Diss. ETH No. 18913

Generalized Corruption Models in Secure Multi-Party Computation

A dissertation submitted to

ETH ZURICH

for the degree of
Doctor of Sciences

presented by

Vassilis Zikas
Dipl., National Technical University of Athens

born July 11, 1981, in Athens, Greece
citizen of Greece

accepted on the recommendation of

Prof. Dr. Ueli Maurer, examiner
Prof. Dr. Rafail Ostrovsky, co-examiner
Prof. Dr. Stathis Zachos, co-examiner
Dr. Martin Hirt, co-examiner

2010

Acknowledgments

First I would like to thank my advisor Ueli Maurer for giving me the opportunity to work on the fascinating world of cryptography and introducing me to his unique abstract thinking, and for being a supportive and motivating force through my PhD studies.

Further, I wish to thank my co-referee Rafi Ostrovsky for reviewing my thesis and giving me useful feedback.

Special thanks go to my co-referee Martin Hirt for his valuable guidance and his patience during our endless enlightening discussions, as well as for teaching me several things about doing research and presenting my results. Furthermore, I would like to thank my co-referee Stathis Zachos, for supporting me from the very beginning of my studies and for putting me into the track of theoretical research.

I would also like to thank my co-author and office-mate Zuzana Beerliova, for being a great office-mate and collaborator. Moreover, I want to thank my friend and colleague Stefano Tessaro and, also, Björn Tackmann for helping me translate the abstract in German, as well as the rest of my colleagues and collaborators: Matthias Fitzi, Christoph Lucas, Divesh Aggarwal, Peter Gazi, Thomas Holenstein, Renato Renner, Junji Shikata, Stefan Wolf, Dominik Raub, and all those that I forgot to mention for many inspiring discussions.

Finally, many thanks go to my family for their unconditional support during my studies, and also to my two dearest persons Ilias and Mariza for always being there for me.

Abstract

Secure multi-party computation (MPC) allows a set of players to perform a joint computation on their private inputs in a secure way. The security of the computation should be guaranteed even when some players misbehave. The misbehavior of players is modeled by assuming a central adversary who corrupts certain players and uses them to attack the computation. Clearly, the more power we give to the adversary, the harder it becomes to achieve the required security.

The power of the adversary can be specified by several parameters. The most typical parameters are: the considered corruption types, the description of the set of corrupted players, and the way in which this set is chosen. The most common corruption-types are *active* corruption (the adversary has full control over the player), *passive* corruption (the adversary sees whatever the player sees), and *fail*-corruption (the adversary can force the player to crash). With respect to the description of the corrupted set, one typically considers either a *threshold* adversary, described by an upper bound on the total number of corrupted players, or a *general* adversary, described by an enumeration of all corruption-combinations. Finally, with respect to the way in which the corrupted players are chosen, the adversary can be *static* or *adaptive*. A static adversary chooses the set of corrupted players at the beginning, whereas an adaptive adversary can corrupt players during the computation. In this thesis, we consider generalizations of the adversary's power with respect to all three parameters.

With respect to the description of the set of corrupted players, we consider a general adversary who can, simultaneously, actively corrupt, passively corrupt, and fail-corrupt players, and prove exact feasibility bounds for MPC. An interesting effect which shows up when all three corruption-types and a general adversary are considered is a separation between reactive and non-reactive computation. In particular, we show that there are adversaries who

can be tolerated for non-reactive computation, also known as secure function evaluation (SFE), but not for reactive computation.

With respect to the ways in which the players can be corrupted, we consider an alternative corruption type, called omission-corruption, which models network failures in a realistic way. Omission-corruption allows the adversary to selectively block messages sent to and from the corrupted players. We consider omission-corruption in combination with active corruption and fail-corruption, and prove exact feasibility bounds for MPC and SFE tolerating a threshold adversary.

Finally, we observe that allowing the adversary to adaptively choose the players to corrupt, brings up a simulatability/composability issue with the property-based definition of Broadcast. In particular, in the presence of such an adversary, even when only active corruption is considered, Broadcast protocols which are proved secure with respect to the property-based definition do not securely realize the natural Broadcast functionality. We investigate the source of the problem and construct new *adaptively secure* Broadcast protocols. The corresponding bounds are proved exact for adaptive security.

The presented results consider all three security levels, i.e., perfect, statistical, and computational security.

Zusammenfassung

Sichere Multi-Party-Berechnungen (MPC) ermöglichen es einer Menge von Spielern, gemeinsam eine beliebige Berechnung auf ihren geheimen Inputs sicher durchzuführen. Die Sicherheit der Berechnung sollte auch dann garantiert sein, wenn einige Teilnehmer vom Protokoll abweichen. Dieses Fehlverhalten wird durch einen zentralen Gegner modelliert, der bestimmte Spieler korrumpiert und mit ihnen versucht die Berechnung zu verfälschen. Natürlich wird es schwieriger, die erforderliche Sicherheit zu garantieren, wenn dem Gegner mehr Möglichkeiten gegeben werden.

Die Möglichkeiten des Gegners werden durch mehrere Parameter beschrieben. Die üblichsten Parameter sind verschiedene Typen von Korrumpierung, die Beschreibung der Menge korrumpierter Spieler sowie die Art, in der diese Menge gewählt wird. Als Typen von Korrumpierung werden häufig *aktive* Korrumpierung (der Gegner kontrolliert den Spieler vollständig), *passive* Korrumpierung (der Gegner sieht alles, was der Spieler sieht) und *fail*-Korrumpierung (der Gegner kann den Spieler abstürzen lassen) betrachtet. Die Menge der korrumpierten Spieler wird typischerweise entweder durch eine Schwelle beschrieben, also eine obere Schranke für die Anzahl korrumpierter Spieler, oder für einen allgemeinen Gegner durch eine Aufzählung aller gültigen Kombinationen von gleichzeitig korrumpierten Spielern. Schließlich kann der Gegner bezüglich der Bestimmung der Menge entweder *statisch* oder *adaptiv* sein. Ein statischer Gegner wählt die Menge korrumpierter Spieler zu Beginn, ein adaptiver Gegner kann Spieler während der Berechnung korrumpieren. In dieser Dissertation betrachten wir Verallgemeinerungen der Möglichkeiten des Gegners bezüglich aller drei Parameter.

Bezüglich der Beschreibung der Menge korrumpierter Spieler betrachten wir einen Gegner der gleichzeitig Spieler aktiv, passiv, und fail-korrumpieren

kann, und wir beweisen exakte Schranken für die Durchführbarkeit von MPC. Ein interessanter Effekt, der bei der Betrachtung aller drei Arten von Korrumpierung mit einem allgemeinen Gegner auftritt, ist eine Separierung zwischen reaktiven und nicht-reaktiven Berechnungen. Im Speziellen zeigen wir die Existenz von Gegnern, die bei nicht-reaktiven Berechnungen, also sicheren Funktionsauswertungen (SFE), toleriert werden können, jedoch nicht bei reaktiven Berechnungen.

Mit Bezug auf die Typen der Korrumpierung betrachten wir einen alternativen Typ, die *omission*-Korrumpierung, durch die Netzwerkfehler realitätsnah modelliert werden. Omission-Korrumpierung erlaubt es dem Gegner, Nachrichten, die zu oder von einem korrumpierten Spieler gesendet werden, selektiv zu blockieren. Wir betrachten die Kombination von omission-Korrumpierung mit aktiver und fail-Korrumpierung und beweisen exakte Durchführbarkeitsschranken für MPC und SFE mit einem durch einen Schwellwert beschriebenen Gegner

Zuletzt beobachten wir, dass durch die Möglichkeit der adaptiven Korrumpierung ein Problem bei der Simulierbarkeit/Komposition in der eigenschaftsbasierten Definition von Broadcast auftritt. Sogar wenn bei einem adaptiven Gegner nur aktive Korrumpierung betrachtet wird, implementieren Broadcast-Protokolle, die bezüglich der eigenschaftsbasierten Definition von Broadcast als sicher bewiesen wurden, nicht die natürliche Funktionalität von Broadcast. Wir untersuchen die Entstehung des Problems und konstruieren neue, *adaptiv sichere* Broadcast-Protokolle. Wir beweisen dass die Schranken bezüglich adaptiver Sicherheit exakt sind.

Die gezeigten Ergebnisse betrachten alle drei Ebenen von Sicherheit, also perfekte, statistische und berechenmäßige Sicherheit.

Contents

1 Introduction **13**

1.1 Multi-Party Computation 14

1.2 Misbehavior and Adversary 15

1.3 Reactive and Non-reactive Computation: (MPC and SFE) 20

1.4 Byzantine Agreement 21

1.5 Typical Assumptions 21

1.6 Relevant Literature 24

1.7 Contributions of this Thesis 26

2 Setting the Landscape **29**

2.1 Security of Multi-Party Protocols 29

2.2 Corruptions and Guarantees 32

2.3 General Adversary 33

2.4 Characterizing the Players 36

2.5 The Computation 36

2.6 The Network 37

2.7 Computational Assumptions 38

2.8 Digital Signatures and Commitments 38

I MIXED GENERAL ADVERSARY 41

3 Multi-Party Computation 43
3.1 Outline . . . 45
3.2 MPC and/vs. SFE . . . 45
3.3 Passive/Fail Adversary with Perfect Security . . . 48
3.4 Adding Active Corruption . . . 59
3.5 Statistical Security . . . 67
3.6 Computational Security . . . 82
3.7 Summarizing . . . 85

4 Byzantine Agreement 87
4.1 Outline . . . 88
4.2 Consensus and Broadcast . . . 88
4.3 Perfect Security (no setup) . . . 93
4.4 Statistical and Computational Security (with setup) . . . 102
4.5 Composability Considerations . . . 119

II ALTERNATIVE CORRUPTION TYPES 123

5 Omission-Corruption 125
5.1 Outline . . . 126
5.2 The Functionality . . . 126
5.3 The SFE Protocol – A High-Level Description . . . 128
5.4 Engineering the Network . . . 131
5.5 Detectable SFE . . . 139
5.6 Robust SFE . . . 140
5.7 Necessity . . . 145
5.8 (Reactive) MPC . . . 147
5.9 Statistical and Computational Security . . . 147

III ADAPTIVE CORRUPTION **153**

6 Adaptively Secure Broadcast **155**

6.1 Outline 156
6.2 (In)Composability of the Property-Based Definition 156
6.3 The Model 158
6.4 The Broadcast Functionality 159
6.5 Perfect Security (without setup) 163
6.6 Statistical and Computational Security (with setup) 164
6.7 Other Models 172

Chapter 1

Introduction

The first thing that comes in mind when one hears the term cryptology is the words encryption and decryption. Indeed, for many years securing communication has been the exclusive goal of cryptology, and the discipline was visualized as a strangle between the "code-makers" trying to create better encryption-algorithms and the "code-breakers" trying to brake them. During the last decades, the vast development of communication networks and computerized systems has made the need for securing communication even more prominent, but has also brought up the need for additional security guarantees such as preserving the privacy of the exchanged data. New questions have emerged of the type: "Is it possible to have a secure electronic voting scheme?", or "could some company compute market statistics jointly with its competitors without revealing its clients' sensitive data?". Motivated by such questions, the field of multi-party computation was developed, which still occupies a big part of the cryptologic research.

The intuition behind multi-party computation can be best illustrated by an example. Bill and Steve are two millionaires, each of whom claims to be richer than the other, and they would like to find out who is right. The easiest way to do so would be to tell each other the exact amount of money they own. However, this information is secret and none of them is willing to share it with the other. Still they are both curious in learning who is richer. Clearly, this task would be trivial if there would be a third entity/party, e.g. a judge, which both millionaires trust to learn the amounts of money they own, compare them, and announce the identity of the richest. Indeed, in such a case, both millionaires could whisper their respective amounts of money to the

judge's ear, who could then find out who is richer and announce it. However, as such fully trusted and ideally operating parties are "hard to find", an interesting problem is to design a protocol which allows the millionaires to perform this task without the help of such a third party, by simply talking to each other. Ideally, such a protocol should not reveal anything more to any of the millionaires than the answer to the question "who is richer?". It is problems of this type that multi-party computation (in this example there are only two parties) attempts to answer.

The above example could trivially be extended to the case of $n > 2$ parties/millionaires who wish to find out who is richer. What are the security-properties that a protocol solving this task should guarantee? To start with, every party/millionaire should learn the identity of the richest party. In other words, the output of the protocol to each party should be correct. Moreover, the protocol should allow every millionaire to learn nothing else about the fortune of the other participants, other than the identity of the richest party. The above properties capture the essence of the security definition of multi-party computation, which we describe more concretely in the following section.

1.1 Multi-Party Computation

Generalizing the millionaires example, in the context of multi-party computation (MPC) we have n parties, called the *players*, each of which knows some secret value, and their goal is to perform a computation on these values in a "secure" way. In the millionaires example the secret values are the amounts of money each player/millionaire owns, and the goal is to compute the identity of the richest player. The security requirements are that every player receives the correct output of the computation on input the values given by the players (correctness), and that he gets no more information than what he can deduce/compute from this output (privacy). As demonstrated in the millionaires example, such a computation could be easily performed if a fully trusted third party would be available to the players. Motivated by this observation, a more accurate interpretation of what it means for a protocol which performs some multi-party computation to be secure, can be specified in two steps: (1) Define a primitive whose behavior captures what one would *ideally* expect from the computation, i.e., define how the computation would be performed by an hypothetic trusted party, also known as the *functionality*, and (2) show that the protocol "emulates" this behavior. Intuitively, this emulation should

ensure that nothing can go wrong in the *real world*, where the protocol is executed, which could not have gone wrong in a corresponding *ideal world* where the computation is taken care of by the assumed functionality. We shall come back to a more formal description of the security definition for MPC in the following chapter.

In the modern computerized world, the motivating scenario and the most typical setting of MPC is having each player sitting in front of his computer, where the computers are connected via some network, e.g., the Internet. The corresponding protocol defines, on a high level, the program which each computer is supposed to execute.

1.2 Misbehavior and Adversary

The security of multi-party protocols should be guaranteed even when some players are cheating, i.e., they are not behaving according to their protocol. Clearly, such misbehavior has the highest effect when the cheaters coordinate their actions. To model such a worst-case scenario, it is assumed that the actions of misbehaving players are coordinated centrally by an external entity, called the *adversary* . The adversary can corrupt players and use them to attack the security of the protocol. For a protocol to be secure, we require that any negative effect which the adversary's strategy has in the (real-world) protocol execution, can be translated to a corresponding effect in the respective ideal-world computation, i.e., when a trusted functionality is assumed which takes care of the computation. The correspondence is defined through the simulation-paradigm, which, in a nutshell, requires that for any real-world adversary $\mathcal{A}$ attacking the protocol, there exists an adversary $\mathcal{S}$, also referred to as the *simulator*, who is attacking the ideal-world computation, corrupts the same players as $\mathcal{A}$, and can achieve (in the ideal-world) whatever $\mathcal{A}$ can achieve in the real-world.

The power of the adversary can be described by specifying several parameters, such as the different ways in which the corrupted players might misbehave, but also how many (or even which) players can be simultaneously corrupted, and how the set of corrupted players is chosen. In the following we review some of these parameters which are relevant for the results in this thesis.

Corruption Types

The strongest and most destructive type of corruption is to allow the adversary full control over the corrupted player. This models situations in which the corresponding corrupted player might arbitrarily misbehave. However, there are many cases in which the misbehavior of players is limited and can be described by specifying a restricted set of (misbehaving) actions. For example, consider a player who has network-problems and might fail to send/receive some of his protocol messages, but he is following all his protocol instructions by letter. For this player it is an overkill to assume that he is arbitrarily misbehaving. In fact, protocols designed to be secure against this weaker type of misbehavior can tolerate strictly more corrupted players than protocols which are designed to tolerate arbitrary misbehavior. Such limited types of misbehavior are modeled by different corruption types, which specify exactly what the adversary can do with the corrupted player. In the following we describe four such corruption types.

- **Active corruption:** This is the strongest possible corruption-type and allows the adversary full control over the corrupted player. In particular, the adversary has access to everything the corrupted player knows or sees during the protocol, and can make him misbehave in an arbitrary way. This corruption-type can trivially model any type of misbehavior and is the hardest to protect against.

- **Fail-corruption:** This is a much weaker type of corruption, which allows the adversary to force the corrupted player to *crash* at some point of her (the adversary's) choice. When a player crashes, he stops sending or receiving messages in the protocol. Furthermore, the crashing is *irrecoverable*, i.e., a crashed player will never participate in any protocol in the future; one can think of crashing as making the player disappear from the world. Note, however, that forcing the corrupted players to crash is the only power which the adversary has over a fail-corrupted player. Fail-corruption is often referred to as *fail-stop*, or *fail-crash* corruption.

- **Omission-corruption:** When a player is omission-corrupted, the adversary can arbitrarily and selectively block messages sent to and from this player, but cannot make the player misbehave in any other way. The corruption power of the adversary lies in between active and fail-corruption, as the adversary could make an omission-corrupted player

behave as fail-corrupted by simulating the crash with permanently blocking all incoming and outgoing communication. In fact, as we shall see, tolerating omission-corruption is strictly harder than tolerating fail-corruption and strictly easier than tolerating active corruption (where hardness is measured in terms of how large the tolerable set of corrupted players can be). Omission-corruption can model situations, where the communication network of the player has temporary problems, e.g., due to buffer overflow in some router, or even situations where the player is temporarily unavailable, e.g. due to power breakdown.

- **Passive corruption:** A passively corrupted player is running all his protocol instructions correctly, but the adversary has access to everything this player knows and sees during the execution of the protocol, in particular, to his input(s) and the messages he receives. In contrast to all the above corruption types, a passively corrupted player always sends his correct messages, as instructed by his protocol, and cannot be distinguished from an uncorrupted player, not even by somebody who sees all computation and communication among the players. Therefore, passively corrupted players are often referred to as *semi-honest*. With passive corruption we can model situations where the players do follow the protocol, but form coalitions and try to obtain information on the values of other-players by looking at their joint view.

The goal of introducing corruption types other than active is to allow for a systematic study of feasibility of MPC under different ways of misbehavior. In fact, it is no surprise that by putting limits on the effect of corrupting a player, allows to achieve stronger security guarantees for the corrupted player. These guarantees are easier to understand in the idea-world evaluation (note that the ideal-world simulator has the same power on the corrupted player as the real-world adversary). For example, for an omission-corrupted player we cannot guarantee that his input-value will be considered in the evaluation of the function, as the simulator can block this player's communication with the functionality; but we *do* guarantee that this player is not going to give a wrong input (in the worst case he might give no input). Similarly, for a passively corrupted player, we cannot have any privacy guarantees for his input, as the simulator can see it, but we guarantee that he always gives his (correct) input to the functionality and receives from it his (correct) corresponding output. Note that the security of the protocol implies that such guarantees which the players enjoy in the ideal-world evaluation, they should also have in the real world, when executing the protocol.

MIXED ADVERSARY To study situations where players might misbehave in more than one ways, we consider a *mixed* adversary. Such an adversary can corrupt players in more than one ways, simultaneously. For example, a mixed active/passive adversary can corrupt some player actively and some player passively. Of course, the adversary could even choose to corrupt the same player in two different ways. Hence, with a mixed adversary we can also model additional types of misbehavior, which result by combining corruption types; e.g., with a mixed passive/fail adversary, some player might be both passively corrupted and fail-corrupted (at the same time), in which case the adversary sees whatever the player sees, but can also force him to crash.[1]

Description of the Corrupted Set

A second parameter specifying the adversary's strength has to do with the description of the sets of players that can be (simultaneously) corrupted. The most common description is by a threshold t which defines the maximum number of players that the adversary can corrupt, i.e., the adversary can corrupt all subsets of the player set with cardinality at most t. We refer to such a *threshold* adversary as a t-adversary. Threshold adversaries have been extensively studied in the MPC literature. However, they cannot model all possible corruption combinations, for example they are not suitable for addressing a setting where the adversary can either corrupt the first eight players (in any ordering) or, alternatively, corrupt the nine-th player alone.[2]

A generalization of the threshold adversary is the so-called *general adversary*: the adversary is described by a so called *adversary structure* $\mathcal{Z}$, which is an enumeration of the corruptible player sets. More precisely, $\mathcal{Z}$ is a collection of player sets, $\mathcal{Z} = \{Z_1, \ldots, Z_m\}$, where the adversary can choose any one of these sets, e.g., Z_k, and corrupt all the players in it. We refer to such an adversary as a $\mathcal{Z}$-adversary. We point out that the adversary structure is the most general way for specifying the corruptible sets. Furthermore, it is the most insightful way to discuss feasibility results for secure multi-party computation, as one is forced to exploit the structural properties of $\mathcal{Z}$ and cannot hide behind symmetries of the representation. The generality and importance of the model will be further explained in the following sections.

The notions of both the threshold and the general adversary can be extended to mixed adversaries. A mixed threshold adversary is characterized

[1] Often, especially in the distributed-systems literature, "fail-corruption" refers to what we consider as the result of passively corrupting and, in the same time, fail-corrupting a player.

[2] Of course, one could consider a threshold adversary with $t = 8$, but this is clearly an overkill.

by c thresholds, where c is the number of considered corruption types. For example, a threshold active/passive adversary is described by two thresholds ($c = 2$) t_a and t_p which are upper bounds on the number of actively and passively corrupted players, respectively. Such an adversary is denoted as (t_a, t_p)-adversary. Similarly, a mixed general adversary is described by an adversary structure $\mathcal{Z}$ which is a collection of c-tuples of player sets (again, c is the number of corruption types), where each tuple corresponds to a possible combination of corruptible sets for the different corruptions. We refer to these tuples as the *classes* of the adversary structure. For example, for an active/passive general adversary ($c = 2$), $\mathcal{Z}$ is a collection of classes (pairs), i.e., $\mathcal{Z} = \{(A_1, E_1), \ldots, (A_m, E_m)\}$, where the meaning of each class $(A, E) \in \mathcal{Z}$ is that the corresponding adversary actively corrupts the players in A and passively corrupts the players in E.

