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, Edinburgh, U.K.
K.Brazitikos@sms.ed.ac.uk

[2] Purdue University, West Lafayette, USA
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. All our results are for the classical synchronous model of computation.

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.

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

The full version of this paper can be found at the IACR Cryptology ePrint Archive, report 2024/209.

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.[1] Naturally, adversaries with fail-corruption allow for 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 Sect. 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

[1] 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.

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 (see discussion of MPC results in Sect. 2), translating threshold bounds to this setting is far from trivial—this reaffirms what the complex bounds of Beerliova *et al.* [5] demonstrate for the active/passive/fail case. Furthermore, existing arguments and techniques from the cryptographic literature are 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. 11) serves as a perfect demonstration of the technical challenges associated with devising such a bound, protocol, and associated tightness proof.

Finally, our results are proven secure in a simulation-based composable framework. Although we do not consider this to be our key technical contribution, it is, to our knowledge, a first both for general (mixed) adversary MPC and for MPC with omission corruptions. Our treatment demonstrates the challenges of a composable treatment of omission-faults. Therefore we believe it to be a milestone in the literature which can be of independent interest.

1.1 Related Literature

In this section we discuss the related literature, where we focus on synchronous[2] protocols with perfect security, i.e., with zero error probability, which is also the type of protocols we develop here.

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. Sects.3.6, 3.7). The seminal results by Lamport, Shostak, and Pease [27,32], 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

[2] We note that the feasibility questions discussed here have not been considered in any other model, e.g., asynchronous or partially synchronous; we consider this an interesting future direction.

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 [35] where feasibility of two-party computation was shown. The seminal works of Ben-Or, Goldwasser, and Wigderson [7] 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 [21,22] proved that a necessary and sufficient condition, if no setup[3] 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:[4]

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

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 [2] and a combination of fail-corruption and passive corruption by Beerliova *et al.* [5] (We refer to [36] for a comprehensive survey of the relevant literature).

The results from [5] offer a first demonstration of the unstranslatability of threshold feasibility results to the general adversary setting. Indeed, in the threshold setting, the active/passive/fail (tight) bound, i.e., $3t_a + 2t_p + t_f < n$ [18], is a simple combination of the corresponding active-only ($3t_a < n$), passive-only ($2t_p < n$), and fail-crash-only ($t_f < n$) bounds. On the other hand, in the general adversary setting, the tight (necessary and sufficient) bound is the combination of the following two conditions (each of them is necessary) [5, Theorem 1]:

$$\forall (A_i, E_i, F_i), (A_j, E_j, F_j), (A_k, E_k, F_k) \in \mathcal{Z} : E_i \cup E_j \cup A_k \cup (F_i \cap F_j \cap F_k) \neq \mathcal{P} \quad (2)$$

and

$$\forall (A_i, E_i, F_i), (A_j, E_j, F_j), (A_k, E_k, F_k) \in \mathcal{Z} : E_i \cup A_j \cup A_k \cup (F_j \cap F_k) \neq \mathcal{P}. \quad (3)$$

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

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

Each of the above triples (A, E, F), so-called adversary *classes*, describes the choice of the adversary specified by this class—namely the parties in A, E, and F, are actively, passively, and fail-corrupted, respectively.

In fact, the inability to translate threshold bounds to general adversaries is further demonstrated by the fact that if one is interested in non-reactive (one-shot) computation of a function, a problem often referred to as *Secure Function Evaluation (SFE)*, then the following *strictly weaker* (and substantially more complex) bound is necessary and sufficient [5, Theorem 2]: The bound from Eq. 2 together with the following condition

$$\begin{aligned}&\exists \text{ an ordering } (A_1, E_1, F_1), ..., (A_m, E_m, F_m) \text{ of the maximal classes in } \mathcal{Z} \text{ s.t.}\\ &\qquad \forall i, j, k \in \{1, ..., m\}, i \leq k : E_k \cup A_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P}. \quad (4)\end{aligned}$$

The above results demonstrate the untranslatability of threshold to general adversary results in the active/passive/fail setting. As we show in this work, a similar untranslatability—with even more counter-intuitive phenomena (see Sect. 2 for a discussion)—is evident also in our active/omission corruptions setting.

Omission Faults. The first variant of omissions was introduced to the distributed literature by Hadzilacos [20], 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 [33], 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 [1], 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 [20,33], 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 [31,34] 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 [26] 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 [37] 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 [36] by adding fail corruption.

