Zuzana Beerliová-Trubíniová, Matthias Fitzi, Martin Hirt, Ueli M. Maurer, and Vassilis Zikas. MPC vs. SFE: perfect security in a unified corruption model. In Ran Canetti, editor, *Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008*, volume 4948 of *Lecture Notes in Computer Science*, pages 231–250. Springer, 2008.

BGP89. Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Towards optimal distributed consensus (extended abstract). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 410–415. IEEE Computer Society, 1989.

BGW88. Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 1–10. ACM, 1988.

BHR07. Zuzana Beerliová-Trubíniová, Martin Hirt, and Micha Riser. Efficient byzantine agreement with faulty minority. In Kaoru Kurosawa, editor, *Advances in Cryptology - ASIACRYPT 2007, 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007, Proceedings*, volume 4833 of *Lecture Notes in Computer Science*, pages 393–409. Springer, 2007.

Can00. Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptol.*, 13(1):143–202, 2000.

Can01. Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 136–145. IEEE Computer Society, 2001.

CCD88. David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 11–19. ACM, 1988.

CCGZ16. Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 240–269. Springer, 2016.

CDD+01. Ran Canetti, Ivan Damgård, Stefan Dziembowski, Yuval Ishai, and Tal Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In Birgit Pfitzmann, editor, *Advances in Cryptology - EUROCRYPT 2001, International Conference on the Theory and Application of Cryptographic Techniques, Innsbruck, Austria, May 6-10, 2001, Proceeding*, volume 2045 of *Lecture Notes in Computer Science*, pages 262–279. Springer, 2001.

CGZ23. Ran Cohen, Juan A. Garay, and Vassilis Zikas. Completeness theorems for adaptively secure broadcast. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I*, volume 14081 of *Lecture Notes in Computer Science*, pages 3–38. Springer, 2023.

DDWY93. Danny Dolev, Cynthia Dwork, Orli Waarts, and Moti Yung. Perfectly secure message transmission. *J. ACM*, 40(1):17–47, 1993.

ELT22. Karim Eldefrawy, Julian Loss, and Ben Terner. How byzantine is a send corruption? In Giuseppe Ateniese and Daniele Venturi, editors, *Applied Cryptography and Network Security - 20th International*

Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings, volume 13269 of *Lecture Notes in Computer Science*, pages 684–704. Springer, 2022.

FM88. Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In Janos Simon, editor, *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, pages 148–161. ACM, 1988.

Had85. Vassos Hadzilacos. *Issues of Fault Tolerance in Concurrent Computations (Databases, Reliability, Transactions, Agreement Protocols, Distributed Computing)*. PhD thesis, Harvard University, USA, 1985. AAI8520209.

HM97. Martin Hirt and Ueli M. Maurer. Complete characterization of adversaries tolerable in secure multiparty computation (extended abstract). In James E. Burns and Hagit Attiya, editors, *Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997*, pages 25–34. ACM, 1997.

HM00. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptol.*, 13(1):31–60, 2000.

HZ10. Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Monaco / French Riviera, May 30 - June 3, 2010. Proceedings*, volume 6110 of *Lecture Notes in Computer Science*, pages 466–485. Springer, 2010.

ISN89. Mitsuru Ito, Akira Saito, and Takao Nishizeki. Secret sharing scheme realizing general access structure. *Electronics and Communications in Japan (Part III: Fundamental Electronic Science)*, 72(9):56–64, 1989.

KMTZ13. Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 477–498. Springer, 2013.

Koo06. Chiu-Yuen Koo. Secure computation with partial message loss. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 502–521. Springer, 2006.

LP09. Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptol.*, 22(2):161–188, 2009.

LSP82. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

Mau02. Ueli M. Maurer. Secure multi-party computation made simple. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28. Springer, 2002.

PR03. Philippe Raipin Parvédy and Michel Raynal. 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*, page 212. IEEE Computer Society, 2003.

PSL80. Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

PT86. Kenneth J. Perry and Sam Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Software Eng.*, 12(3):477–482, 1986.

Ray02. Michel Raynal. 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*, pages 221–228. IEEE Computer Society, 2002.

Yao82. Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

ZHM09. Vassilis Zikas, Sarah Hauser, and Ueli M. Maurer. Realistic failures in secure multi-party computation. In Omer Reingold, editor, *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, volume 5444 of *Lecture Notes in Computer Science*, pages 274–293. Springer, 2009.

Zik10. Vassilis Zikas. *Generalized corruption models in secure multi-party computation*. PhD thesis, ETH Zurich, 2010.