Choice of the Corrupted Set

Another dimension of describing the adversary's strength has to do with the question: "when is the set of corrupted players chosen?". We distinguish two types of adversaries called *static* and *adaptive*. A static adversary chooses all the players to corrupt at the beginning of the computation, before any message is exchanged. In contrast, an adaptive adversary corrupts players at any point *during* the computation, possibly depending on information which she has seen so far. More precisely, an adaptive adversary can corrupt additional players during the computation, having only the limitation which is imposed to her by the description of the corruptible sets. In particular, an adaptive (threshold) t-adversary can corrupt new players as far as the total number of corrupted players is at most t; an adaptive (general) $\mathcal{Z}$-adversary can corrupt new players as long as the set of all corrupted players is included in some of the classes in $\mathcal{Z}$. Note that even a static adversary is allowed to decide the future messages of corrupted players at any point *depending* on her current view of the computation until that point. What a static adversary is not allowed to do is modify her choice of corrupted players.

Computing power

Another parameter for characterizing the adversary's strength, which is a bit more esoteric to cryptography and complexity theory, is his computing power. In cryptography, we often make statements based on the assumed hardness of some computational problem. For example, RSA which is the

most known public key cryptosystem is based on the assumption that factoring integers is hard. To be able to base the security of protocols on the hardness of such problems, we assume that they are hard for the adversary to solve. In other words, we assume that the adversary does not have enough computing power/time to solve these problems. In that case we speak about a *computationally bounded* adversary and the corresponding protocol tolerating such an adversary is typically *computationally secure*. Note however, that many of the results in the field, and also in this thesis, do not make such an assumption, and hold even when the adversary is *unbounded* and can perform arbitrary long (but finite) computations. Protocols that are secure against such an unbounded adversary are called *information-theoretically secure*. The notions of "computational security" and "information-theoretic security" are defined more formally in the following chapter.

1.3 Reactive and Non-reactive Computation: (MPC and SFE)

Depending on whether or not the assumed ideal functionality corresponding to the computation *interacts* with the players *during the computation*, we distinguish two types of computation/functionalities, called *reactive* and *non-reactive*. The non-reactive computation, also referred to as *algorithmic* computation, corresponds to the traditional notion of algorithmically producing an answer to a computational problem, and proceeds in a closed-box fashion: the functionality receives inputs from the players, performs some computation on these inputs, and outputs the result(s) to the corresponding players. The most typical example of non-reactive computation is the evaluation of arithmetic functions. We shall refer to the computation of non-reactive functionalities as *Secure Function Evaluation* (SFE).

Unlike SFE, in reactive computation, also referred to as *interactive* computation, the functionality is interacting with the players during the evaluation, sending outputs and/or receiving new inputs. Reactive functionalities are to be thought of more as performing a task or providing a service to the players rather than computing the answer to an algorithmic question. As an example, consider a key-distribution service: the functionality maintains a list of keys which have been generated. At any point the players can make queries to the functionality to receive a key or verify the originality of some key of another player. It is straight-forward that such a reactive functionality *should be able to keep a private state of the computation*, which for the above example

would include the list of all keys. We shall use the term *multi-party computation* (MPC) to refer to this most general case of computing any (even reactive) functionality by a multi-party protocol.

1.4 Byzantine Agreement

One of the, perhaps most famous, instances of secure function evaluation is the so-called *Byzantine Agreement* (BA) problem. The problem comes in two flavors called *Consensus* and *Broadcast*. Both primitives require that the output of the players is consistent, i.e., that all players output the same value. The difference is that in Consensus, every player has an input and we require that if all players have the same input x then their output equals x. In Broadcast, on the other hand, there is a single player, called the *sender*, who has an input (every other player has no input) and the goal is to have all players output the sender's input. In both primitives, the consistency on the output should be ensured independent of the adversary's strategy.

The relevance of BA in the context of secure multi-party computation lies on the fact that both primitives are heavily used for the design of multi-party protocols. In fact, it is often the case that protocols are constructed in a model where Broadcast is assumed as a primitive which can be used by all players. The security of such protocols is argued independently of the actual implementation of the Broadcast primitive.

1.5 Typical Assumptions

In theoretical cryptology, it is essential that one can work at different levels of abstraction. To this direction, when studying MPC on a theoretical level, one makes several assumptions which vary from assumptions on the communication network to assumptions on resources that the players might be given access to. We already saw an example of an assumption, namely the infeasibility of certain computational problems, e.g., factoring large integers. In this section we review some additional assumptions which are relevant for our results.

Privacy and Authenticity of the Network

Typically, in MPC we assume that the players are connected with each other through a complete network of bilateral channels. The security properties of the underlying network play a central role in the security of the protocol. The two basic properties are the privacy of the transmitted message and the authenticity. According to these two properties we consider the following types of channels:

- **secure channels** A secure channel guarantees both privacy and authenticity of the transmitted message. Roughly speaking, when a player p_i sends a message x over a secure channel to some player p_j, then p_j receives x unchanged and knows that x was sent by p_i (authenticity). Furthermore, the adversary receives no information on the exchanged message, unless he actively or passively corrupts one of the players p_i and p_j (privacy). This is the strongest possible form of secure communication.

- **authenticated channels** An authenticated channel offers the same authenticity guarantee as a secure channel, but no privacy. In particular, as soon as a message is sent through an authenticated channel, the adversary learns it.

- **insecure channels** An insecure channel has neither authenticity nor privacy guarantees. In particular, the adversary is allowed to both read any sent message and modify it, and there is no way to verify that a message was sent from some specific p_i (unless the message includes some authentication data, e.g., a digital signature or a Message Authentication Code).

We point out that a big part of the cryptologic research is concerned with developing ways to augment the security guarantees of channels. For example, a suggested interpretation of an encryption scheme describes it as the means to transform an authenticated channel into a secure one [MS94, Mau09a, Mau09b]. In this thesis, as in most of the MPC literature, we abstract away from the question of how the assumed network properties can be implemented from weaker assumptions. In particular, if our protocol requires secure channels, then we assume that secure channels are given. This allows us to focus on the protocol aspects of the problem, rather than problems which occur due to "imperfectness" of the communication.

Synchronicity

Another assumption that one makes in the setting of MPC concerns the synchronicity of the communication network, which has mainly to do with the question: "how long does it take for a message which is sent until it is delivered". The most common assumption which is used to achieve *synchronous* communication, is that there is a known upper-bound on the delivery time of each message sent through the network. On the other extreme, an *asynchronous* network allows the adversary to decide when each message will be delivered. However, the adversary is not allowed to infinitely delay messages (which would essentially correspond to blocking them). In this thesis we shall consider almost exclusively synchronous communication.

Further Assumptions on the Communication

It is also typical in the multi-party protocols literature to assume a Broadcast primitive, also called a Broadcast channel. Such a Broadcast primitive is to be thought of as a megaphone which is given to the sender, and which every player listens to. Wherever, in this thesis, we assume Broadcast, we mean authenticated Broadcast, i.e., every player knows the identity of the sender.

Another assumption has to do with the atomicity of sending a message to several players. In particular, many works assume, implicitly or explicitly, some kind of *simultaneous multi-send*, which allows a sender to send several messages to several recipients simultaneously, i.e., in one shot as an atomic operation. Such a simultaneous multi-send primitive is, in fact, a quite powerful assumption, as it ensures that, unless the sender is corrupted, by the time the adversary learns any of the (multi-)sent message, the sender is already committed to all of them, and it is guaranteed that all players will get their respective message.

Trusted Setup

A further assumption one typically makes is that of a trusted setup. Roughly speaking, a trusted setup is some information to which all parties have access and which is guaranteed to satisfy certain (security) properties. An example of such a setup is a Public Key Infrastructure (PKI), which associates to each p_i, a secret-key/public-key pair $(\mathsf{sk}_i, \mathsf{pk}_i)$. Such a PKI can, for example, be used in a public-key cryptosystem or in a digital signatures scheme.

1.6 Relevant Literature

ORIGINS AND FIRST RESULTS The first glint of secure multi-party protocols can be found in the seminal paper of Yao [Yao82] who gave first solutions for the case of two-party computation. Since then, a big part of the MPC literature has been concerned with characterizing the maximal adversary tolerable for securely computing any functionality among n players. Protocols which can be used to compute *any* given specification/functionality are referred to as *general* MPC protocols.[3] The first general multi-party solution is due to Goldreich, Micali, and Wigderson [GMW87], who proved that, based on computational hardness assumptions, general secure MPC is possible if and only if $t < n/2$ players are actively corrupted, or, for passive corruption, if and only if $t < n$ players are passively corrupted. The first information-theoretic solutions were suggested by Ben-Or, Goldwasser, and Wigderson [BGW88], and independently by Chaum, Crépeau, and Damgård [CCD88], who proved that information-theoretically secure general MPC is possible if and only if $t < n/3$ players are actively corrupted, or, for passive corruption, if and only if $t < n/2$ players are passively corrupted. The results in [BGW88] achieve even *perfect* security, i.e., information-theoretic with zero error probability, whereas the solutions from [CCD88] involve some negligible error probability.

MIXED ADVERSARY Mixed adversaries were studied in the context of MPC by Fitzi, Hirt, and Maurer [FHM98]. They considered a (t_a, t_p, t_f)-adversary, i.e., an adversary who can actively corrupt t_a players, passively corrupt t_p players, and fail-corrupt t_f players, simultaneously, and showed that such an adversary can be tolerated for perfectly secure MPC if and only if $3t_a + 2t_p + t_f < n$. Furthermore, they showed that when a Broadcast primitive is assumed, then statistical (i.e., information-theoretic with negligible error probability) security is possible if and only if $2t_a + 2t_p + t_f < n$.

GENERAL ADVERSARY The study of general adversaries in MPC was initiated by Hirt and Maurer [HM97] (see also [HM00]), where exact conditions on feasibility of MPC were proved for either passive or active corruption. In particular, they showed that perfectly secure MPC tolerating a general adversary is possible if and only if no three (resp. no two) actively (resp. passively)

[3] Note that here the term "general" is used different than in the context of general adversaries. In particular, a general MPC protocol is not (necessarily) one which is secure against a general adversary.

corruptible sets cover the whole player set. A mixed (active/passive) general adversary was consider in [FHM99], where corresponding exact feasibility bounds for perfect security, and also for statistical security given Broadcast were proved. However, fail-corruption was not considered, and these results do not handle the computational-security case, or the statistical-security case without Broadcast. Based on work by Beaver and Wool [BW98] on Quorum-based MPC and also work on MPC with efficiently representable adversary structure [CDM00], Maurer [Mau02] (see also [Mau06]) simplified the MPC protocols for a mixed (active/passive) general adversary, and suggested an approach for designing a general MPC protocol, which resembles the threshold approach.

BYZANTINE AGREEMENT A similar development as with MPC can be observed in the setting of Byzantine Agreement (BA). The problem originates in the seminal papers by Lamport, Shostak, and Pease [PL80, LSP82], where it was introduced and first solutions were suggested. In particular, they showed that when no setup is assumed, Consensus and Broadcast are possible if and only if less than a third of the players are malicious (i.e., $t < n/3$). Later solutions [DS82, BPW91, BHR07] considered a setting where a setup allowing digital signatures is available, and showed that Broadcast tolerating an arbitrary number of active cheaters ($t < n$) is possible, whereas Consensus is possible if and only if $t < n/2$.[4] Lamport and Fisher [LF82] considered an adversary who can fail-corrupt up to t players, and showed that any $n - 1$ players being fail-corrupted can be tolerated for Broadcast. The above results were unified in [GP92], where it was shown that if at most t_a players can be actively corrupted and, simultaneously, at most t_f can be fail-corrupted, and no setup is assumed, then $3t_a + t_f < n$ is an exact bound for feasibility of BA. In [HMZ08], it was observed that when a setup is assumed, then the existing protocols for Broadcast and Consensus do not work for an adversary who can actively and, simultaneously, passively corrupt players. The reason is that in such a model the signatures of passively corrupted players are not reliable, as the adversary knows the signing keys and can trivially fake them. In [GGBS08] it was shown that given a PKI, an adversary who can actively corrupt up to t_a players and passively corrupt up to $t_p > 0$ players can be tolerated for Consensus if and only if $2t_a + \min\{t_a, t_p\} < n$. Finally, in [LLR02] a Broadcast protocol for $t < n$ was described, which is concurrently composable when unique session IDs are available; this last result implicitly as-

[4] In fact, feasibility of Broadcast for $t < n$ when a setup is available was also proved in [LSP82], but the suggested protocol has exponential communication complexity.

sumes that the players can simultaneously multi-send messages (c.f. [LLR02, Sect. 2.1]).

The first solutions to feasibility of BA tolerating a general adversary follow from the results in [HM97] for the case of active corruption without a setup, where it was shown that $\mathcal{Z}$-secure BA (for any security level) is possible if and only if no three corruptible sets cover the whole player set. In the same setting, i.e., no setup, [AFM99] considered a mixed general adversary who can actively and fail-corrupt players (no passive corruption) and proved corresponding exact feasibility bounds; the protocols from [AFM99] are efficient (polynomial) in terms of communication and bit complexity.

OMISSION-CORRUPTION The first to consider omission-corruption were Perry and Tueg [PT86]. They considered a threshold adversary who can omission-corrupt up to t players and showed that BA tolerating this adversary is possible if and only if $t < n$. However, their consistency-guarantee is limited to the outputs of uncorrupted players, i.e., omission-corrupted players are allowed to output arbitrary values. Raynal and Parvedy [Ray02, PR03] proved that if we require omission-corrupted players to output either the correct value (i.e., consistent with the output of uncorrupted players) or no value, then Consensus is possible if and only if $2t < n$. In the context of general MPC, omission-corruption was first studied, in combination with active corruption, by Koo [Koo06]. He considered a threshold adversary who can actively corrupt up to t_a players and, simultaneously, omission-corrupt up to t_ω players,[5] and proved that the conditions $3t_a + 2t_\omega < n$ and $3t_a + 4t_\omega < n$ are sufficient for perfectly secure Consensus and general MPC, respectively; however, necessity was not proved for any of the conditions.

1.7 Contributions of this Thesis

The main goal of this thesis is to investigate feasibility of multi-party computation under adversaries with different power. The power of the adversary is specified through the parameters which were described in Section 1.2: corruption-types, mixed or single corruption, threshold or general, adaptive or static. More precisely, the thesis is split in three parts/subject-units, indicated by the Latin numbers I, II, and III.

[5] Koo refers to omission-corrupted players as *constrained* and to actively corrupted as *corrupted*.

Part I considers a static mixed general adversary who can actively corrupt, passively corrupt, and fail-corrupt players, simultaneously. For this adversary we prove feasibility bounds for all three security levels, i.e., perfect security [BFH+08], and statistical and computational security [HMZ08] assuming Broadcast. The problem of securely realizing Broadcast and Consensus, for all three security levels, tolerating such an adversary is then studied in Chapter 4 [HZ10b]. The results in this part, extend the known bounds for MPC [FHM99, Mau02] and BA [AFM99] by adding fail-corruption and passive corruption, respectively. We point out that the corresponding exact conditions are much more involved than in past results. Furthermore, the results bring up a separation of MPC and SFE in this model, i.e., we show that there are adversary structures which can be tolerated for SFE but not for MPC. Interestingly, such a separation does not show up when threshold corruption or when no fail-corruption is considered. To our knowledge, the separation was first observed by Altmann in his master thesis at ETH Zurich [Alt99]. The first published result implying such a separation is due to Ishai et al [IKLP06], where it was shown that an adversary who can *either* corrupt $t_a < n/3$ players actively, or, alternatively, $t_p < n/2$ players passively can be tolerated for general SFE but not for MPC.[6]

In Part II we look at the extension of the adversary's power to another dimension, namely an alternative corruption-type. More precisely, we consider a static mixed threshold adversary who can actively corrupt t_a players, and, simultaneously, fail-corrupt t_f and omission-corrupt t_ω players. For this adversary we show that the bound $3t_a + 2t_\omega + t_f < n$ is necessary and sufficient for MPC with perfect security, and the bound $2t_a + 2t_\omega + t_f < n$ for computational and statistical security assuming Broadcast. The sufficiency result for the case of perfect security is based on the results from [ZHM09], whereas the corresponding proof of necessity and the handling of statistical and computational security are extensions of [ZHM09]. The above bounds for MPC extend the landscape of previously known results in several ways. First, if one sets $t_f = 0$, i.e., no fail-corruption, then our result significantly improves the corresponding bound from [Koo06] (and fixes some issues with the functionality) and extends it to the cases of statistical and computational security; furthermore these bounds are proved exact, i.e., no further optimization with respect to corruption resilience is possible. Moreover, our results consider fail-corruption in addition to active corruption and omission-corruption.

[6] We point out that the adversary in [IKLP06] is a restriction of the mixed threshold adversary defined in Section 1.2, as he is not allowed to corrupt players passively and actively, simultaneously.

Finally, in Part III we consider an adaptive adversary. We observe an (in)composability issue that occurs in most, if not all, approaches to secure Broadcast in the literature, and suggest solutions when a threshold active-only adaptive adversary is considered [HZ10a]. Interestingly, the exact bounds for computational and statistical security, assuming a setup for generation/verification of digital signatures, are different than the bounds which exist in the literature. This implies that the corresponding known protocol are secure only against a static adversary.

Chapter 2

Setting the Landscape

In this chapter we briefly review some standard definitions from the literature and introduce some notation. Furthermore, we describe the model in which we work in the following chapters. The goal of the current chapter is not to formally review all relevant definitions, but rather to describe them on a level which is sufficient for understanding the claims and the proofs in this thesis. Nevertheless, for readers who are interested in a more formal handling of the mentioned concepts, we give thorough literature-pointers. Readers who are familiar with the field of multi-party computation can skip this chapter. Wherever, in the following chapters, some non-standard notation is used, we give a pointer to the corresponding section where this notation is introduced.

2.1 Security of Multi-Party Protocols

In a big part of the MPC literature, especially in the first works [Yao82, GMW87, CCD88, BGW88], the security of multi-party protocols is specified by two properties, namely *correctness* and *privacy*. Roughly speaking, correctness requires that the output of the computation is correctly computed on the players' inputs, even when some (corrupted) parties misbehave; privacy requires that the adversary gets no more information on the inputs of uncorrupted parties than what she can compute from the outputs of corrupted parties. Despite the apparent completeness of this definition, it was observed that in some cases it fails to capture all the aspects of what one expects from secure computation.

A more accurate way of defining security of protocols is the simulation-based approach [Can00, PSW00, DM00, Can01, PW01, BPW03]. Here, the security of protocols is argued via the ideal-world/real-world paradigm. In the real-world the players execute the protocol. The ideal-world corresponds to a specification of the task which we want the protocol to implement. More concretely, in the ideal-world the players can involve a fully trusted party, called the *functionality* and denoted as $\mathcal{F}$, in the following way: the players send their input(s) to $\mathcal{F}$; $\mathcal{F}$ is given a specification of the computation as a program; it runs this program on the received inputs (while running the program, $\mathcal{F}$ might interact with the players and the adversary), and returns to the players their respective outputs. The specification of $\mathcal{F}$ is such that this ideal evaluation captures, as good as possible, the goals of the designed protocol. The security of a protocol is defined in terms of securely realizing the corresponding ideal functionality $\mathcal{F}$.

Intuitively, a protocol *securely realizes* (emulates) the functionality $\mathcal{F}$, when the adversary cannot achieve more in the protocol than what she could achieve in an ideal-evaluation of $\mathcal{F}$. This is modeled by requiring that for any real-world adversary $\mathcal{A}$ attacking the protocol, there exists an ideal-world adversary $\mathcal{S}$, also known as the *simulator*, whose goal is to "fake" an interaction among the players and the adversary which "resembles" the interaction of a real-world execution of the protocol. Furthermore, $\mathcal{S}$ should be able to do so, independently of the context/environment in which the protocol is invoked. To formalize this, we assume an environment Ψ which decides the inputs of all players, and also sees their outputs.[1] Also, Ψ might communicate with the adversary in an arbitrary way. We denote the view of Ψ for an invocation of protocol π with adversary $\mathcal{A}$ as $\text{EXEC}_{\pi,\mathcal{A},\Psi}$. As usually, all the participating entities, i.e., the players, the adversary, and the environment, are modeled as probabilistic Turing machines, and their view consists of the contents of all their tapes. A protocol π *$\mathcal{A}$-securely realizes functionality $\mathcal{F}$* when for the adversary $\mathcal{A}$ attacking protocol π, there exists an ideal-world adversary $\mathcal{S}$ (the *simulator*) such that no environment Ψ can tell whether it is interacting with $\mathcal{A}$ and the players running π or with $\mathcal{S}$ and the players running the ideal-world protocol (we denote the view of Ψ in an ideal-evaluation of $\mathcal{F}$ as $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi}$). In the case of a threshold adversary, we say that a protocol t-securely realizes $\mathcal{F}$, if for any t-adversary $\mathcal{A}$, protocol π $\mathcal{A}$-securely realizes functionality $\mathcal{F}$. Similarly, for a general adversary, we say that a protocol $\mathcal{Z}$-securely realizes $\mathcal{F}$ if for any $\mathcal{Z}$-adversary $\mathcal{A}$, protocol π $\mathcal{A}$-securely realizes functionality $\mathcal{F}$.

[1] In most works the symbol $\mathcal{Z}$ is used to denote the environment. As we use $\mathcal{Z}$ for denoting adversary structures we shall use the letter Ψ for denoting the environment.

The quantities $\text{EXEC}_{\pi,\mathcal{A},\Psi}$ and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi}$ are probability ensembles, i.e., families of distributions of random variables, parameterized by some security parameter κ.[2] Depending on the computing power of $\mathcal{A}$ and Ψ three security levels are specified as follows: For *perfect security* a computationally unbounded adversary $\mathcal{A}$ is assumed, and $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi}$ and $\text{EXEC}_{\pi,\mathcal{A},\Psi}$ are required to be identical, we denote this as $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi} \equiv \text{EXEC}_{\pi,\mathcal{A},\Psi}$ (i.e., an unbounded environment has zero distinguishing advantage). For *statistical security*, $\mathcal{A}$ is assumed to be computationally unbounded, but the requirement is that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi}$ and $\text{EXEC}_{\pi,\mathcal{A},\Psi}$ are statistically close (i.e., an unbounded environment Ψ has negligible distinguishing advantage). Finally, for *computational security*, $\mathcal{A}$ is assumed to be computationally bounded (efficient), and we require that $\text{IDEAL}_{\mathcal{F},\mathcal{S},\Psi}$ and $\text{EXEC}_{\pi,\mathcal{A},\Psi}$ are computationally indistinguishable (i.e., an efficient, e.g., polynomial, environment has negligible distinguishing advantage). For formal definitions of the notion of probability ensembles, the indistinguishability notions, and the security levels mentioned above the reader is referred to [Gol04, Can00, Can01].