More recently, Eldefrawy, Loss, and Terner [16] 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 [37] but for perfect security only. As demonstrated in [16] the shift to computational security carries unexpected complications, which is yet another indication of the challenges associated with omission-corruption. More concretely, [16] 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. They also proved this bound tight, albeit 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.

This lower bound was recently improved by Loss and Stern [29] to cover a worst-case adversary, i.e., without spotty send-omission corruptions. In fact, this seemingly simple generalization required developing novel techniques to deal with omissions, an additional indication of the challenges related to feasibility in the presence of active and omission corruptions.

We note in passing that, although in the threshold setting separating (full) omissions to send-omissions and receive-omissions helps to find tight feasibility bounds [16,29,37], this does not appear to be the case in the general adversary setting. Indeed, splitting omissions this way would complicate the description of the adversary structure—one would need two sets in each class to describe just omissions—and we conjecture this would also yield more complex and less intuitive bounds.

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 [7]. (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 [7,12,27], 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 [13]. 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 [10] put forth by Katz *et al.* [25]. 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, [25] 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 [25] 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 [25] 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. Due to the page constraint, we refer the reader to the full version [9] for proofs and other details.

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 a structure 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 $A^\star$ and $\Omega^\star$ 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 [37] 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 [3,11]. 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 [14,23].

1.3 Organization of the Paper

The remainder of the paper is organized as follows: Sect. 2 includes an exposition of our results and an overview of the techniques and related challenges. Section 3 includes the details on our tight feasibility results for BA and Sect. 4 our tight feasibility for MPC.

2 Technical Overview

Before diving into the technical part, it is useful to give an overview of our results and the associated techniques and challenges.

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:

$$C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \quad (5)$$

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 Sect. 3.2 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 Lemma 1) that the decision will always be correct as long as the following condition[5] is satisfied

$$C_{FIXR}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j \in \mathcal{Z} : \Omega_i \cup \Omega_j \neq \mathcal{P}, \quad (6)$$

which is also proven necessary for the above task in Lemma 3.2. 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 [37]: 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 the adversary. This makes it suitable for BA but insufficient for MPC (see below) 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.

Let us return to our overview of our BA feasibility result: 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 [8]–which was previously adapted to general adversary structures (with fail-corruption instead

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

of omissions) by Altmann, Fitzi, and Maurer [2]. Concretely, we gradually build protocols with stronger guarantees, from *Weak Consensus* (Sect. 3.3), to *Graded Consensus* (Sect. 3.4), to *King Consensus* (Sect. 3.5), and then iterate through different parties as kings to achieve the consistency and validity conditions of consensus (see Theorem 2).

The above similarity in the structure of our protocol to that from [2] might mislead the reader to believe that the search for the tight BA bound is straightforward given the above result. This is, however, far from true. To demonstrate this, it is useful to discuss the main challenge in shifting from a combination of active corruptions and fail-corruptions (for which we know tight bound both for BA [2] and for MPC [5]) 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. For instance, a standard method in the fail-corruption literature to limit the effect of fail-crashes is to embed a heartbeat 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 heartbeat 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 translating bounds from the threshold literature, or even the existing general adversary literature (i.e., without omission-corruptions) simply does not work. In fact, there is a number of ways that we demonstrate such a translation fails. For example, it is known that in the case of active-only general adversary structures, MPC is feasible if and only if the $Q^3_A(\mathcal{P}, \mathcal{Z})$ condition (Eq. 1) holds [21]. Hence, in search of a feasibility result, 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^3_Ω

$$CP^{(\Omega)}_{CONS}(\mathcal{P}, \mathcal{Z}) \Longleftrightarrow \forall \Omega_i, \Omega_j, \Omega_k \in \mathcal{Z} : \Omega_i \cup \Omega_j \cup \Omega_k \neq \mathcal{P}, \tag{7}$$

would be 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.

Similarly, drawing intuition from existing impossibility results can derail the search for lower bounds. Indeed, the general rule is that general-adversary impossibility results translate to threshold (though, not always in a trivial manner) but not the other way. Intuitively, the underlying reason is that the asymmetry of general adversary structures allows solutions which could never exist in a threshold setting. This untranslatability becomes ever more prominent when considering omission-corruptions (combined with active), and makes finding the tight condition on general structures for this case a far more challenging task than in the threshold case (in fact, it is challenging even given a tight threshold condition).

As an example, for active/passive adversaries the tight condition $3t_a + 2t_p < n$ [18] was "translated" in [19] to the general adversary setting as:

$$\forall (A_1, E_1), (A_2, E_2), (A_3, E_3) \in \mathcal{Z} : E_1 \cup E_2 \cup A_1 \cup A_2 \cup A_3 \neq \mathcal{P}, \tag{8}$$

(where sets A and E in the above bound correspond to actively and passively corrupted parties, respectively). But an analogous translation for active/omission adversaries of the tight threshold $3t_a + 2t_\omega < n$ [37] as

$$\forall (A_1, \Omega_1), (A_2, \Omega_2), (A_3, \Omega_3) \in \mathcal{Z} : \Omega_1 \cup \Omega_2 \cup A_1 \cup A_2 \cup A_3 \neq \mathcal{P}, \tag{9}$$

does *not* yield a bound necessary for MPC. In fact, in the appendix of the full version of the paper [9] we describe a structure that violates (an even more restrictive version of) the above condition but still allows for an MPC protocol.

In the same spirit, as we show in Sect. 4.1 of the full paper, common techniques used in the threshold MPC literature to recover from corruptions, such as *player elimination* [6], 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 [37] 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. 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 [15] (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 $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$, which in combination with $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ gives us the condition that is proven to be tight for DetSMT with sender p_s and receiver p_r. The $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ states that for any three $Z_i, Z_j, Z_k \in \mathcal{Z}$:

$$\begin{aligned}&\text{if } (p_s \in \Omega_i \cap \Omega_j \wedge p_r \in \Omega_k) \text{ OR } (p_r \in \Omega_i \cap \Omega_j \wedge p_s \in \Omega_k)\\&\text{then } A_i \cup A_j \cup \Omega_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}.\end{aligned} \tag{10}$$

The sufficiency of the above condition for detectable SMT and its necessity for (detectable) SMT (hence also for MPC) are proven in Sect. 4.1 (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 $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r)$ 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 $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ holds, where

$$C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \Leftrightarrow C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}) \wedge \forall p_s, p_r \in \mathcal{P} :\ C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_s, p_r). \tag{11}$$

The necessity of the above condition follows from the fact that both SMT and broadcast are special cases of MPC. In fact, since both the above are non-reactive functionalities, our impossibilities (along with the feasibility discussed below) imply that $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is tight for both (reactive) MPC and for SFE (i.e., non-reactive MPC.) Thus our results prove that unlike the active/passive/fail general adversary model where [5] proved a separation between the (tight feasibility bounds) for MPC and SFE, such a separation does not exist in the active/omission setting.

For the sufficiency we use the following idea: We modify the general adversary protocol from [5] by first projecting it to the active-corruption-only case (recall that [5] 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 [5] 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 [5] 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 [28].

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. Although we do not consider this to be our core technical contribution, 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 [5,21,37] 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.

Because the core novelty of our results is in the protocol constructions and proofs, to eliminate the technical burden put upon the reader in extracting the ideas from the simulation, we have employed a special proof structure: First we prove properties that our protocols have, akin to the traditional property-based approach used in the general adversary literature; subsequently, we describe our simulator and use the above properties, along with additional arguments wherever necessary, to argue perfect indistinguishability of real and ideal world.

3 Byzantine Agreement with Active and Omission Corruption

3.1 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,*

$$C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z}) \iff \forall Z_i, Z_j, Z_k \in \mathcal{Z} : A_i \cup A_j \cup A_k \cup (\Omega_i \cap \Omega_j) \neq \mathcal{P}. \tag{12}$$

The proof of the theorem is in 3.7 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).

3.2 Detection of Omission-Failures on Public Point-to-Point Communication

We begin by investigating 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 in the full version of the paper [9].

For our functionalities we follow the template of [13] 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* can be found in the full version of the paper [9].

Lemma 1. *If the condition $C^{(A,\Omega)}_{FIXR}(\mathcal{P},\mathcal{Z})$ (see Eq. 6) holds, the protocol* FixReceive *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{FR}$.*

To prove the above lemma, we will use the following properties of our protocol. i) 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. ii) Moreover, p_j might become a zombie only if p_j is omission-corrupted. From there, since we prove static security and these are public state protocols where all inputs are revealed by the functionality, the simulator needs to simply run a simulated copy of the protocol with these input. A formal simulation proof and proof of the properties can be found in the full version.