COMPOSABILITY AND THE HYBRID MODEL The most important feature of the simulation-based approach is that it allows to argue about security of protocols in a composable way. In particular, let π_1 be a protocol which securely realizes a functionality $\mathcal{F}_1$. If we can prove that a protocol π_2 securely realizes a functionality $\mathcal{F}_2$ using *ideal-calls* to $\mathcal{F}_1$,[3] then it follows automatically that the protocol which results by replacing, in π_2, the calls to $\mathcal{F}_1$ by invocations of π_1 also securely realizes $\mathcal{F}_2$. Therefore we only need to prove the security of π_2 in the so-called $\mathcal{F}_1$*-hybrid* model, where the players run π_2 and are allowed to make ideal-calls to $\mathcal{F}_1$. For more details on composability of protocols and a formal handling of composition, the reader is referred to [DM00, Can00, Can01].

A disadvantage of the simulation-based approach is that, sometimes, proofs become cumbersome and non-intuitive. Most proofs in this thesis, especially the ones considering a static adversary, are, in fact, proof-sketches which give the guidelines on how the security of the corresponding protocol can be argued.

[2] The corresponding probabilities are defined over the random choices of the players, the adversary, and the environment.

[3] An ideal call to $\mathcal{F}_1$ is an invocation of $\mathcal{F}_1$ as in the ideal-world.

2.2 Corruptions and Guarantees

The corruption types which we consider in this thesis are the ones introduced in the previous chapter. We briefly repeat the effect each corruption type has on a corrupted player p:

- active corruption: the adversary has full control over p.
- passive corruption: the adversary sees the internal state of p.
- fail-corruption: the adversary can force p to crash at any point.
- omission-corruption: the adversary can selectively block messages sent or received by p.

Following the intuition of the simulation-based definition, i.e., that the adversary cannot do more harm in the real-world than what she can do in the ideal-world, it is natural to provide to the corrupted players all the guarantees which they would enjoy in the ideal-world. This is the approach we take in this thesis. In particular, the assumed simulator has the same power over the corrupted players (in the ideal-world) as the adversary has in the real-world. This implies that for a passively corrupted player, it is guaranteed that he gives his correct/intended input(s) to the functionality, and (correctly) receives from it his corresponding outputs. For a fail-corrupted player, we make sure that he never gives a wrong input or receives a wrong output, but he might, if he crashes before the end of the computation, give no input (or receive no output).[4] Moreover, the privacy of a fail-corrupted player's input(s) is always guaranteed (unless he is, at the same time, also actively or passively corrupted). Similarly to fail-corruption, an omission-corrupted player sends the functionality either his correct input(s) (resp. receives his correct output(s)) or no input (resp. no output) at all; and, also gets the same privacy guarantee as a fail-corrupted player. Finally, for actively corrupted players we give no correctness/privacy guarantees whatsoever.

In some works, an alternative approach is taken, which provides no or limited guarantees to corrupted players, independently of the corruption type. The main arguments for justifying such an approach are, first, that a corrupted player is, anyway, dishonest and does not deserve any guarantees, and, second, that this way one can tolerate more corrupted players. However, we argue that this approach is not the most natural. On the one hand,

[4]Note that the functionality has to be instructed how to cope with such a case where some player does not send his input; a possible way would be to take a default value for this input.

it contradicts the motivation of many of the corruption types, e.g., omission-corruption is motivated by situations, where the corrupted player is honestly executing all his protocol instructions but has network problems. On the other hand, for such an approach to (formally) work, one needs to consider an ideal-world adversary/simulator, who has strictly more power over the corrupted parties than the corresponding real-world adversary. This "strengthening the simulator" is inconsistent with the intuition of the simulation-based definition. Additionally, when taking such a "strengthening the simulator's power"-approach, one needs to be careful, as unexpected things might happen.

2.3 General Adversary

We describe in details the general adversary model and introduce some necessary notation. A general adversary is described by an adversary structure $\mathcal{Z}$, which is an enumeration of all possible corruption-combinations which the adversary is allowed to do. For the case of a plain (non-mixed) adversary, i.e., an adversary who can corrupt players in only one way/type, the adversary structure $\mathcal{Z}$ is a set of subsets of some given player set $\mathcal{P}$, i.e., $\mathcal{Z} = \{Z_1, \ldots, Z_m\}$, where $Z_k \subseteq \mathcal{P}, k = 1, \ldots, m$. The interpretation is that a $\mathcal{Z}$-adversary might corrupt all the players in some set $Z_k \subseteq \mathcal{P}$ if and only if $Z_k \in \mathcal{Z}$.

The extension of the notion of an adversary structure to capture mixed adversaries is straight-forward: Instead of subsets of $\mathcal{P}$, the elements Z_k of $\mathcal{Z}$ are c-vectors of subsets of $\mathcal{P}$ (c is the number of corruptions), called *classes*. The ℓ-th component ($1 \leq \ell \leq c$) of every class $Z_k \in \mathcal{Z}$ corresponds to the same corruption type, and is the set of players which an adversary of this class corrupts according to this type. To clarify this extension and allow the reader more familiarity with the kind of adversary structures considered in this thesis, we describe in more detail the adversary structure for an active/passive/fail adversary. Such an adversary structure $\mathcal{Z}$ is a collection of triples of player sets (classes), i.e., $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$. The interpretation of a class $(A, E, F) \in \mathcal{Z}$ is that an adversary of this class actively corrupts the players in A, passively corrupts the players in E,[5] and fail-corrupts the players in F. For a static $\mathcal{Z}$-adversary $\mathcal{A}$ we shall denote the class which characterizes the actual corruption choice of $\mathcal{A}$ as $(A^\star, E^\star, F^\star)$

[5]We use E for passive corruption corresponding to "Eavesdropping".

and refer to it as the *actual adversary* class. Note that $(A^\star, E^\star, F^\star)$ is not known to the players and appears only in the security analysis.

For sake of simplicity, in the remaining of this section (and also in Section 2.3.1) we only consider an active/passive/fail general adversary, as this is the type which is mostly addressed in this thesis.[6] However, the approach can be easily extended to deal with more or fewer corruption-types.

We describe a way of visually representing an adversary structure $\mathcal{Z}$, which is convenient for most of our (small scale) examples. The structure $\mathcal{Z}$ is represented as a matrix, where the columns are indexed by the players ($|\mathcal{P}|$ columns) and the rows are indexed by the classes in $\mathcal{Z}$ ($|\mathcal{Z}|$ rows). The ith row corresponds to the ith class of $\mathcal{Z}$, where for each player $p_j \in \mathcal{P}$ the entry in the jth column describes whether and in which way an adversary of this class corrupts player p_j. More concretely, in position (i, j) (i.e., ith row and jth column) there is an entry a if the adversary of class i can actively corrupt p_j; similarly, there is an entry e or f if the adversary of class i can passively corrupt or fail-corrupt p_j, respectively. If the adversary can corrupt p_j in more than one ways, then the corresponding position contains all the corresponding symbols. For example, the structure $\mathcal{Z} = (\{p_1\}, \emptyset, \{p_2\}), (\emptyset, \{p_1, p_3\}, \{p_1\}), (\{p_3\}, \{p_2\}, \emptyset)$ is visually represented as follows:

	p_1	p_2	p_3
Z_1	a	f	
Z_2	e/f		e
Z_3		e	a

2.3.1 Conventions and Notation

We introduce the following binary relation on adversary classes: $(A_i, E_i, F_i) \sqsubseteq (A_j, E_j, F_j)$ if and only if $A_i \subseteq A_j \wedge E_i \subseteq E_j \wedge F_i \subseteq F_j$. We denote by $\biguplus \mathcal{Z}$ the monotone closure of $\mathcal{Z}$ with respect to $\sqsubseteq$, which, for each class $(A, E, F) \in \mathcal{Z}$, contains all its sub-classes, i.e., $\biguplus \mathcal{Z} = \{(A', E', F') : \big((A', E', F') \sqsubseteq (A, E, F)\big) \wedge \big((A, E, F) \in \mathcal{Z}\big)\}$. We also introduce the notion of the *basis* structure $\overline{\mathcal{Z}}$, which includes only the maximal (with respect to $\sqsubseteq$) classes in $\mathcal{Z}$, i.e., $\overline{\mathcal{Z}} = \{(A, E, F) : \nexists (A', E', F') \in \mathcal{Z}$ with $(A', E', F') \neq (A, E, F) \wedge (A, E, F) \sqsubseteq (A', E', F')\}$. We point out

[6] At times, we consider general adversaries where some of the three corruption types are missing, but no general adversaries with additional corruption types.

that all the protocols which are proved to be $\mathcal{Z}$-secure in this thesis are trivially also $(\biguplus \mathcal{Z})$-secure. This is an intuitive monotonicity property, as it models the fact that if any adversary of class (A, E, F) is tolerated, then any (weaker) adversary of any sub-class $(A', E', F') \sqsubseteq (A, E, F)$ is also tolerated. Additionally, this property implies that for the design of our protocols we can consider the basis structure $\overline{\mathcal{Z}}$ instead of $\mathcal{Z}$. Observe that $\overline{\mathcal{Z}}$ is typically smaller than $\mathcal{Z}$; therefore, because the complexity of the designed protocols depends on the number of classes in $\mathcal{Z}$, taking $\overline{\mathcal{Z}}$ instead of $\mathcal{Z}$ typically leads to more efficient protocols.

Remark 2.1 (Monotonicity). In some of the general-adversary literature, the adversary structure is, *by definition*, required to be monotone, i.e., it is required that $\biguplus \mathcal{Z} = \mathcal{Z}$. In this thesis we take the more general approach and allow even non-monotone structures. However, as already mentioned, all our $\mathcal{Z}$-secure protocols are also secure with respect to the monotone closure $\biguplus \mathcal{Z}$.

To simplify the description we make the following convention: For a mixed active/passive/fail general adversary $\mathcal{Z}$, for any class $(A, E, F) \in \mathcal{Z} : E \supseteq A$ and $F \supseteq A$.[7] This is without loss of generality, as an actively corrupted player could behave as being passively corrupted or fail-corrupted. In fact, all our protocols have the property that if a corruption-combination is tolerated, where some player p is actively corrupted, then the same combination, where p is passively corrupted or fail-corrupted (instead of actively corrupted) is also tolerated.

Finally, we introduce the following operators on adversary structures: For a player set B, we denote by $\mathcal{Z}|^{B \subseteq F}$ the sub-structure of $\mathcal{Z}$ that contains only classes which allow fail-corrupting all the players in B, i.e., $\mathcal{Z}|^{B \subseteq F} = \{(A, E, F) \in \mathcal{Z} : B \subseteq F\}$.[8] Furthermore, for a player set $\mathcal{P}'$, we denote by $\mathcal{Z}|_{\mathcal{P}'}$ the adversary structure with all classes in $\mathcal{Z}$ restricted to the player set $\mathcal{P}'$, i.e., $\mathcal{Z}|_{\mathcal{P}'} = \{(A \cap \mathcal{P}', E \cap \mathcal{P}', F \cap \mathcal{P}') : (A, E, F) \in \mathcal{Z}\}$. As syntactic sugar, we write $\mathcal{Z}|^{B \subseteq F}_{\mathcal{P}'}$ for $(\mathcal{Z}|^{B \subseteq F})|_{\mathcal{P}'}$. Because all our $\mathcal{Z}$-secure protocols are also $(\biguplus \mathcal{Z})$-secure and vise versa, in the construction of the protocols we can use $\overline{\mathcal{Z}}$ instead of $\mathcal{Z}$ and apply the above operators to it without loss of security or generality.

[7] When, at times, we only consider active/fail (resp. active/passive) adversaries, we only assume $F \supseteq A$ (resp. $E \supseteq A$) to hold.

[8] Note that we only introduce this selection operator $\mathcal{Z}|^{\cdot}$ for the condition $B \subseteq F$, as this is the only way we use it in this thesis. However, one could extend it to an arbitrary condition.

2.4 Characterizing the Players

We use the following characterizations for players: A player is said to be *alive* at a certain point of the protocol if he has not crashed up to that point. Note that only fail-corrupted players might stop being alive (when they crash).[9] Moreover, we say that a player is *correct* at a certain point of the protocol if he has followed the protocol instructions correctly up to that point. A player who has deviated from the protocol (e.g., has crashed or has sent inconsistent messages) is called *incorrect*. Note that a fail-corrupted player is both correct and alive until the point when he crashes. Furthermore, an omission-corrupted player becomes incorrect as soon as the first of his sent/received messages is blocked. With respect to corruption, a player is said to be *uncorrupted* or *honest* at a certain point in the protocol, if he has not been corrupted (in any corruption type) up to that point. Observe that for a static adversary, the set of uncorrupted players is decided before the beginning of the protocol and remains unchanged.

2.5 The Computation

As already mentioned, the goal of MPC is to securely perform some given computation on inputs provided by the players in a distributed manner. The computation is represented as an arithmetic circuit C over some finite field $\mathbb{F}$, consisting of input, addition, multiplication, random, and output gates. All the gates have a single output (fan-out equals one), where, the multiplication and addition gates take two inputs (fan-in equals two), the input and output gates take one input (fan-in equals one), and the random gates have no input. Note that any efficient computation[10] can be represented as such an arithmetic circuit with number of gates polynomial in the number of steps of a corresponding Turing machine performing this computation.

The "Standard" Paradigm

Most MPC protocol in the literature follow a "standard" paradigm which is based on secret sharing. A *secret sharing scheme* allows a player, called the

[9] Although, some actively corrupted player might behave as having crashed, he is still considered alive, as he might, if the adversary wishes, start playing again at a future point.

[10] Here, by efficient we mean a computation that can be carried out by a Turing machine within a polynomial number of steps.

dealer, to distribute a value among the players in some set $\mathcal{P}$, in such a way that only qualified subsets of $\mathcal{P}$ can reconstruct it, and non-qualified sets learn nothing about the secret. If we make sure that any set of passively or actively corruptible players is non-qualified, then we know that the adversary learns nothing about the secret. In that case, the following paradigm can be used to construct an MPC protocol: The evaluation of the circuit C proceeds in a gate-by-gate fashion, where the invariant is that all intermediate values are secret-shared. In particular, for evaluating an input gate, the input-player secret-shares his input; for evaluating an addition or multiplication gate, a sharing of the corresponding sum or product, respectively, is computed; for evaluating a random gate, a sharing of a uniformly random value is computed; and for evaluating an output gate, the sharing of the output value is reconstructed towards the corresponding player.

2.6 The Network

Most results in this thesis are stated in the synchronous secure channels model [BGW88, CCD88]. More precisely, the players are connected through a complete network of bilateral secure channels. The communication is synchronous, which means that there is a known upper-bound on the delivery-time of any sent message, and the players have synchronized clocks. Protocols for such a synchronous network proceed in rounds, where in each round every player can send a message to every other player. Messages that are sent within some round are delivered by the beginning of the next round.

We do not assume simultaneous multi-send, i.e., a player who is instructed to send messages to more than one players can do so one message at a time. Note that this is consistent with the formulation of [Can01], where it is required that the processes are activated in turns, and at each point only a single process might send/receive messages to/from exactly one other process.

For simplicity in the description we adopt the following convention: Whenever a player does not receive a message (when expecting one), or receives a message outside of the expected range, the special symbol $\perp \notin \mathbb{F}$ is taken for this message.

2.7 Computational Assumptions

Computational, aka cryptographic, security is based on unproved computational assumptions. These are assumptions on the hardness of certain computational problems, i.e., we assume that these problems are hard for the considered adversary to solve. For example, RSA is believed to be secure assuming that factoring large integers is hard. A famous computational assumption is that of the existence of one-way trapdoor permutations. Informally, these are collections of permutations which are "easy" to compute but "hard" to invert, unless the inverting algorithm is given some additional information, i.e., some trapdoor τ. Given the trapdoor, it is "easy" to invert the permutation. A standard assumption which is used in the construction of computationally secure MPC protocols is this of *enhanced one-way trapdoor permutations*. These are one-way trapdoor permutations with an "enhanced" hardness property, with respect to the standard definitions of one-way trapdoor permutations. Reviewing the details of this assumption is not necessary for understanding the results in this thesis, but for the interested reader we point to [Gol04] for formal definitions.

2.8 Digital Signatures and Commitments

Many of our protocols make use of digital signatures. In particular they assume a trusted setup which allows for generation and verification of digital signatures. Such a setup can be a Public-Key Infrastructure (PKI): every player p_i holds some signing-key sk_i known exclusively to p_i, such that every p_j holds a corresponding (public) verification-key pk_i. Using sk_i player p_i can generate a signature on any message m of his choice, such that when this signature is given to any p_j, then p_j can verify (using pk_i) that the signature "matches" p_i's signing key. We denote a signature on message m that has been generated using p_i's signing key as $\mathrm{sig}_i(m)$. We use the standard notion of security of computational signatures, namely *existential unforgeability against chosen message attacks* (EU-CMA). In a nutshell, this requires that the probability of an efficient adversary forging a signature of some player of whom she does not have the signing key on any message is negligible, even if she is allowed to see arbitrary many signatures (on any other message).

To simplify the description of protocols using digital signatures, the PKI assumption is often replaced by assuming a signatures functionality $\mathcal{F}_{\mathrm{SIG}}$, which takes care of the generation and verification of digital signatures.

In such an approach, the corresponding protocols are invoked in the $\mathcal{F}_{\text{SIG}}$-hybrid world: whenever a player wishes to generate/verify some signature, the functionality $\mathcal{F}_{\text{SIG}}$ is invoked. Such a functionality was suggested by Canetti [Can03], where it is proved that it satisfies the EU-CMA security notion mentioned above, and it was also shown how this functionality can be securely realized using a scheme satisfying this notion. The advantage of stating our results in such an $\mathcal{F}_{\text{SIG}}$-hybrid world is that we can make statements about statistical security, assuming that a signature scheme is given which allows to statistically securely realize the functionality $\mathcal{F}_{\text{SIG}}$. However, we should point out that it remains an open problem to come up with such schemes.

In Chapter 3, in the computational-security section (Section 3.6), we make use of a commitment scheme. Informally, a commitment scheme consists of two protocols Commit and Open, and involves two players p_c, called the commiter, and p_r, called the receiver. In Commit the commiter, who has as input a value s, computes (using s and some randomness r) and sends some information to the receiver. We refer to this information as *p_c's commitment on s*. Later on, in Open the commiter may release more information to the receiver to open his commitment, so that the receiver learns s. This is typically done by the commiter sending to the receiver the randomness r which he used in protocol Commit, and the receiver checking that this matches the commitment he received.

Loosely speaking, the two basic properties we require from a commitment scheme are, first, that it is *hiding*: a cheating receiver cannot learn s from protocol Commit, and, second, that it is *binding*: a cheating commiter cannot change his mind about s after protocol Commit is complete, i.e., in Open p_r does not accept any $s' \neq s$. Each of the two properties can be satisfied unconditionally or computationally, i.e., relatively to some complexity assumption. Perfectly hiding (and computationally binding) commitments, which is the kind of commitments used in Chapter 3, are known to exist if enhanced one-way trapdoor permutations exist. We refer to [Gol03, Gol04] for a good textbook description and constructions of such commitments.

Part I

MIXED GENERAL ADVERSARY

Chapter 3

Multi-Party Computation

In this chapter we concentrate on a general adversary who can actively corrupt, passively corrupt, and fail-corrupt players simultaneously. The adversary structure consists of classes which are triples of player sets, i.e. $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$, where the adversary of class (A_i, E_i, F_i) can actively corrupt the players in A_i, passively corrupt the players in E_i, and fail-corrupt the players in F_i. Recall that, without loss of generality, we assume that for all $(A, E, F) \in \mathcal{Z} \ : \ E \subseteq A$ and $F \subseteq A$; furthermore, the protocols presented in this chapter remain secure even if we replace $\mathcal{Z}$ with its monotone closure $\biguplus\mathcal{Z}$ (we refer to Section 2.3.1 for explanation of the used notation). The goal of this chapter is to prove exact bounds for general secure multi-party computation (MPC) and secure function evaluation (SFE) tolerating such an adversary for all three security levels, i.e., perfect, information-theoretic, and computational security.[1] As we show, in all three models the two primitives separate, namely there are adversaries which can be tolerated for SFE but not for MPC. The results in this chapter are based on [BFH$^+$08, HMZ08].

Our results are stated in the secure-channels model, where it is assumed that the players have access to a reliable Broadcast channel. The properties that such a Broadcast channel is required to satisfy are described in Chapter 4, where it is also shown how to implement Broadcast from secure (or even authenticated) channels. Throughout this chapter we consider a non-adaptive adversary, i.e., the adversary chooses the class to corrupt, denoted as $(A^\star, E^\star, F^\star) \in \mathcal{Z}$, at the beginning of the protocol.

[1] Recall that we use SFE to refer to non-reactive computation.

SECRET SHARING SCHEME We use the following secret-sharing scheme which is based on the corresponding sharing scheme from [BW98, Mau02]: the secret s is split into summands that add up to s, where each summand might be given to several players. The sharing is characterized by a vector $\mathcal{S} = (S_1, \ldots, S_m)$ of subsets of $\mathcal{P}$, called the *sharing specification*. More precisely, a value s is shared according to $\mathcal{S}$ if there exist summands $s_1, \ldots, s_m \in \mathbb{F}$ such that $\sum_{k=1}^{m} s_k = s$, and for each $k = 1, \ldots, m$ every $p_j \in S_k$ holds s_k. For each $p_j \in \mathcal{P}$ the vector $\langle s \rangle_j = (s_{j_1}, \ldots, s_{j_\ell})$ is considered to be p_j's *share* of s, where $s_{j_1}, \ldots, s_{j_\ell}$ are the summands held by p_j. The vector of all shares, denoted as $\langle s \rangle = \big(\langle s \rangle_1, \ldots, \langle s \rangle_n\big)$, is a *sharing* of s. We say that $\langle s \rangle$ is an $\mathcal{S}$*-consistent* sharing of s if for each $k = 1, \ldots, m$ all (correct) players in S_k have the same view on the summands s_k.