Proof. (sketch) According to the protocol, p_j becomes a zombie only if there exists no adversary class Z that could explain the $\perp$ messages p_j received. This guarantees that p_j is omission-corrupted because the $\perp$ messages he received cannot be explained in another way. Now, if p_j is alive and p_i is correct until the end of the protocol, p_j will output the correct value $x' = x$ because all the correct players will have received the x value from p_i and additionally the condition $C_{FIXR}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ ensures that there are enough of them to carry the correct value to p_j. On the opposite case where p_i was not correct, p_j is not guaranteed to reach a correct result. If there are conflicting values due to the malicious behavior of p_i, p_j will output $\perp$. □

We also prove in the full paper 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. 6) does not hold, then no protocol can satisfy the properties of the *FixReceive* protocol stated above.

3.3 Weak Consensus

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 $\mathcal{F}_{WC}$ in the full paper [9] 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 "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 "$\mathsf{n/v}$".

Lemma 2. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ (see Eq. 12) holds, the protocol* WeakConsensus *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{WC}$.*

To prove this lemma we will make use of the following properties of our protocol. *(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 $y = x$. The full proof can be found in the full version of the paper [9].

Proof. (sketch) First of, we can prove that the selection of the output value y_j is unique, assuming that there can be two that satisfy the conditions and reaching a contradiction. After that, we can prove the weak consistency and persistency properties, by using the sets $P_j^{(\perp,0,1)}$ which split the whole player set according to the values that where received. After that, we can use the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ together with the requirements of the protocol for choosing y_j in order to show that the desired properties hold true. This is done after a lengthy separation of the sets of players and a case study of what values they can send. $\square$

3.4 Graded Consensus

The next step towards our goal 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 [17]. 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 $\mathcal{F}_{GC}$ presented in detail in the full paper [9] 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 $x'_i \in \{y_j, \text{“n/v”}\}$). 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.

Lemma 3. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* GradedConsensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{GC}$.

To prove this lemma we will make use of the following properties of our protocol. *(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. (sketch) First of all we can prove that for any player $p_j \in \mathcal{P} \setminus A^\star$, $y_j = 1$ only when a condition respective to the case of $y_j = 0$ holds. After that, in a similar manner as in Weak Consensus we can prove the properties of *graded consistency* and *graded persistency*, leveraging the properties of Weak Consensus. All details can be found in the formal proof in the full paper [9]. □

3.5 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 this is described in detail in the full paper [9]. The functionality guarantees that persistency is kept 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 that are 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.

Lemma 4. *If the condition* $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 12) holds, the protocol* KingConsensus *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{KC}$.

To prove this lemma we make use of the following properties of our protocol. *(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 $y_j = x$. The formal proof can be found in the full paper [9].

3.6 Consensus

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 in detail in the full paper, 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.

Theorem 2. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 12) 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 repeats a finite number of times a terminating protocol, so it is guaranteed to terminate. □

3.7 Broadcast

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 $\mathcal{F}_{BC}$. The proof can be found in the full paper and is mainly derived from the properties of *Consensus*.

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*$(\mathcal{P}, \mathcal{Z}, (x_1, \dots, x_n))$ 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*$(\mathcal{P}, \mathcal{Z}, (b_1, \dots, b_n))$.
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. 12) holds, the protocol* Broadcast *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{BC}$.

Table 1. The classes Z_1, Z_2, Z_3 with $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$.

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

Necessity of Conditions for Byzantine Agreement. Next, we show that the $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ condition is also necessary for Broadcast.

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. For a more detailed proof with further discussion about why older techniques wouldn't work see the full version [9].

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.*

Proof. (sketch) Assuming that the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$(12) does not hold and that we have secure broadcast we will reach a contradiction. This means that $\exists\ Z_1, Z_2, Z_3$ such that $A_1 \cup A_2 \cup A_3 \cup (\Omega_2 \cap \Omega_3) = \mathcal{P}$. Using a player-simulation argument, we get three scenarios that are indistinguishable for the players, using the adversary structure shown in Table 1. 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.

In the first scenario (see Fig. 1), p_1 is actively corrupted and all other players are honest. He sends different values to p_2 and p_3 and nothing to p_4. This makes p_2 believe 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.

In the second scenario (see Fig. 2), p_2 is actively corrupted and p_4 is omission-corrupted. The sender sends his input 1 to p_3 but is blocked from reaching p_4. At the same time the adversary is using p_2 to claim that the sender sent him the value 0. This makes p_2 believe 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.

In the third scenario (see Fig. 3), p_3 is actively corrupted and p_4 is omission-corrupted. The sender sends his input 0 to p_2 but is blocked from reaching p_4. At the same time the adversary is using p_3 to claim that the sender sent him the value 1. This makes p_2 believe 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

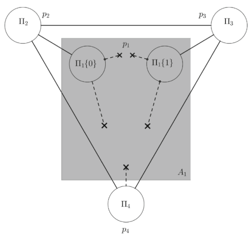

Fig. 1. p_1 is actively corrupted.

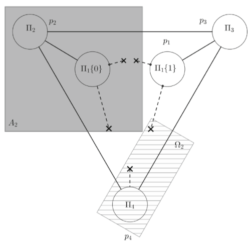

Fig. 2. p_2 is actively, p_4 is omission-corrupted.

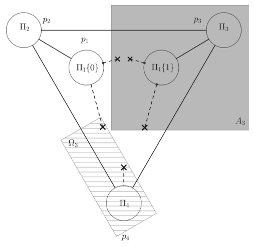

Fig. 3. p_3 is actively, p_4 is omission-corrupted.

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 p_4 the three scenarios are indistinguishable. Hence, the output should be 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 for Broadcast, meaning that the desired properties hold true if and only if our condition holds. In the full paper [9] we show that this condition is tight for Consensus as well. □

4 Multi-party Computation

In this section we extend our study to multi-party computation. A discussion about the challenges of such an extension, and a showcase that existing techniques from the threshold literature either do not work, or yield counter-intuitive results can be found in the full version of the paper [9]. 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 Sect. 4.1. 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 Sect. 4.2.

Next, we tackle one of the final problems, namely securely computing the gates in Sect. 4.3, 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 Sect. 4.4 to present our full MPC protocol and in Sect. 4.4 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[6].

4.1 Detectable Secure Message Transmission

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.

To achieve this, the condition $C_{BA}^{(A,\Omega)}(\mathcal{P},\mathcal{Z})$ of broadcast is no longer sufficient. On top of it, we need the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P},\mathcal{Z},P_s,P_r)$ for this pair of parties, as explained in 2.

In more detail, we have the sender P_s who wants to send a message $s \in \mathbb{F}$ to the receiver P_r. The functionality dictates that this is accomplished without leaking any information to the adversary. If no input is received by the functionality from P_s due to his fault (i.e., $P_s \in \Omega^\star$ or $A^\star$) then $\mathcal{F}_{DetSMT}$ sends a default message "$\mathsf{n/v}$" to P_r. If P_r is omission-corrupted and is unable to receive the intended value from $\mathcal{F}_{DetSMT}$, he outputs a special symbol $\perp$ to denote this.

The functionality that captures our goal is presented in detail in the full version of the paper [9]. 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 Sect. 3.2; 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

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

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 complement 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.

Lemma 6. *If the condition* $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ *(see Eq. 11) holds, the protocol* DetSMT *perfectly* $\mathcal{Z}$*-securely realizes the functionality* $\mathcal{F}_{detSMT}$*.*

To prove this lemma we will make use of the following properties of our protocol. Either the protocol aborts with some set B of corrupted parties or it terminates and we have the following properties: 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 $P_s \in A^\star$. Also, P_r might become a zombie only if he is omission-corrupted. Furthermore, no information on s is leaked to the adversary. The complete proof can be found in the full paper [9]. One thing that we need to point out is the following caveat. Extra care needs to be taken in order to properly describe the simulation for the functionality, because the simulator has to act differently based on whether the adversary has access to some summand of the secret value or not. This subtlety is expanded upon in the full proof.

Necessity of SMT Condition. Additionally, we prove that the condition $C_{SMT}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z}, p_1, p_2)$ is actually necessary for the (plain, not detectable) $\mathcal{F}_{SMT}$

functionality that enables p_1 to securely send a message to a receiver p_2, making our result tight (both sufficient and necessary).

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

As before, further details and the complete proof can be found in the full version of the paper [9].

4.2 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 [28].

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 in the full version of the paper [9]. We can implement this through the broadcast of the bit b by the player in question. 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 properties 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 broadcast protocol.

Lemma 8. *If the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 12) holds, the protocol* Heartbeat *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{Hb}$.*

The proof is derived from the *Broadcast* properties. The complete version can be found in the full paper [9].

Verifiable Secret Sharing. A very important primitive that is essential in keeping the privacy of the players' input is called Verifiable Secret Sharing (VSS). On top of guaranteeing that the input of the player is kept secret, we also get that all players agree on the way that the initial value is shared among them. By splitting the message s in random summands s_k using a sum share and then giving each one of those to the corresponding set S_k we achieve the privacy property Furthermore, by having the players in each S_k cross check their values we get the verifiability property. This idea was first developed in [24] and has since been used in many MPC protocols.