For an adversary structure $\mathcal{Z}$, we say that a sharing specification $\mathcal{S}$ is $\mathcal{Z}$*-private* if for any sharing $\langle s \rangle$ according to $\mathcal{S}$ and for any $\mathcal{Z}$-adversary, there exists a summand s_k which this adversary does not know. Formally, $\mathcal{S}$ is $\mathcal{Z}$-private if $\forall (A, E, F) \in \mathcal{Z}\ \exists S \in \mathcal{S} :\ S \cap E = \emptyset$.[2] For an adversary structure $\mathcal{Z}$ with maximal classes $\overline{\mathcal{Z}} = \big\{(\cdot, E_1, \cdot), \ldots, (\cdot, E_m, \cdot)\big\}$, we denote the natural $\mathcal{Z}$-private sharing specification by $S_{\mathcal{Z}} = \big(\mathcal{P} \setminus E_1, \ldots, \mathcal{P} \setminus E_m\big)$. Also, we shall say that a sharing specification $\mathcal{S}$ is *perfectly* $\mathcal{Z}$*-robust* if there exists an efficient (reconstruction) protocol Π such that for any sharing $\langle s \rangle$ according to $\mathcal{S}$ and for any $\mathcal{Z}$-adversary, if each p_i inputs his share $\langle s \rangle_i$ to Π, then Π outputs s. Analogously, we say that $\mathcal{S}$ is *statistically* (resp. *computationally*) $\mathcal{Z}$*-robust* if there exist such an efficient Π which outputs s except with negligible probability in the presence of an unbounded (resp. efficient) $\mathcal{Z}$-adversary.

Remark 3.1 (Linearity of our Secret Sharing). One can easily verify that the above secret sharing scheme satisfies the following linearity property: Given that $\langle s \rangle$ and $\langle t \rangle$ are $\mathcal{S}$-consistent sharings of s and t, respectively, the vector $\langle s \rangle + \langle t \rangle$ (where $+$ denotes the standard component-wise addition of vectors) is an $\mathcal{S}$-consistent sharing of the sum $s+t$. This implies that from the sharings $\langle s \rangle$ and $\langle t \rangle$, the players can compute an $\mathcal{S}$-consistent sharing $\langle s + t \rangle$ of $s + t$ without any interaction, by (locally) adding their respective shares from $\langle s \rangle$ and $\langle t \rangle$ (i.e., p_i computes his share of $\langle s + t \rangle$ as $\langle s + t \rangle_i := \langle s \rangle_i + \langle t \rangle_i$). By iteratively using this trick the players can locally compute a sharing of any linear function of s and t.

[2] Recall that for all $(A, E, F) \in \mathcal{Z} :\ A \subseteq E$.

3.1 Outline

In Section 3.2 we discuss the relation between MPC and SFE. In Sections 3.3-3.5 we handle the case of information-theoretic (i.t.) security. More precisely, in Sections 3.3 and 3.4 we look at information-theoretic security with zero error probability, aka perfect security, whereas in Section 3.5 we look at statistical security, i.e., i.t. with negligible error probability. The high-level SFE protocols for all three sections have the same structure which is thoroughly explained and analyzed in Section 3.3 for the simple case, where only passive and fail corruption are accounted. Finally, in Section 3.6, we handle the computational-security case.

3.2 MPC and/vs. SFE

As we already discussed in the introduction, MPC is the most general type of distributed computation, where players might give inputs and receive outputs several times during the computation. Instead, SFE considers only computation of non-reactive circuits, i.e., circuits where the order in which the gates are to be evaluated is not relevant (of course, if the input to some gate G_2 is the output of an other gate G_1, then G_1 needs to be computed before G_2). The existence of a genera MPC protocol implies the existence of a general SFE protocol, as the MPC protocol can be trivially used also for SFE. However, as we demonstrate by the following example, the opposite is not in general true.

Example 3.1. Consider the player set $\mathcal{P} := \{p_1, p_2\}$, and an adversary who can either passively corrupt p_1 or fail-corrupt p_2. For this setting it is trivial to construct a general SFE protocol. Indeed, let C denote the circuit to be evaluated. The protocol proceeds as follows: p_1 sends his input(s) to p_2 who evaluates C on these input(s) (and his own input) and sends back to p_1 his output(s); if p_1 receives no value from p_2, then he evaluates C himself, where p_2's input(s) is set to a default value corresponding to p_2 not sending his input.[3] Because p_2 cannot be passively corrupted, the above protocol is perfectly private. Furthermore, both p_1 and p_2 receive the output of the circuit evaluated on their intended inputs, unless p_2 crashes in which case it is legitimate that p_1 sets p_2's input to a default value. Hence, the above SFE protocol perfectly securely computes any given (non-reactive) circuit. However,

[3]Note that as p_2 might crash, the function to be computed needs to be defined even when p_2 gives no input.

general MPC is not possible for this setting, as there is no way for the players to keep a private joint state. For simplicity we show this for the case of perfect security, but the argument extends easily also to statistical and computational security. Indeed, let v denote a value in the (perfectly private) joint state. Because p_1 might be passively corrupted, his view should contain no information about v. However, this means that if the adversary fail-corrupts and crashes p_2, then the value v can no longer be recovered.

The above example, demonstrates that there are adversary structures that can be tolerated for SFE but not for MPC. Despite its simplicity, the example brings up the main reason for this separation: in MPC it is essential that the players can keep a private joint state, whereas in SFE it is acceptable that, as long as no information has leaked to the adversary, we can "throw away" the state and restart the computation. In fact, as we shall see in the following sections, there are numerous (infinitely many) structures which can be tolerated for SFE but not for MPC. However, under certain conditions, we can get a reduction of MPC to SFE. In the remaining of this section we give such a generic condition which allows given a general SFE protocol to construct a general MPC protocol.

As we argued in the introduction, in MPC it is important that the players can keep a private joint state of the computation. One way to store such a private state when a $\mathcal{Z}$-adversary is considered is to have all inputs and outputs of any gate that has been computed so far secret-shared with respect to a $\mathcal{Z}$-private sharing specification $\mathcal{S}$. However, in order to make sure that the adversary cannot force the state to be lost, we need the sharing to be robustly reconstructible, i.e., $\mathcal{S}$ should be $\mathcal{Z}$-robust. As we show in the following, given an SFE protocol and a sharing specification $\mathcal{S}$ which is both $\mathcal{Z}$-private and $\mathcal{Z}$-robust we can construct an MPC protocol. The security level of the MPC protocol depends on the security of the SFE protocol and the level of the $\mathcal{Z}$-robustness of $\mathcal{S}$; in particular the security of the MPC protocol is defined by the weakest of the security/robustness assumptions. In the following we show how to construct an MPC protocol out of an SFE protocol, when a $\mathcal{Z}$-robust sharing specification $\mathcal{S}$ is given.

The computation to be done is represented as a circuit C consisting of input, addition, multiplication, and output gates.[4] The circuit is evaluated in a gate-by-gate manner. The reactiveness of C is modeled by assigning to each gate a point in time in which it should be evaluated (alternatively we could model the reactiveness by assuming an order in which the gates are

[4] As we argued in Section 2.5, this suffices to capture also randomized circuits.

evaluated). Note that in a reactive computation the players do not necessarily need to know the complete circuit in advance; it suffices that at each point they know which gate is to be evaluated next. The invariant of the computation is that in each step all intermediate results, i.e., the outputs of all gates which have been processed so far, are secret-shared among the players in the assumed player-set $\mathcal{P}$ according to the $\mathcal{Z}$-private and $\mathcal{Z}$-robust sharing specification $\mathcal{S}$.

The MPC protocol evaluates each type of gate as follows: to have player p give his input s, SFE is invoked to compute the circuit $C_{\langle I \rangle}$ which on input s computes a uniformly random sharing $\langle s \rangle$ of s according to $\mathcal{S}$. The shares $s_1, \ldots, s_{|\mathcal{S}|}$, for such a random sharing are computed by generating $|\mathcal{S}| - 1$ fresh uniformly random values for the shares $s_2, \ldots, s_{|\mathcal{S}|}$ and setting $s_1 := s - \sum_{k=2}^{|\mathcal{S}|} s_k$. To add or multiply two shared values s and t, SFE is invoked to compute the circuit $C_{\langle M \rangle}$ or $C_{\langle A \rangle}$, respectively, which on input the sharings $\langle s \rangle$ and $\langle t \rangle$ of the values s and t computes and outputs a uniformly random sharing of the sum $s+t$ or the product st, respectively, according to $\mathcal{S}$ (note that such a circuit is guaranteed to exist by the $\mathcal{Z}$-robustness property of $\mathcal{S}$, which ensures that both s and t can be computed from the corresponding sharings). For the evaluation of a random gate, SFE is invoked to compute the circuit $C_{\langle R \rangle}$ which takes no input and outputs a uniformly random sharing of a uniformly random value. For the evaluation of an output gate, SFE is invoked to compute the circuit $C_{\langle O \rangle}$ which on input a sharing $\langle s \rangle$ of the output value, reconstructs s and outputs it towards the corresponding player. We give in the following (see next page) a formal description of the protocol MPC and state the achieved security in a theorem. The theorem is generic and applies to all three security levels. The proof follows directly from the assumed properties of $\mathcal{S}$ and the composability of the security definition.

Theorem 3.1. *Assuming that* SFE *is a $\mathcal{Z}$-secure SFE protocol and $\mathcal{S}$ is a $\mathcal{Z}$-robust and $\mathcal{Z}$-private sharing specification, the protocol* MPC *is $\mathcal{Z}$-secure. The security level of* MPC *depends on the corresponding level of the assumed properties, i.e., the $\mathcal{Z}$-security of* SFE *and the $\mathcal{Z}$-robustness of $\mathcal{S}$, as follows: if both properties are perfect, then* MPC *is perfectly secure, otherwise, if at least one property is statistical and the other is perfect or statistical then* MPC *is statistically secure, otherwise, i.e., if at least one is computational and the other is perfect, statistical, or computational, then* MPC *is computationally secure.*

Protocol $\mathsf{MPC}(\mathcal{P}, \mathcal{Z}, \mathcal{S}, C)$**)**

For every gate to be evaluated, do the following:

- *Input gate for p:* Invoke SFE to compute the circuit $C_{\langle I \rangle}$ which on input s from player p outputs a sharing $\langle s \rangle$ of s according to $\mathcal{S}$.
- *Addition gate:* Invoke SFE to compute the circuit $C_{\langle A \rangle}$ which on input two sharings $\langle s \rangle$ and $\langle t \rangle$ according to $\mathcal{S}$ of the values s and t, respectively, outputs a sharing of the sum $s + t$ according to $\mathcal{S}$.
- *Multiplication gate:* Invoke SFE to compute the circuit $C_{\langle M \rangle}$ which on input two sharings $\langle s \rangle$ and $\langle t \rangle$ according to $\mathcal{S}$ of the values s and t, respectively, outputs a sharing of the product st according to $\mathcal{S}$.
- *Random gate:* Invoke SFE to compute the circuit $C_{\langle R \rangle}$ which takes no input and outputs a sharing of a uniformly random value r according to $\mathcal{S}$.
- *Output gate:* Invoke SFE to compute the circuit $C_{\langle O \rangle}$ which on input a sharing $\langle s \rangle$ of the output s, reconstructs s towards the output player.

3.3 Passive/Fail Adversary with Perfect Security

We start the analysis by the relatively simple case, where the adversary can only passively corrupt and fail-corrupt players (no active corruption). The absence of active corruption is modeled by assuming that for all $(A, E, F) \in \mathcal{Z} : A = \emptyset$. To simplify the description, we might denote the adversary structure as a collection of pairs of sets, i.e., write $\mathcal{Z} = \{(E_1, F_1), \ldots, (E_m, F_m)\}$ instead of $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$.

In the following we first prove an exact bound for general SFE in this model, and, subsequently, we extend this result to general (reactive) MPC using Theorem 3.1.

3.3.1 SFE

The exact bound for perfectly $\mathcal{Z}$-secure SFE is described in the following theorem:

Theorem 3.2. *In the model with passive corruption and fail-corruption, where a Broadcast channel is given, a set of players can perfectly $\mathcal{Z}$-securely compute any function if and only if the conditions* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ *and* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ *hold, where*

$$\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z}) \iff \forall (E_1,F_1),(E_2,F_2)\in\mathcal{Z}:\ E_1\cup E_2\cup(F_1\cap F_2)\neq\mathcal{P}$$

$$\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z}) \iff \begin{cases} \exists \text{ an ordering } \big((E_1,F_1),\ldots,(E_m,F_m)\big) \text{ of } \overline{\mathcal{Z}} \text{ s.t.}^5 \\ \forall 1\le i,j\le m,\ i\le j:\ E_j\cup F_i\neq\mathcal{P} \end{cases}$$

For the sufficiency of the above conditions we provide a protocol for perfectly $\mathcal{Z}$-securely evaluating any function. Our protocol is trivially also information-theoretically and computationally secure. The necessity of the conditions is proved in Lemmas 3.8 and 3.9 at the end of this section.

Sufficiency – A perfectly $\mathcal{Z}$-secure SFE protocol

The circuit C to be computed consists of input, addition, multiplication, and output gates. Observe, that a circuit with these four types of gates can be used for computing also randomized circuits: the evaluation of a random gate is simulated by having each player p_i give an (additional) uniformly random input r_i, and compute the output r of the gate as the sum of these random inputs, i.e., $r := \sum_{i=1}^{|\mathcal{P}|} r_i$; as long as at least one player p_i is uncorrupted, his input r_i completely randomizes the sum r.

We build a protocol for perfectly $\mathcal{Z}$-securely computing C. The protocol evaluates C in a gate-by-gate manner, where the input gates are evaluated first, subsequently the addition and multiplication gates, and finally the output gates are evaluated. The invariant of the computation is that in each step of the protocol each intermediate result, i.e., the output of each gate which has been processed so far, is secret-shared among the players in $\mathcal{P}$ according to the natural $\mathcal{Z}$-private sharing specification $\mathcal{S}_{\mathcal{Z}}$.

For each of the four types of gates we build a sub-protocol which will be invoked by the high-level SFE protocol for evaluating any gate of the corresponding type. All the sub-protocol are perfectly private, i.e., they leak no information on their inputs to the adversary. However, they are non-robust,

[5] Recall that $\overline{\mathcal{Z}}$ denotes the maximum classes in $\mathcal{Z}$ (see Section 2.3.1). One can verify that such an ordering exists for $\overline{\mathcal{Z}}$ exactly if it exists for $\mathcal{Z}$.

i.e., they might abort, but when they abort every player learns the same non-empty set $B \subseteq \mathcal{P}$ of incorrect players;[6] we say then that the sub-protocol *aborted with set* B. Note that the perfect privacy of the sub-protocols is ensured even when they abort.

Abortion of a sub-protocol is handled by the high-level SFE protocol as follows: when some sub-protocol aborts with set B, then the SFE protocol aborts with the same B; subsequently, the players in B are eliminated and the whole computation is repeated in a smaller setting, i.e., with player set $\mathcal{P}' := \mathcal{P} \setminus B$ and with adversary structure $\mathcal{Z}'$ which is the restriction of $\mathcal{Z}$ to the players in $\mathcal{P} \setminus \mathcal{P}'$ and includes only the classes of $\mathcal{Z}$ which are consistent with the players in B being incorrect (in our notation $\mathcal{Z}' := \mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B}$). The reason why this "resetting" of the computation does not cause any security loss is the following: first, as the players in $\mathcal{P} \setminus \mathcal{P}'$ are incorrect, their input needs not be considered in the computation (it can be set to a default value). Second, the adversary gets no advantage by forcing the protocol to reset. Indeed, due to the perfect privacy of the sub-protocols, even when they abort the adversary learns no information on the inputs of non-passively corrupted players. Hence, even after such a reset the adversary has still to chose his strategy independently of those inputs.

In the remaining of this section we describe sub-protocols for evaluating each type of gate and, at the end, describe in detail the SFE protocol.

INPUT GATE For the evaluation of an input gate the sub-protocol Share is invoked which is constructed along the lines of the sharing protocol from [Mau02]. Share allows a player p, called the *dealer*, to share his input s among the players in $\mathcal{P}$ according to the sharing specification $\mathcal{S}_{\mathcal{Z}}$ associated with $\mathcal{Z}$. Share might abort with $B = \{p\}$ when p is incorrect.

Protocol Share$(\mathcal{P}, \mathcal{Z}, p, s)$

1. Dealer p chooses the summands $s_2, \ldots, s_{|\mathcal{S}_{\mathcal{Z}}|}$ uniformly at random and sets $s_1 := s - \sum_{k=2}^{|\mathcal{S}_{\mathcal{Z}}|}$.
2. For each $S_k \in \mathcal{S}_{\mathcal{Z}}$: p sends s_k to every $p_i \in S_k$.
3. p broadcasts a heart-bit $b = 1$; if p does not broadcasts a 1-bit then Share aborts with $B = \{p\}$.

[6] Recall that a player p is incorrect if he has deviated from the protocol instructions, which, as no active corruption is considered, happens only when p is fail-corrupted and has crashed.

Lemma 3.3. *Assuming that Broadcast is given, the protocol* Share$(\mathcal{P}, \mathcal{Z}, p, s)$ *has the following properties: (correctness) Either it outputs an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing of s or it aborts with set $B = \{p\}$; it might abort only if the dealer p is fail-corrupted and has crashed before completing his protocol instructions. (privacy) No information on s leaks to the adversary.*

Proof. Correctness follows by inspection of the protocol. For privacy, we observe that independent of whether Share aborts or not, the adversary gets at most his shares from the sharing $\langle s \rangle$. Because $\mathcal{S}_{\mathcal{Z}}$ is $\mathcal{Z}$-private, these shares give the adversary no information on s. □

ADDITION GATE The evaluation of addition gates (and more general of any linear gate) can be done locally using the linearity of the secret-sharing scheme. In particular, from two sharings $\langle s \rangle$ and $\langle t \rangle$, a sharing of $\langle s + t \rangle$ is computed by every player locally adding his shares of the two values, i.e., the share of each p_i is computed as $\langle s + t \rangle_i := \langle s \rangle_i + \langle t \rangle_i$. Because the computation is local, it is trivially perfectly private. We denote the above addition protocol as Add .

MULTIPLICATION GATE To compute a sharing of the product of two shared values, the protocol Mult (see next page) is invoked. Given $\mathcal{S}_{\mathcal{Z}}$-consistent sharings $\langle s \rangle$ and $\langle t \rangle$ of the values s and t, respectively, protocol Mult computes an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing of the product st. The idea, which is based on the multiplication protocol from [BW98, Mau02], is the following: As s and t are shared according to $\mathcal{S}_{\mathcal{Z}}$, we can use the summands $s_1, \ldots, s_{|\mathcal{S}_{\mathcal{Z}}|}$ and $t_1, \ldots, t_{|\mathcal{S}_{\mathcal{Z}}|}$ to compute the product st as $st := \sum_{k,\ell=1}^{|\mathcal{S}_{\mathcal{Z}}|} s_k t_\ell$. To do so, each term $x_{k,\ell} = s_k t_\ell$, is shared by a player $p^{(k,\ell)} \in S_k \cap S_\ell$, i.e., a player who knows both s_k and t_ℓ (if there are more than one such players take the one with the smallest index). Subsequently, every player locally adds his shares of all terms $x_{k,\ell}$ resulting in a sharing of st.

Lemma 3.4. *Assuming that Broadcast is given, $\langle s \rangle$ and $\langle t \rangle$ are $\mathcal{S}_{\mathcal{Z}}$-consistent sharings of s and t, respectively, and the following condition holds: $\forall (E_1, F_1), (E_2, F_2) \in \mathcal{Z} : E_1 \cup E_2 \neq \mathcal{P}$, the protocol* Mult$(\mathcal{P}, \mathcal{Z}, \langle s \rangle, \langle t \rangle)$ *has the following properties: (correctness) Either it outputs an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing of the product st, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. (privacy) No information on $\langle s \rangle$ and $\langle t \rangle$ leaks to the adversary.*

Proof. Both the correctness and the privacy properties follow from the corresponding properties of protocol Share. □

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

1. For every $(S_k, S_\ell) \in \mathcal{S}_\mathcal{Z} \times \mathcal{S}_\mathcal{Z}$: the player $p^{(k,\ell)} \in S_k \cap S_\ell$ with the smallest index computes $x_{k,\ell} := s_k t_\ell$ and $\mathsf{Share}(\mathcal{P}, \mathcal{Z}, p^{(k,\ell)}, x_{k,\ell})$ is invoked; denote the resulting sharing as $\langle x_{k,\ell}\rangle$.
2. Each $p_i \in \mathcal{P}$ locally adds his shares of all the sharings $\langle x_{k,\ell}\rangle$ $(1 \leq k, \ell \leq |\mathcal{S}_\mathcal{Z}|)$.
3. If any of the invocations of Share aborts with set B then Mult also aborts with B.

OUTPUT GATE The last gate-type to consider is the output gate, which is also the trickiest. For the evaluation of such a gate we construct a protocol, called OutputGeneration, which, given as input an $\mathcal{S}_\mathcal{Z}$-consistent sharing of $\langle s\rangle$, outputs s towards every player. Similar to protocols Share and Mult, the protocol OutputGeneration is non-robust and might abort with a non-empty set B of incorrect players. What makes the design of such a protocol tricky is the requirement that when it succeeds it outputs s towards everybody, but when it aborts it leaks no information on s to the adversary.

A first idea for constructing a reconstruction protocol would be to have each p_i broadcast all the summands he knows, and compute s as the sum of the announced summands $s_1, \ldots, s_k$, i.e., $s := \sum_{k=1}^{|\mathcal{S}_\mathcal{Z}|} s_k$. In fact, if we could guarantee that for every $S_k \in \mathcal{S}_\mathcal{Z} : S_k \not\subseteq F^*$, then the above reconstruction protocol would be even robust, as for each S_k there would be at least one player in $S_k \setminus F^*$ who would correctly broadcast s_k. However, as demonstrated by the following example, this condition is not necessary for a non-robust reconstruction protocol with the properties sketched above.

Example 3.2. Let $\mathcal{P} = \{p_1, p_2, p_3\}$ and let $\mathcal{Z}$ be the adversary structure represented by the following table:

	p_1	p_2	p_3
Z_1	e/f	f	
Z_2	e/f		e
Z_3		e	e/f

Assume that a value s is shared according to the sharing specification $\mathcal{S}_\mathcal{Z}$ associated with $\mathcal{Z}$, i.e., there are summands s_1, s_2, and s_3 such that $\langle s\rangle_1 = (s_3)$, $\langle s\rangle_2 = (s_1, s_2)$, and $\langle s\rangle_3 = (s_1)$. Denote by $\langle s\rangle = (\langle s\rangle_1, \langle s\rangle_2, \langle s\rangle_3)$ the corresponding sharing and assume that any two summands give no information on s. It is clear that there is no way for robustly reconstructing s,

as, for example, an adversary of class Z_1 can force p_1 to crash, in which case there is no way of recovering the summand s_3. However, we can announce the summands in a way which guarantees that if the announcing of some summand fails, then we know that at least one of the summands which is unknown to the adversary has not been announced yet. This can be achieved by announcing the summands sequentially in the following order: first s_3, then s_2, and at the end s_1. It is easy to verify that if the announcing of some summand aborts then the adversary has learned at most two of the summands, which gives him no information on s.

The above example demonstrates that even when a sharing is not robustly reconstructible, there might still be an order of announcing the summands (depending on $\mathcal{Z}$) such that if some announcing fails, then we can be sure that some of the summands that have not been announced yet are not known to the adversary. Using the specifics of our sharing scheme, this translates to existence of an *ordering* $\texttt{Ord}(\mathcal{S}_{\mathcal{Z}}) = (S_1, \ldots, S_m)$ of the sets in $\mathcal{S}_{\mathcal{Z}}$,[7] such that if $S_i \subseteq F^*$ then at least one of the summands $s_{i+1}, \ldots, s_m$ is not known to the adversary. Because $\mathcal{S}_{\mathcal{Z}}$ associates each summand s_j to the adversary of class (E_j, F_j) (i.e., the adversary who does not receive s_j), in order to achieve the above property for an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing $\langle s \rangle$ it suffices to require that for the ordering $(E_1, F_1), \ldots, (E_m, F_m)$ of the classes in $\overline{\mathcal{Z}}$ which is imposed by $\texttt{Ord}(\mathcal{S}_{\mathcal{Z}})$, it holds that if $i \leq j$ then $S_j \setminus F_i \neq \emptyset$ (or written differently, $E_j \cup F_i \neq \mathcal{P}$).

The protocol OutputGeneration (see next page), is based on the above idea: It is parameterized by an ordering $\texttt{Ord}(\mathcal{S}_{\mathcal{Z}}) = (S_1, \ldots, S_m)$ of $\mathcal{S}_{\mathcal{Z}}$ with satisfies the condition:

$$\forall i, j \in \{1, \ldots, m\}, i \leq j : S_j \setminus F_i \neq \emptyset.$$

It is easy to verify that, by definition of $\mathcal{S}_{\mathcal{Z}}$, the condition $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ from Theorem 3.2 implies the existence of such an ordering in a straightforwards way, i.e., by setting $S_j := \mathcal{P} \setminus E_j$. The protocol OutputGeneration is given as input an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing $\langle s \rangle$ of some value s and either it reconstructs s towards every player, or it aborts with a set $B \subseteq \mathcal{P}$ of incorrect players. Similarly to Example 3.2, the privacy of the protocol when it aborts is guaranteed as long as the summands of $\langle s \rangle$ which are not given to the adversary are uniformly distributed.

[7] Recall that $S_i := \mathcal{P} \setminus E_i$ for $(E_i, F_i) \in \overline{\mathcal{Z}}$.

Protocol OutputGeneration$\big(\mathcal{P}, \mathcal{Z}, \texttt{Ord}(\mathcal{S}_\mathcal{Z}) = (S_1, \ldots, S_m), \langle s \rangle\big)$

1. For $k = 1, \ldots, m$, the following steps are executed *sequentially*:
   (a) Every $p_\ell \in S_k$ broadcasts s_k.
   (b) If every player in S_k broadcasts $\perp$ then OutputGeneration aborts *immediately* with $B = S_k$.
2. Every $p_j \in \mathcal{P}$ (locally) computes $s := \sum_{k=1}^{m} s_k$ and outputs s.

Lemma 3.5. *Assume that Broadcast is given,* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ *holds,* $\texttt{Ord}(\mathcal{S}_\mathcal{Z}) = (S_1, \ldots, S_m)$ *is the ordering imposed on* $\mathcal{S}_\mathcal{Z}$ *by* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$*, and* $\langle s \rangle$ *is a* $\mathcal{S}_\mathcal{Z}$*-consistent sharing of* s *with the property that those summands that are unknown to the adversary are randomly chosen. Then the protocol* OutputGeneration *either publicly reconstructs* s*, or it aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. When* OutputGeneration *aborts, it leaks no information on* s *to the adversary.*

Proof (sketch). If OutputGeneration does not abort, then every summand s_k has been broadcast from at least one player in S_k. If OutputGeneration aborts with $B = S_i$ then we know that every player in S_i has crashed, i.e., $S_i \subseteq F^\star$. Because the ordering $\texttt{Ord}(\mathcal{S}_\mathcal{Z})$ is the one imposed by $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$, we know that no adversary class $(E_j, F_j) \in \mathcal{Z}$ with $i \leq j$ satisfies the condition $S_i \subseteq F_j$, hence the summand $s^\star$ associated with the actual adversary has not yet been announced. As the summands not known to the adversary are randomly chosen, $s^\star$ serves as a perfect blinding for s. □

THE SFE PROTOCOL We can now describe in detail the protocol SFE which perfectly $\mathcal{Z}$-securely evaluates a given circuit C. For sake of simplicity, we assume that C has one global output. Using known techniques the protocol can be extended to compute circuits with multiple outputs and even private outputs: For the case of multiple outputs, the vector of outputs is embedded in a field of appropriate size and the embedded element is reconstructed. For the case of private outputs, for each output-gate the dedicated output player inputs (in addition to his inputs to C) a one-time pad key used for perfectly blinding his output.

The protocol SFE uses the notation $\mathcal{Z}|^{B \subseteq F}_{\mathcal{P}'}$ which was introduced in Chapter 2 (see Page 35). It is easy to verify that for any set $B \subseteq \mathcal{P}$ of incorrect players, both conditions $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\cdot,\cdot)$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\cdot,\cdot)$ hold for any $(\mathcal{P} \setminus B, \mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B})$ if they hold for the initial player set and adversary structure $(\mathcal{P}, \mathcal{Z})$.

Protocol $\mathsf{SFE}(\mathcal{P}, \mathcal{Z}, C)$

0. Assume that the condition $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ holds, and let $\mathtt{Ord}(\mathcal{S}_{\mathcal{Z}}) = (\mathcal{P} \setminus E_1, \ldots, P \setminus E_m)$ for the assumed ordering $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ of $\overline{\mathcal{Z}}$.
1. *Input stage:* For every input gate in C, Share is invoked to have the input player p_i share his input x_i according to $\mathcal{S}_{\mathcal{Z}}$.[a]
2. *Computation stage:* The gates in C are evaluated as follows:
   - *Addition gate:* Invoke Add to compute a sharing of the sum according to $\mathcal{S}_{\mathcal{Z}}$.
   - *Multiplication gate:* Invoke Mult to compute a sharing of the product according to $\mathcal{S}_{\mathcal{Z}}$.
3. *Output stage:* Invoke $\mathsf{OutputGeneration}(\mathcal{P}, \mathcal{Z}, \mathtt{Ord}(\mathcal{S}_{\mathcal{Z}}), \langle s \rangle)$ for the sharing $\langle s \rangle$ of the public output.
4. If any of the sub-protocols aborts with B, then set $\mathcal{P} := \mathcal{P} \setminus B$, set $\mathcal{Z} := \mathcal{Z}|^{B \subseteq F}_{\mathcal{P} \setminus B}$, and restart the whole computation.

[a] If at a later iteration a player $p_i \notin \mathcal{P}$ should give input, then the players in $\mathcal{P}$ pick the default sharing of a default value.

Lemma 3.6. *Assuming that Broadcast is given, the above SFE protocol is $\mathcal{Z}$-secure if the conditions* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ *and* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ *hold.*

Proof (sketch). It is easy verify that the conditions in the lemma imply all conditions required for the sub-protocols, hence the security of the sub-protocols implies the security of the SFE protocol. In particular, when none of the sub-protocols aborts, then, because Share, Mult, and Add leak no information, and OutputGeneration publicly reveals the output, the simulator can perfectly simulate the adversary's view.
Special care needs to be taken only to handle the fact that the adversary can make the sub-protocols abort. Termination is guaranteed by the fact that in each repetition an additional player in $\mathcal{P}$ is eliminated hence the number of resets is upper bounded by $|\mathcal{P}|$. Furthermore, because any sub-protocol leaks no information even when it aborts, the simulator can again perfectly simulate the adversary's view in every aborting iteration (even without knowing the public output), hence the capacity to abort an iteration does not give the adversary any additional power. □

Necessity of $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ for SFE

We now turn into proving that the conditions $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ are necessary for perfectly $\mathcal{Z}$-secure SFE. In particular, we show that if any of the two is violated, then there are functions which cannot be $\mathcal{Z}$-securely computed. For the case of $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ the impossibility result holds also for statistical security, whereas, as we show in Lemma 3.9, for $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ it even holds for computational security.

We start with the necessity of $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$. We use the following impossibility result from [HM00]:

Lemma 3.7 ([HM00])**.** *If there are sets E_1 and E_2 such that $E_1 \cup E_2 = \mathcal{P}$, then there are functions which cannot be perfectly securely computed in the presence of an adversary who can passively corrupt all the players in any one of the sets E_1 and E_2. The statement holds also for statistical security even when a trusted setup is assumed.*

This directly extends to the following:

Lemma 3.8. *If $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ is violated, then there are functions which cannot be perfectly $\mathcal{Z}$-securely computed. The statement holds also for statistical security even when a trusted setup is assumed.*

The necessity of $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ is stated in Lemma 3.9. The intuition of the proof is the following: if $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ is violated, then with some (non-zero) probability, the adversary might learn the output, and still be able to force so many players to crash that the remaining players do not jointly have enough information on the output.

Lemma 3.9. *If $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ is violated, then there are functions which cannot be perfectly $\mathcal{Z}$-securely evaluated. The statement holds also for statistical and computational security even when a trusted setup is assumed.*

Proof. We only describe the proof for the case of perfect security; the extension to statistical and computational security is straight-forward.

Consider $\mathcal{P}$ and $\mathcal{Z}$ with $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ violated, i.e., for every ordering $\big((A_1,E_1,F_1),\ldots,(A_m,E_m,F_m)\big)$ of $\overline{\mathcal{Z}}$ there exists $i,j \in \{1,\ldots,m\}$ such that $i \leq j$ and $E_j \cup F_i = \mathcal{P}$. Consider the identity function, where every player $p_i \in \mathcal{P}$ inputs some value x_i, and the public output is the vector

$\vec{x} = (x_1, \ldots, x_n)$. To arrive at a contradiction, assume that there exists a perfectly $\mathcal{Z}$-secure SFE protocol for this function. This protocol implicitly defines for every set $L \subseteq \mathcal{P}$ the protocol-round in which the players in L obtain full joint information about the output. We denote the index of this round as $\phi(L)$, i.e., the joint view of the players in L in round $\phi(L)$ contains full information on $\vec{x}$, but their joint view in round $\phi(L)-1$ does not contain full information. The function ϕ implies an ordering $\big((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\big)$ on the adversary classes in $\overline{\mathcal{Z}}$ such that for every $1 \leq i \leq j \leq m:\ \phi(E_i) \leq \phi(E_j)$. Denote by i, j those indices that satisfy $i \leq j$ and $E_j \cup F_i = \mathcal{P}$ (which are assumed to exist). The adversary corrupts (A_i, E_i, F_i) and behaves as follows: Up to round $\phi(E_i) - 1$, the adversary lets the corrupted players behave correctly. In round $\phi(E_i)$, the adversary crashes the players in F_i after receiving their messages for this round. Still, the adversary obtains full information on the output $\vec{y} = \vec{x}$ in round $\phi(E_i)$ (she knows all correct messages that were sent, respectively should have been sent to the players in E_i); however, the players in E_j do not have full information on the output, as $\phi(E_j) \geq \phi(E_i)$, which contradicts the assumed perfect security of the protocol. □

3.3.2 MPC (Reactive)

In this section we describe the necessary and sufficient condition for the evaluation of any (even reactive) circuit (MPC). Because an MPC protocol can be used also for SFE, the conditions $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ are trivially necessary for MPC. However, as we demonstrated in Section 3.2, SFE does not imply MPC. In particular, for MPC we need to ensure that the players can keep a private joint state, which can be implemented by having all the values in the state secret-shared according to a $\mathcal{Z}$-private and $\mathcal{Z}$-robust sharing specification.

As demonstrated in Example 3.2, for robust reconstructibility of values shared with respect to some $\mathcal{S} = (S_1, \ldots, S_m)$, it suffices to make sure that for every summand s_k there is at least one player in S_k who is not fail-corrupted and will announce this summand if instructed to. Hence, if we use $\mathcal{S}_{\mathcal{Z}}$ as our sharing specification, a sufficient condition for robust reconstructibility is that for any $\mathcal{P} \setminus E_i = S_i \in \mathcal{S}_{\mathcal{Z}}$ and any $(E_j, F_j) \in \mathcal{Z}$ it holds that $S_i \setminus F_j \neq \emptyset$. In other words, if the condition $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z})$ holds, where

$$\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z}) \iff \forall (E_1, F_1)(E_2, F_2) \in \mathcal{Z}:\ E_1 \cup F_2 \neq \mathcal{P},$$

then the sharing specification $\mathcal{S}_{\mathcal{Z}}$ associated with $\mathcal{Z}$ is perfectly $\mathcal{Z}$-robust.

In the following lemma we prove the necessity of this condition for the players to be able to keep a private joint state. Note that the necessity applies to all three security levels, i.e., even for computational security independent of whether or not a trusted setup is assumed.

Lemma 3.10. *If* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ *is violated then the players cannot hold a private joint state with perfect security. The statement holds also for statistical and computational security even when a trusted setup is assumed.*

Proof. Consider $\mathcal{P}$ and $\mathcal{Z}$ with $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ violated, i.e., there exists $(E_1,F_1),(E_2,F_2)\in\mathcal{Z}$ with $E_1\cup F_2=\mathcal{P}$ (w.l.o.g. assume that $E_1\cap F_2=\emptyset$). Let v be a private joint state defined by the views of the players in $\mathcal{P}$. Because of the privacy requirement, the probability that the players in E_1 alone can reconstruct the state is negligible. Hence, if the adversary corrupts the class (E_2,F_2) and crashes all the players in F_2, then, with overwhelming probability, the state cannot be reconstructed. □

So far, we have shown that each of the conditions $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ is necessary for MPC. To conclude this section, we show that these two conditions together are also sufficient for MPC. Indeed, it is easy to verify that $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ implies $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$.[8] Hence, when both $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ hold, the protocol SFE from the previous section is perfectly $\mathcal{Z}$-secure. Furthermore, as we argued above, $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ ensures that $\mathcal{S}_{\mathcal{Z}}$ is $\mathcal{Z}$-robust and, by definition, $\mathcal{S}_{\mathcal{Z}}$ is also $\mathcal{Z}$-private. This means that when both $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ hold, then the requirements of Theorem 3.1 are satisfied which implies that there exists a perfectly secure general MPC protocol. In fact, the protocol MPC from Section 3.2 is such a protocol for $\mathcal{S}=\mathcal{S}_{\mathcal{Z}}$.

Putting the pieces together we get the following theorem.

Theorem 3.11. *In the model with passive corruption and fail-corruption where a Broadcast channel is given, a set of players can perfectly* $\mathcal{Z}$*-securely compute any (reactive) computation if and only if the conditions* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ *and* $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ *hold, where*

$$\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z}) \iff \forall (E_1,F_1),(E_2,F_2)\in\mathcal{Z}:\ E_1\cup E_2\cup(F_1\cap F_2)\neq\mathcal{P}$$
$$\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z}) \iff \forall (E_1,F_1),(E_2,F_2)\in\mathcal{Z}:\ E_1\cup F_2\neq\mathcal{P}$$

[8] In fact, when $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ holds, then any possible ordering of the classes of $\overline{\mathcal{Z}}$ can be taken as the ordering required by $\mathrm{CP}^{(\emptyset,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$.

3.4 Adding Active Corruption

In Section 3.3 we proved exact feasibility bounds for perfectly secure MPC and SFE with passive corruption and fail-corruption. In this section we extend this result by adding active corruption. We use the same structure as in Section 3.3. In fact, for our positive results, we use exactly the same high-level SFE and MPC protocols, but we re-design the sub-protocols which they use, so that they can also tolerate active corruption. Recall, that throughout this chapter we assume secure channels and Broadcast, and also that for any class $(A_i, E_i, F_i) \in \mathcal{Z} : E_i \supseteq A_i$ and $F_i \supseteq A_i$.

3.4.1 SFE

We first prove a tight bound for SFE which is stated in the following theorem:

Theorem 3.12. *Assuming that Broadcast is given, a set of players can perfectly $\mathcal{Z}$-securely compute any function if and only if the conditions* $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ *and* $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ *hold, where*

$$\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z}) \Leftrightarrow \begin{cases} \forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : \\ E_1 \cup E_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P} \end{cases}$$

$$\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z}) \Leftrightarrow \begin{cases} \exists \textit{ an ordering } ((A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)) \textit{ of } \overline{\mathcal{Z}} \textit{ s.t.} \\ \forall 1 \leq i, j, k \leq m,\ i \leq j :\ E_j \cup A_i \cup A_k \cup (F_i \cap F_k) \neq \mathcal{P} \end{cases}$$

As in Section 3.3.1 we prove the above lemma by, first, constructing a perfectly secure SFE protocol (sufficiency) and, subsequently, proving that each of the two conditions is necessary for SFE (Lemmas 3.20 and 3.21).

Sufficiency – A perfectly $\mathcal{Z}$-secure SFE protocol

We show how the sub-protocols Share, Add, Mult, and OutputGeneration can be modified so that they tolerate all three corruption types. The modified sub-protocols have similar security properties as the old ones, where, of course, active corruption is also taken in consideration, e.g., in Share an actively corrupted dealer is not bound to share his initial input, but it is still required that an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing of some value is achieved. As before, all the sub-protocols are non-robust, and might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players.

INPUT GATE Along the lines of Section 3.3.1, protocol Share allows a dealer p to share his input s among the players in $\mathcal{P}$ according to the sharing specification $\mathcal{S}_\mathcal{Z}$ associated with $\mathcal{Z}$. Because the dealer might be actively corrupted and try to distribute inconsistent values we need to ensure that the resulting sharing is $\mathcal{S}_\mathcal{Z}$-consistent. This is achieved by bilateral consistency checks and a complaint-and-answer mechanism which are implemented in Steps 3a-3c. Share might abort with $B = \{p\}$, when p is incorrect.

Protocol Share$(\mathcal{P}, \mathcal{Z}, p, s)$

1. Dealer p chooses the summands $s_2, \ldots, s_{|\mathcal{S}_\mathcal{Z}|}$ uniformly at random and sets $s_1 := s - \sum_{k=2}^{|\mathcal{S}_\mathcal{Z}|}$.
2. For each $S_k \in \mathcal{S}_\mathcal{Z}$: p sends s_k to every $p_i \in S_k$ who denotes the received value as $s_k^{(i)}$ ($s_k^{(i)} := \perp$ when no value is received).
3. Execute the following steps for $k = 1, \ldots, |\mathcal{S}_\mathcal{Z}|$:
   (a) Every $p_i \in S_k$ sends $s_k^{(i)}$ to every $p_j \in S_k$, who denotes the received value as $s_k^{(i,j)}$.
   (b) Each $p_j \in S_k$ broadcasts a complaint bit $b_{k,j}$, where $b_{k,j} = 1$ if $s_k^{(j)} = \perp$ or if $s_k^{(i,j)} \notin \{s_k^{(j)}, \perp\}$ for some i, and $b_{k,j} = 0$ otherwise.
   (c) For each complaint which was reported (i.e., $b_{k,j} = 1$ for some j), p broadcasts s_k, and every $p_j \in S_k$ sets $s_k^{(j)}$ to the broadcasted value.
4. If p broadcasts $\perp$ in Step 2c, then Share aborts with $B = \{p\}$.

Lemma 3.13. *Assuming that Broadcast is given, the protocol* Share$(\mathcal{P}, \mathcal{Z}, p, s)$ *has the following properties: (correctness) Either* Share *outputs a $\mathcal{S}_\mathcal{Z}$-consistent sharing of some s', where $s' = s$ unless p is actively corrupted, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. (privacy) No information on s leaks to the adversary.*

Proof. (correctness): Clearly, the protocol only aborts when the dealer is incorrect and broadcast $\perp$. When the protocol does not abort, the consistency of the sharing is guaranteed because for each $S_k \in \mathcal{S}_\mathcal{Z}$, either all correct players in S_k hold the same value for s_k, or they complain and get a consistent value for the corresponding summand from the dealer's broadcast; because, when the dealer is correct, all sent and broadcasted summands are s_k such that $s = \sum s_k$, it is clear that the shared value is s. (privacy): Because $\mathcal{S}_\mathcal{Z}$ is $\mathcal{Z}$-private we know that the summands of corrupted players do not reveal information on s. On the other hand, the dealer only broadcasts summands

for which a complaint is broadcast, i.e., two players (claim to) have different values for that summand. This only happens when the dealer or one of the disputing players is actively corrupted, or when the dealer has crashed. In the first case, the adversary is entitled to know the summand, and in the second case, the summand will not be broadcasted (the dealer has crashed). □

ADDITION GATE The protocol Add for evaluating an addition gates (and more general any linear gate) is the same as in Section 3.3.1 (see Page 51).