The functionality that we want to instantiate is presented in detail in the full version of the paper [9]. To give a brief description, it takes as input a value s that needs to be secret shared. 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 given 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. However, all players in each S_k still get the same value s_k. 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. Next, the players cross check their shares to detect inconsistencies. If that is the case for some s_k, a special message (CONTRAST, k) is broadcast and the dealer p_d broadcasts the correct summand in the open. Finally, the players invoke a Heartbeat to communicate to all if someone became a zombie.

Lemma 9. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) 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* VSS *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{VSS}$.*

To prove this lemma we will make use of the following properties of our protocol. *(correctness)* VSS 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.
The complete proof can be found in the full version of the paper [9].

Announce and Reconstruct. The functionalities $\mathcal{F}_{ann}$ and $\mathcal{F}_{recn}$ for the Announce and Reconstruct primitives are given in the full version of the paper [9]. 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.

Lemma 10. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) 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 $\mathcal{F}_{ann}$.*

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.

Lemma 11. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) 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 $\mathcal{F}_{recn}$.*

The complete proofs can be found in the full version of the paper [9].

4.3 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 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 are consistent. As the parties can locally compute addition gates without any communication involved, we omit the rest from the main body and point to the full version of the paper [9] for more details.

Multiplication. 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 $\mathcal{F}_{mult}$ provided in the full version of the paper [9].

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 [4]. 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 a correct player with a default sharing.

Finally, we show that there always exists a non-actively corrupted player having both s_k and t_ℓ, from the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$. Due to that, 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 the protocol *Mult* and it is based on the respective protocols of [5,30]. 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.,

$$st := \sum_{k=1}^{|\mathcal{S}|} \sum_{\ell=1}^{|\mathcal{S}|} s_k t_\ell = \sum_{k,\ell=1}^{|\mathcal{S}|} s_k t_\ell. \tag{13}$$

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.

Lemma 12. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) 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 $\mathcal{F}_{mult}$.*

To prove this lemma we will make use of the following properties of our protocol. *(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. The complete proof can be found in the full version of the paper [9].

4.4 The MPC Protocol

We next proceed to the construction of our *MPC* protocol, which securely realizes the functionality $\mathcal{F}_{MPC}$, detailed in the full version [9]. 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 $\mathcal{P} := \mathcal{P} \setminus B$, 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.

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 needs to be evaluated, 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 $\mathit{Mult}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle s\rangle, \langle t\rangle)$ 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 $\mathit{Reconstruct}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, \langle v\rangle)$ 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.

Theorem 4. *If the condition $C_{MPC}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ (see Eq. 11) holds, the protocol* MPC *perfectly $\mathcal{Z}$-securely realizes the functionality $\mathcal{F}_{MPC}$.*

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

Lemma 13. *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 discussed in 3.7 the condition $C_{BA}^{(A,\Omega)}(\mathcal{P}, \mathcal{Z})$ is necessary for broadcast. Also, in Sect. 4.1 we stated 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. □

5 Conclusion and Open Problems

We put forth the study of Byzantine agreement (BA) and multi-party computation (MPC) in the presence of a mixed general adversary with active and omission corruptions. We provided a tight characterization—necessary and sufficient conditions—for feasibility of synchronous BA—both for broadcast and consensus—tolerating such an adversary. We also provide a tight characterization of feasibility of MPC in this model, where we show that existing techniques fall short in providing feasibility results. Along the way, we also provide the first tight feasibility result for (detectable) Secure Message Transmission (SMT) in this model; by repeating upon failure while excluding the detected party, this yields the first tight feasibility result for SMT in this model. The above results make an important step forward in understanding the relevant landscape and open the floor to follow-up questions that have been resolved in the threshold adversary setting, but are wide open in the general adversary setting, e.g., allowing setup, error probability, and computationally bounded adversaries.

Acknowledgments. The authors would like to thank the anonymous reviewers for their helpful comments on the manuscript, as well as Giorgos Panagiotakos for discussions in the early stages of this work. The research was done in part while Konstantinos Brazitikos was visiting Purdue University and while Vassilis Zikas was at the University of Edinburgh. Konstantinos Brazitikos was supported in part by Input Output (iohk.io) through their funding of the Edinburgh Blockchain Technology Lab and Sunday Group. Vassilis Zikas was supported in part by Sunday Group, IOHK, NSF grants no. 2055599 & 2001096, and BSF grant no. 41000926.