MULTIPLICATION GATE The protocol with most modifications (with respect to the one designed in Section 3.3.1) is Mult. The main idea for computing a sharing $\langle st \rangle$ of two shared values s and t is again to first have each term $x_{k,\ell} = s_k t_\ell$ shared, and then have the players locally add their shares of all $x_{k,\ell}$; because $st := \sum_{k,\ell=1}^{|\mathcal{S}_\mathcal{Z}|} s_k t_\ell$, as long as all terms are correctly shared this will result in a sharing of st. However, the existence of actively corrupted players makes the first step much more involved. Indeed, when no active corruption was considered, it was sufficient to have any player who knows both s_k and t_ℓ share the term $x_{k,\ell}$. But here, we need to make sure that the sharing is correct, as an actively corrupted player might not share the right value.

To resolve the above problem, we use the following trick from [Mau02]: First we have every player who knows both s_k and t_ℓ (i.e, every $p_i \in S_k \cap S_\ell$) share $x_{k,\ell} = s_k t_\ell$ according to $\mathcal{S}_\mathcal{Z}$; subsequently, sharings of the pairwise differences of the different sharings of $x_{k,\ell}$ are computed (by each player locally subtracting his corresponding shares) and publicly reconstructed. If all the reconstructed differences equal 0, then the sharing of $s_k t_\ell$ created by the player in $S_k \cap S_\ell$ with the smallest index, is adopted; otherwise both s_k and t_ℓ are announced and a default sharing of $x_{k,\ell} = s_k t_\ell$ is created, e.g., $s_1 := x_{k,\ell}$ and $s_2 = \cdots = s_{|\mathcal{S}_\mathcal{Z}|} = 0$. Note that the reconstruction of the pairwise differences cannot harm the privacy of the inputs as either all pairwise differences equal 0 and reconstructing them gives the adversary no information, or at least one of the players shared a wrong value which means that he is actively corrupted and the adversary already knew s_k and t_ℓ.

To implement the above solution we need a protocol which allows the players in any $S_k \in \mathcal{S}_\mathcal{Z}$ to publicly announce the summand s_k of some sharing $\langle s \rangle$ according to $\mathcal{S}_\mathcal{Z}$, and a protocol for reconstructing such a sharing $\langle s \rangle$. In the following we build the corresponding sub-protocols denoted as PublicAnnounce and PublicReconstruct, respectively. Both protocols are non-robust and might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players.

Furthermore, for the case of PublicAnnounce, when it aborts, then the players learn some information about the actual adversary class $(A^\star, E^\star, F^\star)$ which they will use in the main SFE protocol to recover from this abortion.

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

1. Every $p_i \in S_k$ broadcasts his value for s_k; the players denote the broadcasted value as $s_k^{(i)}$.
2. Let $V \subseteq \mathbb{F}$ denote the set of values v that are "explainable" with some adversary in $\mathcal{Z}$, i.e., for which there is an adversary class $(A, E, F) \in \mathcal{Z}$, such that $\{p_i \in S_k : s_k^{(i)} = \perp\} \subseteq F$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$.
3. If $|V| = 1$ then every $p_j \in \mathcal{P}$ outputs $s_k \in V$, otherwise PublicAnnounce aborts with $B = \{p_i \subseteq S_k : s_k^{(i)} = \perp\}$.

Lemma 3.14. *Assuming that Broadcast is given and* $\forall (A_1, \cdot, \cdot), (A_2, \cdot, \cdot) \in \mathcal{Z}$*:* $S_k \not\subseteq A_1 \cup A_2$*, the protocol* PublicAnnounce *either publicly announces* s_k*, or aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. When it aborts, then there exists an adversary class* $(A, E, F) \in \mathcal{Z}$ *such that* $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$*.*

Proof. Clearly, V contains at least the correct summand s_k, as for this summand the condition of Step 2 is trivially satisfied. Therefore, PublicAnnounce either outputs s_k or aborts. It remains to be shown that when it aborts with B, then $|B| > 0$ and there exists an adversary class $(A, E, F) \in \mathcal{Z}$ such that $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$. Because $s_k \in V$, PublicAnnounce aborts only when there exists a value $v \neq s_k$ with $v \in V$. This implies that there is an adversary class $(A, E, F) \in \mathcal{Z}$ with $\{p_i \in S_k : s_k^{(i)} = \perp\} \subseteq F$ and $\{p_i \in S_k : s_k^{(i)} \notin \{v, \perp\}\} \subseteq A$. Because $v \neq s_k$, we need $\{p_i \in S_k : s_k^{(i)} \neq \perp\} \subseteq A \cup A^\star$, which implies that $S_k \subseteq A^\star \cup A \cup (F^\star \cap F)$. Furthermore, B must be non-empty, because otherwise $S_k \subseteq (A^\star \cup A)$ would hold, contradicting the assumption in the lemma. □

Protocol PublicReconstruct$(\mathcal{P}, \mathcal{Z}, \langle s \rangle)$

1. For every $S_k \in \mathcal{S}_\mathcal{Z}$, PublicAnnounce is invoked to have the correct summand s_k announced. If an invocation of PublicAnnounce aborts with B, then also PublicReconstruct aborts with B.
2. Every $p_j \in \mathcal{P}$ computes $s := \sum_{k=1}^{|\mathcal{S}|} s_k$ and outputs s.

Lemma 3.15. *Assuming that Broadcast is given, $\langle s \rangle$ is an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing, and $\forall k = 1, \ldots, |\mathcal{S}|$, $\forall\ (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z}$: $E_1 \cup A_2 \cup A_3 \neq \mathcal{P}$, then the protocol* PublicReconstruct *either publicly reconstructs s, or aborts with a non-empty set B of incorrect players.*

The proof follows immediately from Lemma 3.14.

A detailed description of protocol Mult is given in the following (see below).

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

1. For every $(S_k, S_\ell) \in \mathcal{S}_{\mathcal{Z}} \times \mathcal{S}_{\mathcal{Z}}$, the following steps are executed:
   (a) Every $p_i \in (S_k \cap S_\ell)$ computes the products $x_{k,\ell} := s_k t_\ell$ and invokes Share$(\mathcal{P}, \mathcal{Z}, p_i, x_{k,\ell})$; denote the resulting sharing as $\langle x_{k,\ell}^{(i)} \rangle$.
   (b) Let p_i denote the player with the smallest index in $(S_k \cap S_\ell)$. For every $p_j \in (S_k \cap S_\ell)$, the difference $\langle x_{k,\ell}^{(j)} \rangle - \langle x_{k,\ell}^{(i)} \rangle$ is computed and, by invoking PublicReconstruct, reconstructed.
   (c) If all differences are 0, then the sharing $\langle x_{k,\ell}^{(i)} \rangle$ of p_i is adopted as sharing of $x_{k,\ell}$, i.e., $\langle x_{k,\ell} \rangle := \langle x_{k,\ell}^{(i)} \rangle$. Otherwise (i.e., some difference is non-zero), PublicAnnounce is invoked to have both s_k and t_ℓ announced, and a default sharing $\langle x_{k,\ell} \rangle$ of $x_{k,\ell} = s_k t_\ell$ is created (e.g., the first summand is set to $x_{k,\ell}$ and the other summands are set to 0).
2. Each player in $\mathcal{P}$ (locally) computes his share of the product st as the sum of his shares of all terms $x_{k,\ell}$.
3. If any of the invoked sub-protocols aborts with B, then also Mult aborts with B.

Lemma 3.16. *Assume that Broadcast is given, $\langle s \rangle$ and $\langle t \rangle$ are $\mathcal{S}_{\mathcal{Z}}$-consistent sharings, and $\forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : E_1 \cup E_2 \cup A_2 \neq \mathcal{P}$ holds. Then the protocol* Mult$(\mathcal{P}, \mathcal{Z}, \langle s \rangle, \langle t \rangle)$ *has the following properties: (correctness) It either outputs an $\mathcal{S}_{\mathcal{Z}}$-consistent sharing of st or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. (privacy) No information on the inputs (i.e., on $\langle s \rangle$ and $\langle t \rangle$) leaks to the adversary.*

Proof (sketch). (correctness) The conditions in the lemma are sufficient for all the invoked sub-protocols (Share, PublicReconstruct, PublicAnnounce). The condition $\forall (A_1, E_1, F_1), (A_2, E_2, F_2), (A_3, E_3, F_3) \in \mathcal{Z} : E_1 \cup E_2 \cup A_2 \neq \mathcal{P}$ ensures that for any pair $(S_k, S_\ell) \in \mathcal{S}_{\mathcal{Z}} \times \mathcal{S}_{\mathcal{Z}} :\ S_k \cap S_\ell = \mathcal{P} \setminus (E_k \cup E_\ell) \not\subseteq A^\star$,

i.e., every $x_{k,\ell}$ is known to at least one player who is not actively corrupted; hence if no invocation of Share aborts and all differences are zero, then the shared values are correct. (privacy) Due to the security of Share, the invocations of Share do not leak information to the adversary. Furthermore, the reconstruction of the pairwise differences leaks no information about the inputs as either they all equal 0, or at least one of the players shared a wrong value which means that he is actively corrupted and the adversary already knew s_k and t_ℓ. Finally, PublicAnnounce is invoked on summands s_k, t_ℓ only when two players in $S_k \cap S_\ell$ contradict each other; again, at least one of these players is actively corrupted, hence the adversary already knows s_k and t_ℓ before PublicAnnounce is invoked. □

OUTPUT GATE For the evaluation of an output gate the approach is the same as in Section 3.3.1: to reconstruct a value s shared according to $\mathcal{S}_\mathcal{Z}$ we announce the summands in the order implied by $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$. This way we are guaranteed that either all summands are announced, or, if the announcing aborts, there is still at least one summand which is not known the adversary and has not been announced yet. More precisely, $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ requires that there exists an ordering $\big((A_1,E_1,F_1),\ldots,(A_m,E_m,F_m)\big)$ of $\overline{\mathcal{Z}}$ such that

$$\forall 1 \le i,j,k \le m,\ i \le j:\ E_j \cup A_i \cup A_k \cup (F_i \cap F_k) \neq \mathcal{P}$$

The summands of $\langle s\rangle$ are announced using PublicAnnounce in the order implied by $\mathtt{Ord}(\mathcal{S}_\mathcal{Z}) := (\mathcal{P}\setminus E_1,\ldots,\mathcal{P}\setminus E_m)$, i.e., first the summand given to $S_1 = \mathcal{P}\setminus E_1$, subsequently the one given to $S_2 = \mathcal{P}\setminus E_2$ and so on. If the announcing of some s_j aborts, then Lemma 3.14 ensures that there exists $(A,E,F)\in\mathcal{Z}$ such that $S_j \subseteq A^\star \cup A \cup (F^* \cup F)$; but, because $S_j = \mathcal{P}\setminus E_j$, the condition $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ ensures that the summand $s_k = s^*$ associated with the actual adversary $(A^\star,E^\star,F^\star)$ has not been announce yet, i.e., $k > j$. In the following, we formally describe protocol OutputGeneration (see next page) and state the achieved security.

Lemma 3.17. *Assume that Broadcast is given, the condition* $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ *holds,* $\mathtt{Ord}(\mathcal{S}_\mathcal{Z}) = (S_1,\ldots,S_m)$ *is the ordering imposed on* $\mathcal{S}_\mathcal{Z}$ *by* $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$*, and* $\langle s\rangle$ *is an* $\mathcal{S}_\mathcal{Z}$*-consistent sharing with the property that those summands that are unknown to the adversary are randomly chosen. Then the protocol* OutputGeneration *either publicly reconstructs* s*, or it aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. When* OutputGeneration *aborts, it does not leak any information on* s *to the actual adversary.*

The proof is along the lines of the proof of Lemma 3.5

Protocol OutputGeneration$(\mathcal{P}, \mathcal{Z}, \mathtt{Ord}(\mathcal{S}_{\mathcal{Z}}) = (S_1, \ldots, S_m), \langle s \rangle)$

1. For $k = 1, \ldots, m$, the following steps are executed *sequentially*:
   (a) PublicAnnounce$(\mathcal{P}, \mathcal{Z}, S_k, s_k)$ is invoked to have the correct summand s_k published.
   (b) If PublicAnnounce aborts with B, then OutputGeneration *immediately* aborts with B.
2. Every $p_j \in \mathcal{P}$ (locally) computes $s := \sum_{k=1}^{m} s_k$ and outputs s.

THE SFE PROTOCOL Plugging the above described sub-protocols into the Protocol SFE (Page 55), yields a perfectly secure SFE protocol.

Lemma 3.18. *Assuming Broadcast, the above SFE protocol is $\mathcal{Z}$-secure if the conditions* $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ *and* $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ *hold.*

Necessity of $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ **and** $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ **for SFE**

To complete the proof of Theorem 3.12 we prove the necessity of the conditions $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ and $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ for perfectly secure SFE in the Lemmas 3.20 and 3.21, respectively.

For the proof of Lemma 3.20 we use the following result from [FHM99]

Lemma 3.19 ([FHM99])**.** *If there are player sets E_1, E_2, and A_3 such that $E_1 \cup E_2 \cup A_3 = \mathcal{P}$, then there are functions which cannot be perfectly securely computed in the presence of an adversary who can passively corrupt the players in any one of the sets E_1 and E_2, or actively corrupt the players in A_3.*

The above lemma can be extended in a straight-forward way to a necessity claim of $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ for SFE.

Lemma 3.20. *If* $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P}, \mathcal{Z})$ *is violated, then there are functions which cannot be perfectly $\mathcal{Z}$-securely computed.*

The necessity of $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ is stated in the following lemma, which can be proved along the lines of the proof of Lemma 3.9.

Lemma 3.21. *If* $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ *is violated, then there are functions which cannot be perfectly $\mathcal{Z}$-securely computed.*

3.4.2 MPC

The tight feasibility bounds for MPC in our setting is proved similar to Section 3.3.2. The bound is stated in the following theorem.

Theorem 3.22. *Assuming that Broadcast is given, a set of players can perfectly $\mathcal{Z}$-securely compute any (even reactive) computation if and only if the conditions $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ hold, where*

$$\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \begin{cases} \forall (A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3) \in \mathcal{Z} : \\ E_1 \cup E_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P} \end{cases}$$

$$\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \begin{cases} \forall (A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3) \in \mathcal{Z} : \\ E_1 \cup A_2 \cup A_3 \cup (F_2 \cap F_3) \neq \mathcal{P} \end{cases}$$

To prove the sufficiency, we observe the following: It is easy to verify that when $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ holds, then protocol PublicReconstruct robustly reconstruct any given $\mathcal{S}_{\mathcal{Z}}$-consistent sharing, i.e., $\mathcal{S}_{\mathcal{Z}}$ is $\mathcal{Z}$-robust. Also, the condition $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ implies the condition $\mathrm{CP}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$, hence the conditions $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ (together) are sufficient for SFE (Theorem 3.12). This means that when both $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ hold, then the requirements of Theorem 3.1 are satisfied which implies that there exists a perfectly secure general MPC protocol. In fact, the protocol MPC from Section 3.2 is such a protocol for $\mathcal{S} = \mathcal{S}_{\mathcal{Z}}$.

For the necessity of the bound in Theorem 3.22, we use Lemma 3.20 (necessity of $\mathrm{CP}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ for SFE), and the following lemma, which implies necessity of $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ for MPC.

Lemma 3.23. *If $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ is violated, then the players cannot hold a private joint state with perfect security.*

Proof. Consider $\mathcal{P}$ and $\mathcal{Z}$ with $\mathrm{CP}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ violated, hence there exist $(A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3) \in \mathcal{Z}$ with $E_1 \cup A_2 \cup A_3 \cup (F_2 \cap F_3) = \mathcal{P}$. Without loss of generality assume that $E_1 = \{p_1\}$, $A_2 = \{p_2\}$, $A_3 = \{p_3\}$, and $F_2 = F_3 = \{p_4\}$. We denote the view of p_i as v_i. To arrive at a contradiction, assume that these views define a secret joint state v. Privacy requires that v_1 does not determine v, hence there exists a different state $v' \neq v$ which could be represented by the views (v_1, v_2', v_3', v_4'). Now consider the following two cases: (i) The secret state is v, and the adversary corrupts (A_2,E_2,F_2) and makes p_4 crash and p_2 take a random view, which (with perhaps negligible probability) could be v_2'. (ii) The secret state is v', and the adversary

corrupts (A_3, E_3, F_3) and makes p_4 crash and p_3 take a random view, which (with perhaps negligible probability) could be v_3. In both cases, the views of the players are $(v_1, v_2', v_3, \perp)$, but the joint state is once v and once $v' \neq v$, contradicting perfect security. □

3.5 Statistical Security

We next consider the case of statistical security, i.e., information-theoretic security with negligible error probability. We deal directly with the most general mixed corruption case, i.e., the adversary can actively corrupt, passively corrupt, and fail-corrupt players simultaneously. For the construction of SFE and MPC protocols the idea is, as before, to use the protocols SFE and MPC from Section 3.3, but re-design the sub-protocols that are used so that they are secure for the weaker bounds described in the present section; of course, the new protocols are not perfectly secure as this would violate the tightness results of the previous section. However, they do achieve perfect security, except with negligible probability. The running time of the suggested protocols is polynomial in the size of their input which includes the description of the basis $\overline{\mathcal{Z}}$ of the corresponding structure $\mathcal{Z}$.[10]

The feasibility bounds for the case of statistical security are the same as in Section 3.3 where only passive corruption and fail-corruption were considered. This means that an actively corrupted player cannot do more damage than a player who is simultaneously passively corrupted and fail-corrupted. In particular, for any actively corrupted player p, the adversary is forced to either allow p to correctly execute his protocol, or have him misbehave in a publicly noticeable way.[11] This is achieved by employing a statistically secure authentication mechanism which unconditionally binds the sender to the messages he sends and cannot be forged except with negligible probability. For this purpose we use an extension of the Information Checking (IC) method from [RB89] which is due to [CDD+99]. We point out that the described protocols do not assume any kind of trusted setup (but they do assume a Broadcast channel). Before proving the exact bounds for SFE and MPC, we describe the Information Checking method which is of independent interest.

[10] As the adversary structure $\mathcal{Z}$ might be exponentially large, our protocols' worst case running time can be exponential in the size of the player set. However, as shown in [HM00], one cannot hope to achieve more for a general SFE or MPC protocol which tolerates *any* adversary structure.

[11] Of course, the adversary is allowed to chose the inputs of actively corrupted players.

3.5.1 Information Checking

An actively corrupted player might send a value to another player and then deny that the value was sent by him. To deal with such behavior, we need a mechanism which binds a player to the messages he sends. In [RB89, CDD+99, BHR07] the Information Checking (IC) method was developed for this purpose, and used to design unconditionally secure protocols tolerating up to $t < n/2$ active cheaters. In this section, we extend the IC method to the general adversaries setting with active, passive, and fail-corruption.

The IC-authentication scheme involves three players, a sender p_s, a recipient p_r, and a verifier p_v, and consists of three protocols, called IC-Setup, IC-Distr, and IC-Reveal. Protocol IC-Distr allows p_s to send a value v to p_r in an authenticated way, so that p_r can, by invoking IC-Reveal, open v to p_v and prove that v was received from p_s. Both IC-Distr and IC-Reveal assume a secret key α known exclusively to p_s and p_v (but not to p_r). This key is generated and distributed in IC-Setup. Note that the same key can be used to authenticate multiple messages.

Informally, the three protocols can be described as follows: In IC-Setup, p_s generates a uniformly random key α and sends it to p_v over the bilaterally secure channel. In IC-Distr, α is used to generate an authentication tag y and a verification tag z for the sent value v. The values (v, y) and z are given to p_r and p_v, respectively. In IC-Reveal, p_r sends (v, y) to p_v, who verifies that (y, z) is a valid authentication/verification-tag pair for v with key α.

Ideally, an IC-authentication scheme should have the following properties: (1) Any value sent with IC-Distr is accepted in IC-Reveal, (2) in IC-Distr, p_v gets no information on v, and (3) only values sent with IC-Distr are accepted in IC-Reveal. However, these properties cannot be (simultaneously) perfectly satisfied. In fact, Property 3 can only be achieved with negligible error probability, as the adversary might guess an authentication tag y' for a $v' \neq v$. Moreover, it can only be achieved when neither p_s nor p_v is passively corrupted, since otherwise the adversary knows α and z.

In our IC-authentication scheme the key α is chosen uniformly at random from $\mathbb{F}$ and the value v is also from $\mathbb{F}$. The authentication and verification tags, y and z, respectively, are such that for some degree-one polynomial $w(\cdot)$ over $\mathbb{F}$, $w(0) = v$, $w(1) = y$, and $w(\alpha) = z$. In other words, (y, z) is a valid IC-pair if $z = (y - v)\alpha + v$. Defining validity this way gives the IC-authentication scheme an additional *linearity* property. In particular, if (y, z) and (y', z') are valid IC-pairs for v and v', respectively, (for the same α) then $(y + y', z + z')$

is a valid IC-pair for $v + v'$. This implies that when some values have been sent with IC-Distr, then p_r and p_v can, without any interaction, compute valid authentication data for any linear combination of those values.

In the following we describe in detail the protocols IC-Setup, IC-Distr, and IC-Reveal of our IC-authentication scheme.

Protocol IC-Setup$(\mathcal{P}, \mathcal{Z}, p_s, p_v)$

1. p_s chooses a value $\alpha \in \mathbb{F}$ uniformly at random and sends it to p_v.