References

1. Abdallah, M., Pucheral, P.: A low-cost non-blocking atomic commitment protocol for asynchronous systems. In: International Conference on Parallel and Distributed Systems, 1999, Proceedings. IEEE (1999)
2. Altmann, B., Fitzi, M., Maurer, U.M.: Byzantine agreement secure against general adversaries in the dual failure model. In: Jayanti, P. (ed.) Distributed Computing, 13th International Symposium, Bratislava, Slovak Republic, September 27-29, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1693, pp. 123–137. Springer (1999). https://doi.org/10.1007/3-540-48169-9_9
3. Asharov, G., Cohen, R., Shochat, O.: Static vs. adaptive security in perfect MPC: A separation and the adaptive security of BGW. In: Dachman-Soled, D. (ed.) 3rd Conference on Information-Theoretic Cryptography, ITC 2022, July 5-7, 2022, Cambridge, MA, USA. LIPIcs, vol. 230, pp. 15:1–15:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/LIPIcs.ITC.2022.15
4. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-secure multiplication for any t ¡ $n/3$. In: Rogaway, P. (ed.) Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6841, pp. 240–258. Springer (2011).https://doi.org/10.1007/978-3-642-22792-9_14
5. Beerliová-Trubíniová, Z., Fitzi, M., Hirt, M., Maurer, U.M., Zikas, V.: MPC vs. SFE: perfect security in a unified corruption model. In: Canetti, R. (ed.) Theory of Cryptography, Fifth Theory of Cryptography Conference, TCC 2008, New York, USA, March 19-21, 2008. Lecture Notes in Computer Science, vol. 4948, pp. 231–250. Springer (2008).https://doi.org/10.1007/978-3-540-78524-8_14
6. Beerliová-Trubíniová, Z., Hirt, M., Riser, M.: Efficient byzantine agreement with faulty minority. In: Kurosawa, K. (ed.) 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. Lecture Notes in Computer Science, vol. 4833, pp. 393–409. Springer (2007).https://doi.org/10.1007/978-3-540-76900-2_24
7. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 1–10. ACM (1988). https://doi.org/10.1145/62212.62213
8. Berman, P., Garay, J.A., Perry, K.J.: 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, pp. 410–415. IEEE Computer Society (1989).https://doi.org/10.1109/SFCS.1989.63511

9. Brazitikos, K., Zikas, V.: General adversary structures in byzantine agreement and multi-party computation with active and omission corruption. Cryptology ePrint Archive, Paper 2024/209 (2024). https://eprint.iacr.org/2024/209
10. Canetti, R.: 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, pp. 136–145. IEEE Computer Society (2001). https://doi.org/10.1109/SFCS.2001.959888
11. Canetti, R., Damgaard, I., Dziembowski, S., Ishai, Y., Malkin, T.: On adaptive vs. non-adaptive security of multiparty protocols. In: Pfitzmann, B. (ed.) EUROCRYPT 2001. LNCS, vol. 2045, pp. 262–279. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-44987-6_17
12. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 11–19. ACM (1988). https://doi.org/10.1145/62212.62214
13. Cohen, R., Coretti, S., Garay, J., Zikas, V.: Probabilistic termination and composability of cryptographic protocols. In: Robshaw, M., Katz, J. (eds.) CRYPTO 2016. LNCS, vol. 9816, pp. 240–269. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-53015-3_9
14. Cohen, R., Garay, J.A., Zikas, V.: Completeness theorems for adaptively secure broadcast. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part I. Lecture Notes in Computer Science, vol. 14081, pp. 3–38. Springer (2023).https://doi.org/10.1007/978-3-031-38557-5_1
15. Dolev, D., Dwork, C., Waarts, O., Yung, M.: Perfectly secure message transmission. J. ACM **40**(1), 17–47 (1993). https://doi.org/10.1145/138027.138036
16. Eldefrawy, K., Loss, J., Terner, B.: How byzantine is a send corruption? In: Ateniese, G., Venturi, D. (eds.) Applied Cryptography and Network Security - 20th International Conference, ACNS 2022, Rome, Italy, June 20-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13269, pp. 684–704. Springer (2022). https://doi.org/10.1007/978-3-031-09234-3_34
17. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: Simon, J. (ed.) Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA, pp. 148–161. ACM (1988). https://doi.org/10.1145/62212.62225
18. Fitzi, M., Hirt, M., Maurer, U.: Trading correctness for privacy in unconditional multi-party computation. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 121–136. Springer, Heidelberg (1998). https://doi.org/10.1007/BFb0055724
19. Fitzi, M., Hirt, M., Maurer, U.: General adversaries in unconditional multi-party computation. In: Lam, K.-Y., Okamoto, E., Xing, C. (eds.) ASIACRYPT 1999. LNCS, vol. 1716, pp. 232–246. Springer, Heidelberg (1999). https://doi.org/10.1007/978-3-540-48000-6_19
20. Hadzilacos, V.: Issues of Fault Tolerance in Concurrent Computations (Databases, Reliability, Transactions, Agreement Protocols, Distributed Computing). Ph.D. thesis, Harvard University, USA (1985), aAI8520209
21. Hirt, M., Maurer, U.M.: Complete characterization of adversaries tolerable in secure multi-party computation (extended abstract). In: Burns, J.E., Attiya, H. (eds.) Proceedings of the Sixteenth Annual ACM Symposium on Principles of Distributed Computing, Santa Barbara, California, USA, August 21-24, 1997, pp. 25–34. ACM (1997). https://doi.org/10.1145/259380.259412