Protocol IC-Distr$(\mathcal{P}, \mathcal{Z}, p_s, p_r, p_v, \alpha, v)$

1. p_s chooses $\hat{v}, y, \hat{y} \in \mathbb{F}$ uniformly at random and sets $z := (y - v)\alpha + v$, and $\hat{z} := (\hat{y} - \hat{v})\alpha + \hat{v}$. p_s sends $v, \hat{v}, y, \hat{y}$ to p_r and $z, \hat{z}$ to p_v. If p_r or p_v receives a $\perp$ as one of those messages then he takes a default pre-agreed value for it.
2. p_r chooses $d \in \mathbb{F}$ uniformly at random and broadcasts the values $d, \hat{v} + dv, \hat{y} + dy$. If p_r broadcasts $\perp$ then IC-Distr terminates (and p_v outputs z).
3. If p_s observes that p_r broadcast consistent values in Step 2 then he broadcasts "ok". Otherwise, p_s broadcasts ("not ok", v), and IC-Distr terminates with a default authentication pair (e.g., (v, v)). If p_s broadcasts $\perp$ then IC-Distr aborts.
4. If $(\hat{y} + dy, \hat{z} + dz)$ is an α-valid authentication pair for $\hat{v} + dv$, then p_v broadcasts "accept". Otherwise, p_v broadcasts ("reject", α, z). If p_v broadcasts $\perp$ then IC-Distr terminates (and p_r outputs (y, v)). Otherwise p_r sets $y := (z - v)\alpha + v$, where α and z denote the broadcast values.

Protocol IC-Reveal$(\mathcal{P}, \mathcal{Z}, p_s, p_r, p_v, \alpha, (y, z), v)$

1. p_r sends (v, y) to p_v.
2. if (y, z) is an valid IC-pair for v (with key α) then p_v accepts v, otherwise he rejects.

Theorem 3.24. *Assuming that Broadcast is given, our IC-authentication scheme has the following properties: (correctness) When* IC-Distr *succeeds p_r learns a value v', where $v' = v$ unless p_s is actively corrupted.* IC-Distr *might abort only when p_s*

is incorrect. (completeness) If IC-Distr *succeeds and p_r is correct then in* IC-Reveal *p_v accepts v'. (privacy)* IC-Distr *leaks no information on v to any player other than p_r. (unforgeability) When neither p_s nor p_v is passively corrupted, and the protocols* IC-Distr *and* IC-Reveal *have been invoked at most polynomially many times, then the probability that an adversary actively corrupting p_r makes p_v accept some v' which was not sent with* IC-Distr *is negligible.*

Proof. Correctness follows by inspection of the protocol. (completeness) Completeness is relevant only if both p_r and p_v are correct. In that case when IC-Distr succeeds (and p_r learns v'), then either $(\hat{y} + dy, \hat{z} + dz)$ is a valid IC-pair for $\hat{v} + dv'$ or p_v reject and broadcasts α and z in Step 4 (and p_r sets $y := (z - v')\alpha + v'$). In both cases, (y, z) is a valid IC-pair for v'. Privacy follows from the fact that the values seen by players other than p_r are independent of v'. (unforgeability) First, observe that the adversary obtains no information on α in any invocation of IC-Distr, as the values seen by players other that p_s and p_v are independent of α. The probability that an adversary actively corrupting p_r can make p_v accept some v' which was not sent with IC-Distr equals the probability of guessing the key α. Moreover, in any invocation of IC-Reveal the adversary learns at most one value $b \in \mathbb{F}$ such that $\alpha \neq b$ and no other information on α. As the size of $\mathbb{F}$ is exponential ($|\mathbb{F}| = 2^\kappa$) and α is chosen uniformly at random from $\mathbb{F}$, the probability of guessing α after polynomially many invocations of IC-Reveal is negligible. □

General IC-signatures

An IC-authentication scheme allows a sender $p_i \in \mathcal{P}$ to send a value v to a recipient $p_j \in \mathcal{P}$, so that p_j can later prove authenticity of v, but *only* towards a dedicated verifier $p_k \in \mathcal{P}$. In our protocols we want to use IC-authentication as a mechanism to bind the sender p_i to the messages he sends to p_j, so that p_j can prove to *every* $p_k \in \mathcal{P}$ that these messages originate from p_i. In [CDD+99], the IC-signatures where introduced for this purpose. These can be seen as semi "digital signatures" with information theoretic security. They do not achieve all properties of digital signatures, but enough to guarantee the security of our protocols.

The protocols used for generation and verification of IC-signatures are called ICS-Sign and ICS-Open, respectively. ICS-Sign allows a player $p_i \in \mathcal{P}$ to send a value v to $p_j \in \mathcal{P}$ signed with an IC-signature. The idea is the following: for each $p_k \in \mathcal{P}$, p_i invokes IC-Distr to send v to p_j with p_k being the verifier, where p_j checks that he receives the same v in all invocations. As

syntactic sugar, we denote the resulting IC-signature by $\text{sig}_{i,j}(v)$. The idea in ICS-Open is the following: p_j announces v and invokes IC-Reveal once for each $p_k \in \mathcal{P}$ being the verifier. Depending on the outcomes of IC-Reveal the players decide to accept or reject v. As we want every $p_i \in \mathcal{P}$ to be able to send messages with ICS-Sign, we need a secret-key setup, where every $p_i, p_k \in \mathcal{P}$ hold a secret key $\alpha_{i,k}$. Such a setup can be easily established by appropriate invocations of IC-Setup.

The decision to accept or reject in ICS-Open has to be taken in a way which ensures that valid signatures are accepted (completeness), and forged signatures are rejected with overwhelming probability (unforgeability). To guarantee completeness, a signature must not be rejected when only actively corrupted players rejected in IC-Reveal. Hence, the players cannot reject the signature when there exists a class $(A_j, E_j, F_j) \in \mathcal{Z}$ such that all rejecting players are in A_j. Along the same lines, to guarantee unforgeability, the players cannot accept the signature when there exists a class $(A_i, E_i, F_i) \in \mathcal{Z}$ such that all accepting players are in E_i. To make sure that the above two cases cannot simultaneously occur, we require $\mathcal{Z}$ to satisfy the following property, denoted as $\text{CS}_{\text{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$:

$$\text{CS}_{\text{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z}) \Leftrightarrow \forall (A_i, E_i, F_i), (A_j, E_j, F_j) \in \mathcal{Z} : \; E_i \cup A_j \cup (F_i \cap F_j) \neq \mathcal{P}$$

In the following we describe in detail the protocols ICS-Setup, ICS-Sign, and ICS-Open of our IC-signatures scheme.

Protocol ICS-Setup$(\mathcal{P}, \mathcal{Z})$

1. For each $p_i, p_k \in \mathcal{P}$: Invoke IC-Setup$(\mathcal{P}, \mathcal{Z}, p_i, p_k)$ and denote the output by $\alpha_{i,k}$.

Protocol ICS-Sign$(\mathcal{P}, \mathcal{Z}, p_i, p_j, v)$

1. For $k = 1, \ldots, |\mathcal{P}|$: Invoke IC-Distr$(\mathcal{P}, \mathcal{Z}, p_i, p_j, p_k, \alpha_{i,k}, v)$ and denote by (y_k, z_k) the resulting IC-pair. If IC-Distr aborts then ICS-Sign also aborts.
2. p_j broadcasts "ok" if he received the same v in all invocations of IC-Distr in Step 1, and "not ok" otherwise.
3. If p_j broadcast "not ok", then p_i broadcasts v and a default signature on v is adopted. If p_i broadcasts $\perp$ then ICS-Sign aborts.

Protocol ICS-Open$(\mathcal{P}, \mathcal{Z}, p_i, p_j, v, \text{sig}_{i,j}(v))$

1. p_j broadcasts v.
2. For each $p_k \in \mathcal{P}$: Invoke IC-Reveal$(\mathcal{P}, \mathcal{Z}, p_i, p_j, p_k, \alpha_{i,k}, (y_k, v), z_k)$.
3. For each $p_k \in \mathcal{P}$: p_k broadcasts "accept", if in IC-Reveal he accepted the same value v as the one which was broadcast in Step 1, and "reject" otherwise. For $x \in \{\text{"accept"}, \text{"reject"}, \perp\}$ denote by B^x the set of players p_k that broadcast x.
4. If $\exists (A, E, F) \in \mathcal{Z} : (B^{\perp} \subseteq F) \wedge (B^{\text{"accept"}} \subseteq E)$ then the players reject v, otherwise they accept it.

Lemma 3.25. *Assuming that Broadcast is given,* $\text{CS}_{\text{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds, our IC-signatures scheme has the following properties: (correctness) When* ICS-Sign *succeeds, then p_j learns a value v', where $v' = v$ unless p_i is actively corrupted.* ICS-Sign *might abort only when p_i is incorrect. (completeness) If* ICS-Sign *succeeds and p_j is correct then in* ICS-Open *all players accept v'. (privacy)* ICS-Sign *leaks no information on v to any player other than p_j. (unforgeability) When p_i is not passively corrupted,*[12] *and the protocols* ICS-Sign *and* ICS-Open *have been invoked at most polynomially many times, then the probability that an adversary actively corrupting p_j can make the players accept some v' which was not sent with* ICS-Sign *is negligible.*

Proof (sketch). Correctness and privacy follow immediately from the corresponding properties of the IC-authentication scheme. (completeness) If ICS-Sign succeeds and p_j is correct, then by completeness of the IC-authentication, only incorrect players reject v', i.e., $B^{\text{"reject"}} \subseteq A^{\star}$. Clearly also $B^{\perp} \subseteq F^{\star}$. If there exists a class $(A, E, F) \in \mathcal{Z}$ satisfying the condition in Step 4 of ICS-Open, then $E \cup A^{\star} \cup (F \cap F^{\star}) = \mathcal{P}$ which violates $\text{CS}_{\text{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$; hence no such class exists and v is accepted. (unforgeability) If only passively corrupted players accept v' in IC-Reveal, then it is rejected. Hence, to make the players accept a v' which was not sent with ICS-Sign the adversary has to make at least one non-passively corrupted player accept v' in IC-Reveal. By the unforgeability property of the IC-authentication (and a union-bound argument) the probability of this event is negligible, even after polynomially many invocations of ICS-Sign and ICS-Open. □

[12] Note that unforgeability can only be achieved when the sender p_i in not passively corrupted, as, otherwise, the adversary knows all the keys held by p_i and can create authentication tags for any value.

Remark 3.2. (*Linearity of IC-signatures)* The linearity property of the IC-authentication scheme is propagated to the IC-signatures. In particular, when some values have be sent by p_i to p_j with ICS-Sign (using the same secret keys), then the players can locally, i.e., without any interaction, compute p_i's signature for any linear combination of those values, by applying the appropriate linear combination on the respective signatures. This process yields a signature which, when p_j is correct, will be accepted in ICS-Open.

3.5.2 SFE

We are now ready to prove the following theorem stating the exact condition for SFE in our model:

Theorem 3.26. *Assuming that Broadcast is given, a set of players can statistically $\mathcal{Z}$-securely compute any function if and only if the conditions* $\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ *and* $\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ *hold, where*

$$\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \forall (A_1,E_1,F_1),(A_2,E_2,F_2) \in \mathcal{Z} : E_1 \cup E_2 \cup (F_1 \cap F_2) \neq \mathcal{P}$$

$$\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \begin{cases} \exists \textit{ an ordering } \big((A_1,E_1,F_1),\ldots,(A_m,E_m,F_m)\big) \textit{ of } \overline{\mathcal{Z}} \textit{ s.t.} \\ \forall 1 \leq i,j \leq m,\ i \leq j :\ E_j \cup F_i \neq \mathcal{P} \end{cases}$$

The necessity of the conditions follows directly from Lemmas 3.8 and 3.9 (Section 3.3.1) and the assumption that for each $(A,E,F) \in \mathcal{Z} : E \cap F \subseteq A$. For the sufficiency we use the SFE protocol from Section 3.3.1, but we modify the sub-protocols Share, Add, Mult, and OutputGeneration to achieve statistical security with the same bounds as in Section 3.3.1, while tolerating all three corruption types. Intuitively, the main modification is that the exchanged messages are authenticated by IC-signatures. Recall that the adversary coming up with a fake signature of a passively corrupted player p_i is not considered a forgery.

To deal with active cheating, we also augment our secret-sharing scheme (described in Page 44) by adding IC-signatures. As before, the secret s is split into summands $s_1,\ldots,s_m \in \mathbb{F}$ that add up to s, where each player might hold several of those summands according to some sharing specification $\mathcal{S}$. However, for each $S_k \in \mathcal{S}$, every $p_i \in S_k$ holds, in addition to the summand s_k, an IC-signature on s_k by every $p_j \in S_k$. As syntactic sugar, we denote by $\mathrm{sig}_{\mathcal{S}}(s)$ the set of all IC-signatures on the summands $s_1,\ldots,s_m$ held by the players. As before, the share of player p_i is the vector $\langle s\rangle_i = (s_{i_1},\ldots,s_{i_\ell})$, where $s_{i_1},\ldots,s_{i_\ell}$ are the summands held by p_i. A sharing $\langle s\rangle$ of s is the vector of all

shares and the corresponding signatures, i.e., $\langle s \rangle = (\langle s \rangle_1, \ldots, \langle s \rangle_n, \text{sig}_{\mathcal{S}}(s))$. We also denote by $[s] = (s_1, \ldots, s_m)$ the vector of summands in $\langle s \rangle$. The definition of consistency of a sharing is also modified to take into account the respective IC-signatures: We say that $\langle s \rangle$ is an *$\mathcal{S}$-consistent* sharing of s if for each $k = 1, \ldots, m$ all (correct) players in S_k have the same view on the summand s_k and hold signatures on it from all other players in S_k, and $\sum_{k=1}^{m} s_k = s$.

In the following we describe the sub-protocol for processing each type of gate.

INPUT GATE As in the previous sections, protocol Share allows a dealer p to share his input s among the players in $\mathcal{P}$ according to the sharing specification $\mathcal{S}_{\mathcal{Z}}$. The protocol is non-robust and might abort with a set $B \subseteq \mathcal{P}$ of incorrect players. In the following, we describe protocol Share (see below) and state the achieved security in a lemma. The proof of the lemma is along the lines of the proof of Lemma 3.13.

Protocol Share$(\mathcal{P}, \mathcal{Z}, p, s)$

1. Dealer p chooses summands $s_2, \ldots, s_{|\mathcal{S}_{\mathcal{Z}}|}$ uniformly at random and sets $s_1 := s - \sum_{k=2}^{|\mathcal{S}_{\mathcal{Z}}|} s_k$.
2. For each $S_k \in \mathcal{S}_{\mathcal{Z}}$: p sends s_k to every $p_i \in S_k$ who denotes the received value as $s_k^{(i)}$ ($s_k^{(i)} := \perp$ when no value is received).
3. Execute the following steps for $k = 1, \ldots, |\mathcal{S}_{\mathcal{Z}}|$:
   (a) For each $p_i, p_j \in S_k$: ICS-Sign$(\mathcal{P}, \mathcal{Z}, p_i, p_j, s_k)$ is invoked to have p_i send s_k to p_j and attach an IC-signature on it; p_j denote the received summand as $s_k^{(i,j)}$ ($s_k^{(i,j)} = \perp$ is no value was received). If ICS-Sign aborts, then Share aborts with $B := \{p_i\}$.
   (b) Each $p_j \in S_k$ broadcasts a complaint bit $b_{k,j}$, where $b_{k,j} = 1$ if $s_k^{(j)} = \perp$ or if $s_k^{(i,j)} \notin \{s_k^{(j)}, \perp\}$ for some i, and $b_{k,j} = 0$ otherwise.
   (c) For each complaint which was reported (i.e., $b_{k,j} = 1$ for some j), p broadcasts s_k, and every $p_j \in S_k$ sets $s_k^{(j)}$ to the broadcasted value.
4. If p broadcasts $\perp$ in Step 3c, then Share aborts with $B = \{p\}$.

Lemma 3.27. *Assuming that Broadcast is given, then protocol* Share$(\mathcal{P}, \mathcal{Z}, p, s)$ *has the following properties: (correctness) It either outputs a $\mathcal{S}_{\mathcal{Z}}$-consistent sharing*

of s', where $s' = s$ unless the dealer p is actively corrupted, or it aborts with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players. (privacy) No information about s leaks to the adversary.

ADDITION GATE The protocol Add for evaluating addition (and linear) gates is the same as in Section 3.3.1 (see Page 51).

MULTIPLICATION GATE A sharing of the product st according to $\mathcal{S}_\mathcal{Z}$ can be computed out of the sharings $\langle s \rangle$ and $\langle t \rangle$ of s and t, respectively, along the lines of Section 3.4.1: for each pair of summands s_k and t_ℓ, the product $x_{k,\ell} = s_k t_\ell$ is shared by some player $p^{(k,\ell)} \in S_k \cap S_\ell$, and a sharing of the product is computed as a sharing of the sum $\sum_{k,\ell=1}^{|\mathcal{S}_\mathcal{Z}|} s_k t_\ell$ (by invocation of protocol Add).

As in Section 3.4.1 the problem is to make sure that each $p^{(k,\ell)}$ correctly shares the term $x_{k,\ell} = s_k t_\ell$. To come around this problem we use the following idea: First, $p^{(k,\ell)}$ shares $s_k t_\ell$ by invoking Share; denote by $x'_{k,\ell}$ the shared value.[14] Next, $p^{(k,\ell)}$ shares the summands s_k and t_ℓ by a protocol, called SumShare, which guarantees that he shares the correct summands. Finally, $p^{(k,\ell)}$ uses the sharings of s_k, t_ℓ, and $x'_{k,\ell}$ in a protocol, called MultProof, which allows him to prove that $x'_{k,\ell} = s_k t_\ell$ without leaking any information on these values to the adversary. In the following we present the sub-protocols SumShare and MultProof, and then give a detailed description of the multiplication protocol. Before describing these sub-protocols, we show how to announce a summand s_k of a sharing $\langle s \rangle$ according to $\mathcal{S}_\mathcal{Z}$ and how to publicly reconstruct $\langle s \rangle$; the corresponding sub-protocols will be used as tools in the construction of SumShare and MultProof.

To publicly announce a summand s_k, each $p_i \in S_k$ broadcasts s_k and opens all the signatures on s_k which he holds (i.e., the signatures on s_k from all players in S_k). If all the signatures announced by p_i are accepted, then the value he announced is taken for s_k. If no $p_i \in S_k$ correctly announces all the signatures, the announcing aborts with $B := S_k$. In the following we describe in detail the corresponding protocol PublicAnnounce (see next page) and state the achieved security in a lemma.

Lemma 3.28. *Assume that Broadcast is given, the condition $\mathsf{CS}_{\mathrm{IC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ holds, the condition $\forall (A,E,F) \in \mathcal{Z} : \; S_k \not\subseteq E$ holds, and no signature is forged. Then protocol* PublicAnnounce *either publicly announces the correct summand s_k, or it aborts with a non-empty set B of incorrect players. It might abort only if $S_k \subseteq F^\star$.*

[14] Note that Share does not guarantee that $x'_{k,\ell} = s_k t_\ell$ when $p^{(k,\ell)}$ is corrupted.

Protocol PublicAnnounce$\big(\mathcal{P}, \mathcal{Z}, S_k, s_k, \sigma_{S_k}(s_k) = \{\text{sig}_{i,j}(s_k) \mid p_i, p_j \in S_k\}\big)$

1. For each $p_i, p_j \in S_k$: ICS-Open$(\mathcal{P}, \mathcal{Z}, p_i, p_j, s_k, \text{sig}_{i,j}(s_k))$ is invoked.
2. If some $p_j \in S_k$ announced s_k and all the signatures he opened on it were accepted, then output s_k (if there are more than one such p_j's then take the one with the smallest index). If no such p_j exists, then PublicAnnounce aborts with $B := S_k$.

Proof. Assume that no signature is forged. Because $\forall (A, E, F) \in \mathcal{Z} : S_k \not\subseteq E$ holds, there exists at least one $p_i \in S_k$ who is not passively corrupted. As p_i's signature is not forged, PublicAnnounce either outputs s_k or it aborts. If at least one $p_j \in S_k$ is correct then the value he announced is accepted in ICS-Open and PublicAnnounce does not abort. Hence, PublicAnnounce might abort only when no player in S_k behaves correctly, i.e., only when $S_k \subseteq F^\star$.□

To publicly reconstruct a shared value the summands are announced one by one by invoking PublicAnnounce. Note that this reconstruction might abort with set B when PublicAnnounce aborts with B. For completeness we describe in the following the protocol PublicReconstruct.

Protocol PublicReconstruct$(\mathcal{P}, \mathcal{Z}, \langle s \rangle)$

1. For $k = 1, \ldots, |\mathcal{S}_\mathcal{Z}|$: PublicAnnounce$(\mathcal{P}, \mathcal{Z}, S_k, s_k, \sigma_{S_k}(s_k))$ is invoked to publicly announce s_k. If it aborts, then PublicReconstruct aborts with $B := S_k$.
2. Each player computes $s := \sum_{k=1}^{|\mathcal{S}_\mathcal{Z}|} s_k$.

Lemma 3.29. *Assume that Broadcast is given,* $\text{CS}_{\text{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds, the following condition holds:* $\forall (A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z} : E_1 \cup E_2 \neq \mathcal{P}$, $\langle s \rangle$ *is an* $\mathcal{S}_\mathcal{Z}$*-consistent sharing, and no signature is forged. Then the protocol* PublicReconstruct *either publicly reconstructs* s*, or aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players.*

We can now describe the protocols SumShare and MultProof. Protocol SumShare allows a player $p \in S_k$ to share a summand s_k of a sharing $\langle s \rangle$ according to $\mathcal{S}_\mathcal{Z}$. Unlike Share, protocol SumShare guarantees that p_i shares the correct value s_k. The idea is similar to the idea used in protocol Mult from Section 3.4.1 to have the terms $x_{k,\ell}$ correctly shared: first every player in S_k shares s_k, by Share; subsequently, sharings of the pairwise differences of the

value shared by p and the values shared by the other players are computed and publicly reconstructed. If all the reconstructed differences equal 0 then the sharing created by player p is accepted. Otherwise, s_k is announced (by invocation of PublicAnnounce) and a default sharing is adopted. A detailed description follows:

Protocol SumShare$(\mathcal{P}, \mathcal{Z}, S_k, p, s_k)$

1. Invoke Share$(\mathcal{P}, \mathcal{Z}, p, s_k)$; denote the resulting sharing as $\langle s_k \rangle$.
2. For each $p_i \in S_k \setminus \{p\}$: invoke Share$(\mathcal{P}, \mathcal{Z}, p_i, s_k)$; denote the resulting sharing as $\langle s_k^{(i)} \rangle$.
3. For every $p_j \in S_k \setminus \{p\}$, the difference $\langle s_k \rangle - \langle s_k^{(j)} \rangle$ is computed and, by invoking PublicReconstruct, reconstructed.
4. If all differences equal 0, then the sharing $\langle s_k \rangle$ of p is adopted. Otherwise (i.e., some difference is non-zero), PublicAnnounce is invoked to have s_k announced, and a default sharing $\langle s_k \rangle$ of s_k is created (e.g., the first summand is set to s_k and the remaining summands are set to 0, and default IC-signatures are locally generated).
5. If any of the invoked sub-protocols aborts with set B then also SumShare aborts with B.
6. In addition to his share of $\langle s_k \rangle$ player p outputs the vector $[s_k]$ of summands.