22. Hirt, M., Maurer, U.M.: Player simulation and general adversary structures in perfect multiparty computation. J. Cryptol. **13**(1), 31–60 (2000). https://doi.org/10.1007/s001459910003
23. Hirt, M., Zikas, V.: Adaptively secure broadcast. In: Gilbert, H. (ed.) 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. Lecture Notes in Computer Science, vol. 6110, pp. 466–485. Springer (2010). https://doi.org/10.1007/978-3-642-13190-5_24
24. Ito, M., Saito, A., Nishizeki, T.: Secret sharing scheme realizing general access structure. Electron. Commun. Japan (Part III: Fundamental Electronic Science) **72**(9), 56–64 (1989)
25. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36594-2_27
26. Koo, C.: Secure computation with partial message loss. In: Halevi, S., Rabin, T. (eds.) Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings. Lecture Notes in Computer Science, vol. 3876, pp. 502–521. Springer (2006). https://doi.org/10.1007/11681878_26
27. Lamport, L., Shostak, R.E., Pease, M.C.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
28. Lindell, Y., Pinkas, B.: A proof of security of yao's protocol for two-party computation. J. Cryptol. **22**(2), 161–188 (2009). https://doi.org/10.1007/s00145-008-9036-8
29. Loss, J., Stern, G.: Zombies and ghosts: Optimal byzantine agreement in the presence of omission faults. In: Rothblum, G.N., Wee, H. (eds.) Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV. Lecture Notes in Computer Science, vol. 14372, pp. 395–421. Springer (2023).https://doi.org/10.1007/978-3-031-48624-1_15
30. Maurer, U.M.: Secure multi-party computation made simple. In: Cimato, S., Galdi, C., Persiano, G. (eds.) Security in Communication Networks, Third International Conference, SCN 2002, Amalfi, Italy, September 11-13, 2002. Revised Papers. Lecture Notes in Computer Science, vol. 2576, pp. 14–28. Springer (2002). https://doi.org/10.1007/3-540-36413-7_2
31. Parvédy, P.R., Raynal, M.: Uniform agreement despite process omission failures. In: 17th International Parallel and Distributed Processing Symposium (IPDPS 2003), 22-26 April 2003, Nice, France, CD-ROM/Abstracts Proceedings, p. 212. IEEE Computer Society (2003). https://doi.org/10.1109/IPDPS.2003.1213388
32. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980). https://doi.org/10.1145/322186.322188
33. Perry, K.J., Toueg, S.: Distributed agreement in the presence of processor and communication faults. IEEE Trans. Software Eng. **12**(3), 477–482 (1986). https://doi.org/10.1109/TSE.1986.6312888
34. Raynal, M.: Consensus in synchronous systems: A concise guided tour. In: 9th Pacific Rim International Symposium on Dependable Computing (PRDC 2002), 16-18 December 2002, Tsukuba-City, Ibarski, Japan, pp. 221–228. IEEE Computer Society (2002). https://doi.org/10.1109/PRDC.2002.1185641

35. Yao, A.C.: Protocols for secure computations (extended abstract). In: 23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982, pp. 160–164. IEEE Computer Society (1982). https://doi.org/10.1109/SFCS.1982.38
36. Zikas, V.: Generalized corruption models in secure multi-party computation. Ph.D. thesis, ETH Zurich (2010). https://d-nb.info/1005005729
37. Zikas, V., Hauser, S., Maurer, U.: Realistic failures in secure multi-party computation. In: Reingold, O. (ed.) TCC 2009. LNCS, vol. 5444, pp. 274–293. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-00457-5_17