Lemma 3.30. *Assuming that Broadcast is given, the condition* $\mathrm{CS}_{\mathrm{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds, the conditions* $\forall (A, E, F) \in \mathcal{Z} : \; S_k \not\subseteq E$ *holds, and no signature is forged, the protocol* SumShare$(\mathcal{P}, \mathcal{Z}, S_k, p, s_k)$ *has the following properties: (correctness) Either it outputs an* $\mathcal{S}_{\mathcal{Z}}$*-consistent sharing of* s_k *(*p *also outputs the vector* $[s_k]$ *of summands), or it aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. (privacy) No information about* s_k *leaks to the adversary.*

For the proof, we observe that the conditions in the lemma are sufficient for all the invoked sub-protocols (Share, Reconstruct, PublicAnnounce). Assuming that no signature is forged, both correctness and privacy can be argued along the lines of the proof of Lemma 3.16.

Protocol MultProof allows a player p, called the prover, who has shared three values a, b, and c (and knows the corresponding vectors $[a], [b]$, and $[c]$ of summands) to prove that $c = ab$. The protocol can be seen as a distributed challenge-response protocol with prover p and verifier being all the players in $\mathcal{P}$. On a high level, it can be described as follows: First p shares some

appropriately chosen values. Then the players jointly generate a uniformly random challenge r and expose it, and p answers the challenge. If p's answer is consistent with the sharings of a, b, and c and the sharings which he created in the first step, then the proof is accepted, otherwise it is rejected. MultProof is non-robust and might abort with a set $B \subseteq \mathcal{P}$ of incorrect players.

Protocol MultProof$(\mathcal{P}, \mathcal{Z}, p, \langle a\rangle, \langle b\rangle, \langle c\rangle)$

1. PROVER 'S FIRST MESSAGE: p chooses a random $\beta \in \mathbb{F}$, computes $d = \beta b$, and shares both β and d by invoking Share. Denote by $[\beta]$ and $[d]$ the corresponding vectors of summands.
2. GENERATE RANDOM CHALLENGE: The players jointly generate, using standard techniques, a random value $r \in \mathbb{F}$, and expose it.
3. RESPONSE: p broadcasts the vectors $[\rho] := r[a] + [\beta]$ and $[\rho'] := \rho[b] - r[c] - [d]$, where $\rho = \sum_{\rho_k \in [\rho]} \rho_k$. If p broadcasts $\perp$ then MultProof aborts with set $B = \{p\}$.
4. VERIFICATION:
   - 4.1 If $\sum_{\rho'_k \in [\rho']} \rho'_k \neq 0$ then the proof is rejected.
   - 4.2 For each $S_k \in \mathcal{S}_\mathcal{Z}$, each $p_i \in S_k$ broadcast a complaint bit $b_{k,i}$, where $b_{k,i} = 1$ if $\rho_k \neq ra_k + \beta_k$ or $\rho'_k \neq \rho b_k - rc_k - d_k$, and $b_{k,i} = 0$ otherwise. If p_i broadcasts $\perp$ then MultProof aborts with $B = \{p_i\}$.
   - 4.3 For each complaint (i.e., $b_{k,i} = 1$ for some i), PublicAnnounce is invoked to have the k-th summand of both sharings $\langle ra + \beta\rangle$ and $\langle \rho b - rc - d\rangle$ be announced; let q_k and q'_k denote the announced values, respectively. If $q_k \neq \rho_k$ or $q'_k \neq \rho'_k$, then the proof is rejected, otherwise it is accepted.
5. If any of the sub-protocols aborts with set B, then MultProof also aborts with B.

Lemma 3.31. *Assume that Broadcast is given, the condition* $\mathsf{CS}_{\mathrm{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds, the condition* $\forall (A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z} : \; E_1 \cup E_2 \neq \mathcal{P}$ *holds,* $\langle a\rangle, \langle b\rangle$, *and* $\langle c\rangle$ *are* $\mathcal{S}_\mathcal{Z}$*-consistent sharings, and no signature is forged. Then the protocol* MultProof *has the following properties: (correctness) If* $c = ab$*, then either the proof is accepted or* MultProof *aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. Otherwise (i.e, if* $c \neq ab$*), with overwhelming probability, either the proof is rejected or* MultProof *aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. (privacy) No information about* $\langle a\rangle, \langle b\rangle$, *and* $\langle c\rangle$ *leaks to the adversary.*

Proof (sketch). The conditions on the lemma are sufficient for all the invoked sub-protocols (PublicReconstruct, Share, PublicAnnounce). Assuming that no signature is forged, we prove the claimed properties. (correctness) First, observe that the protocol only aborts with set B if one of the invoked sub-protocols aborts with B or if $B = \{p_i\}$, where p_i broadcast $\perp$ in Step 3, hence B is a non-empty set of incorrect players. Consider the case when SumShare does not abort: We consider two (sub)cases (1): $c = ab$ and (2) $c \neq ab$. In Case 1, correctness follows by inspection of the protocol. In Case 2, we show that the probability that the adversary makes the players accept the proof is negligible. First, observe that, as $\forall S_k \in \mathcal{S}_\mathcal{Z}, (\cdot, E, \cdot) \in \mathcal{Z} : \; S_k \not\subseteq E$, for each $S_k \in \mathcal{S}$ there exist at least one correct player $p \in S_k$ who knows all a_k, b_k, and c_k. Hence, if $[\rho] \neq [ra + \beta]$ or $[\rho'] \neq [\rho b - rc - d]$ some player will complain in Step 4.2 and the adversary will be exposed. This implies that in order for the adversary to make the proof accepted, she has to share β and d in his first message such that all equalities $\rho = ra + \beta$, $\rho' = \rho b - rc - d$, and $\sum_{\rho'_k \in [\rho']} \rho'_k = 0$ hold. However, as $c \neq ab$, for each r there exists a unique such pair (β, d) satisfying all those equalities. Hence, the probability that the adversary makes the players accept the proof equals the probability of guessing r which is negligible. (privacy) If p is passively corrupted then privacy is of no issue. Assume that p is at most fail-corrupted: In Step 1, no information leaks to the adversary. In Step 3 announcing $[\rho]$ and $[\rho']$ leaks no information about $\langle a \rangle, \langle b \rangle$, and $\langle c \rangle$ to the adversary, as the summands which are not known to her are perfectly blinded by the corresponding summands of $\langle \beta \rangle$ and $\langle d \rangle$. Finally, Step 4 is executed only if p did not crash before completing Step 3 and has correctly broadcast $[\rho]$, and $[\rho']$. Hence, only actively corrupted players complain in Step 4 and the corresponding values can be announced without violation of privacy as the adversary already knows them. □

Having described the sub-protocols SumShare and MultProof, the protocol Mult (see next page) is straight-forward.

Lemma 3.32. *Assume that Broadcast is given,* $\mathsf{CS}_{\mathrm{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds, the condition* $\forall (A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z} : \; E_1 \cup E_2 \neq \mathcal{P}$ *holds,* $\langle s \rangle$ *and* $\langle t \rangle$ *are* $\mathcal{S}_\mathcal{Z}$*-consistent sharings, and no signature is forged. Then protocol* $\mathsf{Mult}(\mathcal{P}, \mathcal{Z}, \langle s \rangle, \langle t \rangle)$ *has the following properties except with negligible probability. (correctness): It either outputs an* $\mathcal{S}_\mathcal{Z}$*-consistent sharing of* st *or it aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. (privacy) No information about* $\langle s \rangle$ *and* $\langle t \rangle$ *leaks to the adversary.*

Proof. The conditions on the lemma are sufficient for all invoked sub-protocols (Share, SumShare, MultProof). (correctness) As there are at most

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

1. For every $(S_k, S_\ell) \in \mathcal{S}_\mathcal{Z} \times \mathcal{S}_\mathcal{Z}$, the following steps are executed, where $p^{(k,\ell)}$ denotes the player in $S_k \cap S_\ell$ with the smallest index:
   (a) $p^{(k,\ell)}$ computes $x_{k,\ell} := s_k t_\ell$ and shares it, by Share. Denote by $\langle x_{k,\ell}\rangle$ the resulting sharing.[a]
   (b) $\mathsf{SumShare}(\mathcal{P}, \mathcal{Z}, S_k, p^{(k,\ell)}, s_k)$ and $\mathsf{SumShare}(\mathcal{P}, \mathcal{Z}, S_\ell, p^{(k,\ell)}, t_\ell)$ are invoked. Denote by $\langle s_k\rangle$ and $\langle t_\ell\rangle$ the resulting sharings.
   (c) $\mathsf{MultProof}(\mathcal{P}, \mathcal{Z}, p^{(k,\ell)}, \langle s_k\rangle, \langle t_\ell\rangle, \langle x_{k,\ell}\rangle)$ is invoked. If the proof is rejected then Mult aborts with set $B = \{p^{(k,\ell)}\}$.
2. A sharing of the product st is computed as the sum of the sharings $\langle x_{k,\ell}\rangle$ by repeatedly invoking Add.
3. If any of the invoked sub-protocols aborts with B, then also Mult aborts with B.

[a] In addition to his share of $\langle x_{k,\ell}\rangle$, $p^{(k,\ell)}$ also outputs the vector of summands $[x_{k,\ell}]$.

polynomially many invocations of MultProof the probability that some false proof (i.e., with $x_{k,\ell} \neq s_k t_\ell$) is accepted is negligible. When no false proof is accepted, then correctness follows from the assumed condition, which implies the condition $\forall S_k, S_\ell \in \mathcal{S} : \; S_k \cap S_\ell \neq \emptyset$, which, in turns, implies that for every term $s_k t_\ell$ there is at least one player to share it. (privacy) Privacy follows directly from the privacy property of the invoked sub-protocols. □

OUTPUT GATE For the evaluation of an output gate we use the same protocol as in Section 3.4.1 (protocol OutputGeneration in Page 65) with the difference that the protocol PublicAnnounce designed in this section is used. The following lemma states the achieved security.

Lemma 3.33. *Assume that Broadcast is given, the conditions* $\mathsf{CS}_{\mathrm{IC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *and* $\mathsf{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *hold,* $\mathtt{Ord}(\mathcal{S}_\mathcal{Z}) = (S_1, \ldots, S_m)$ *is an ordering imposed on* $\mathcal{S}_\mathcal{Z}$ *by* $\mathsf{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$*, for all* $(A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z} : E_1 \cup E_2 \neq \mathcal{P}$*,* $\langle s\rangle$ *is an* $\mathcal{S}_\mathcal{Z}$*-consistent sharing with the property that those summands that are unknown to the adversary are randomly chosen, and no signature is forged. Then the protocol* OutputGeneration *either publicly reconstructs* s*, or it aborts with a non-empty set* $B \subseteq \mathcal{P}$ *of incorrect players. If* OutputGeneration *aborts, then the protocol does not leak any information on* s *to the adversary.*

The proof is along the lines of the proofs of Lemmas 3.5 and 3.17.

THE SFE PROTOCOL As in the previous section, for computing any function we can use the protocol SFE which we designed in Section 3.3.1 (where the protocols Share, Mult, and OutputGeneration are instantiated by the ones described in this section). One can verify that, unless some signature is forged, all the protocols perfectly achieve their properties (except for Mult where some negligible error probability is involved). Because there are polynomially many signatures generated by non-passively corrupted players in the computation, Lemma 3.25 ensures that the probability that a signature is forged is negligible. The statistical security of SFE follows from the fact that, conditioned on the event that no signature is forged and no invocation of Mult fails, SFE achieves perfect security; this can be shown along the lines of the proof of Lemma 3.6. We state the achieved security in the following lemma.

Lemma 3.34. *Assuming that Broadcast is given, the protocol* SFE *is statistically $\mathcal{Z}$-secure if the conditions* $\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ *and* $\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$ *hold.*

3.5.3 MPC

The following theorem states the exact bound for feasibility of statistically secure MPC when Broadcast is given:

Theorem 3.35. *Assuming that Broadcast is given, a set of players can statistically $\mathcal{Z}$-securely compute any (reactive) computation if and only if the conditions* $\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ *and* $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ *hold, where*

$$\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \forall (A_1,E_1,F_1),(A_2,E_2,F_2) \in \mathcal{Z} : E_1 \cup E_2 \cup (F_1 \cap F_2) \neq \mathcal{P}$$
$$\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z}) \Leftrightarrow \forall (A_1,E_1,F_1),(A_2,E_2,F_2) \in \mathcal{Z} : E_1 \cup F_2 \neq \mathcal{P}$$

Proof (sketch). The necessity of the bounds follows directly from Lemmas 3.8 and 3.9. For the sufficiency, the idea is the same as in the MPC feasibility proofs from previous sections: First, we observe that $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ ensures that $\mathcal{S}_{\mathcal{Z}}$ is $\mathcal{Z}$-robust (in fact the protocol PublicReconstruct can be used for robust reconstruction). Second, $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ trivially implies $\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P},\mathcal{Z})$, which means that the conditions $\mathrm{CS}^{(A,E,F)}_{\mathrm{MULT}}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P},\mathcal{Z})$ (together) are sufficient for SFE (Theorem 3.26). Hence, the pre-conditions of Theorem 3.1 are satisfied which completes the proof. □

3.6 Computational Security

We complete this chapter by showing that when a Broadcast channel is given, then the conditions $\mathrm{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ and $\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$, described in the previous section, are necessary and sufficient for computationally secure SFE and MPC, respectively. The computational hardness assumption upon which the security of our protocols is based, is the existence of enhanced one-way trapdoor permutations, which is standard in the design of cryptographically secure multi-party protocols. For details on the assumption and its applications the reader is referred to [Gol03]. We state the feasibility result in a single theorem.

Theorem 3.36. *Assuming that Broadcast is given and enhanced one-way trapdoor permutations exist, a set $\mathcal{P}$ of players can computationally $\mathcal{Z}$-securely compute any non-reactive computation (SFE) if and only if $\mathrm{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ holds, and any (even reactive) computation (SFE) if and only if $\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ holds.*

The necessity of the conditions follows directly from Lemmas 3.9 and 3.10. In the following two subsections we sketch an SFE and an MPC protocol which are computationally $\mathcal{Z}$-secure under the corresponding condition.

3.6.1 SFE

The design of our computationally secure SFE protocol is somehow different than in the previous sections. On a high level, the evaluation of the given circuit C proceeds in two stages, called the *computation stage* and the *output stage*. In the computation stage a uniformly random *sharing* of the output of C on inputs provided by the players is computed.[15] For this purpose we use the (non-robust) SFE protocol from [Gol04] for dishonest majority which achieves partial fairness and unanimous abort [GL02]. In the output stage, the sharing of the output is publicly reconstructed along the lines of protocol OutputGeneration from Section 3.5.2. Both stages are non-robust and they might abort with a non-empty set $B \subseteq \mathcal{P}$ of incorrect players, but without violating privacy of the inputs. When this happens, the whole evaluation is repeated among the players in $\mathcal{P} \setminus B$, where the inputs of the players in B are fixed to a default pre-agreed value, and the adversary structure $\mathcal{Z}$ is reduced to the structure $\mathcal{Z}|_{\mathcal{P}\setminus B}^{B\subseteq F}$.

[15] As in Section 3.3.1, we assume, without loss of generality, that the circuit C to be computed has one public output.

As in Section 3.5, we use a modification of the secret-sharing scheme described in Page 44. In particular, the secret s is split into uniformly random summands $s_1, \ldots, s_m$ that sum up to s, which are distributed to the players according to the sharing specification $\mathcal{S}_{\mathcal{Z}}$ (see Page 44 for a detailed description). However, to bind each player to his summand(s), instead of IC-signatures we use perfectly hiding commitments.[16] More precisely, for each summand s_k, all players hold a commitment to s_k such that each $p_i \in S_k$ holds the corresponding decommitment information to open it.

In the heart of our construction lies the SFE protocol from [Gol04], which we denote as Π_{G}. The protocol Π_{G} tolerates a dishonest majority and evaluates any given circuit with partial fairness and unanimous abort. More precisely, it was proved in [Gol04], that this protocol achieves the following properties:[17] Let C be the circuit to be evaluated. There is a player $p \in \mathcal{P}$ (specified by the protocol), such that when p is uncorrupted the circuit C is securely evaluated, otherwise, i.e., when p is corrupted, the adversary can decide either to make *all* players abort the protocol or to allow C to be securely evaluated. As proved in [Gol04] the adversary can decide whether or not the protocol aborts even after having received the outputs of the passively corrupted players; but any other information which the adversary sees during the execution of the protocol, she could have simulated herself given these values. Furthermore, by inspecting the protocol in [Gol04], one can verify that it actually satisfies some additional properties, which are relevant when all three corruption types are considered, namely (1) if p is *correct* then the protocol does not abort, (2) a correct player always gives his (correct) input to the evaluation of C, and (3) a non-actively corrupted player does not give a wrong input (but might give no input if he crashes). By the above properties it is clear that the protocol can abort only if p is incorrect (i.e., $B = \{p\}$).

THE COMPUTATION STAGE Let C be the circuit which we wish to compute. We shall use Π_{G} to do the evaluation. However, as already mentioned, in Π_{G} the adversary might abort the computation even after having learned the outputs of the actively or passively corrupted players. As we are interested in security with fairness, this cannot be allowed. Therefore we use the following trick from [IKLP06]: Instead of C, we evaluate the circuit $C_{\langle C \rangle}$ which computes a uniformly random sharing of the output of C according to the

[16] Note that a perfectly hiding commitment scheme is known to exist if (enhanced) trapdoor permutations exist [GMW86].

[17] In [Gol04] a protocol which uses Broadcast for any exchanged messages is designed and then it is argued how Broadcast can be replaced by a protocol over a point-to-point network. As we are given Broadcast, we consider the protocol without this replacement.

sharing specification $\mathcal{S}_{\mathcal{Z}}$ associated with the considered adversary structure $\mathcal{Z}$. Such a circuit can be efficiently computed from C, as shown in [IKLP06]. With this modification we make sure that even when the adversary forces the protocol to abort with some player p, she gets at most the shares of passively or actively corrupted players, which give her no information on the inputs of other players. Furthermore, because the players learn that p is incorrect, they can eliminate him from $\mathcal{P}$ and repeat the computation in a smaller setting, i.e., in $\mathcal{P} \setminus \{p\}$ and with $\mathcal{Z} = \mathcal{Z}|^{\{p\} \subseteq F}_{\mathcal{P} \setminus \{p\}}$. Clearly, as each time we repeat an additional player is eliminated, after at most $|\mathcal{P}|$ repetitions the evaluation will succeed and output a uniformly random sharing of the output of C.

THE OUTPUT STAGE The output stage is initiated as soon as the computation stage has successfully finished and a sharing of the output of C according to $\mathcal{S}_{\mathcal{Z}}$ has been computed. The idea is the same as in the output stage of protocol SFE, described in Section 3.3.1: The summands of the output sharing are announced sequentially in the order implied by $\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$. This guarantees (as in protocol OutputGeneration) that when the announcing of a summand aborts, then the output stage can abort without violating privacy (the summand associated with the actual adversary has not been announced yet). To announce a summand, protocol PublicAnnounce is invoked which is a trivially modified version of PublicAnnounce from Section 3.5.2 to use openings of commitments instead of signatures.

Lemma 3.37. *Assuming that Broadcast is given and enhanced one-way trapdoor permutations exist, the above described SFE protocol is computationally $\mathcal{Z}$-secure if $\mathrm{CS}^{(A,E,F)}_{\mathrm{NREC}}(\mathcal{P}, \mathcal{Z})$ holds.*

3.6.2 MPC

For realizing MPC we invoke protocol MPC from Section 3.2. As in the previous sections we need to ensure that the created sharing is robustly reconstructible, i.e., that $\mathcal{S}_{\mathcal{Z}}$ is $\mathcal{Z}$-robust. This is the case when the condition $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z})$ holds. Indeed, in that case, the sharing $\langle s \rangle$ can be robustly reconstructed as follows: Each p_i announces all the summands he knows and the corresponding decommitment information; if a decommitment succeeds the corresponding summand is accepted. Note that the computational binding property guarantees that the adversary cannot, except with negligible probability, announce a false summand. Furthermore, when $\mathrm{CS}^{(A,E,F)}_{\mathrm{REC}}(\mathcal{P}, \mathcal{Z})$ holds, then for each $i = 1, \ldots, |\mathcal{S}_{\mathcal{Z}}|$, there exists at least one

player in $S_i = \mathcal{P} \setminus E_i$ (i.e., a player who is given the ith summand) who is not actively corrupted or fail-corrupted and will correctly announce the ith summand. Hence, except with negligible probability, all the correct summands – and only those – are accepted and the sharing can be reconstructed by summing them up. Also, as in the previous sections, the condition $\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ implies the condition $\mathrm{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$, and, therefore, is necessary for SFE (Lemma 3.38). This fact, combined with the $\mathcal{Z}$-robustness and the $\mathcal{Z}$-privacy of $\mathcal{S}_{\mathcal{Z}}$, ensure that all the conditions of Theorem 3.1 are satisfied, which implies that the protocol MPC is a computationally $\mathcal{Z}$-secure general MPC protocol.

Lemma 3.38. *Assuming that Broadcast is given and enhanced one-way trapdoor permutations exist, the above described MPC protocol is computationally $\mathcal{Z}$-secure if $\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ holds.*

3.7 Summarizing

We summarize the exact conditions for feasibility of MPC and SFE in the presence of a mixed active/passive/fail general adversary in the following table.

	SFE	MPC (reactive)
Perf.	$\mathrm{CP}_{\mathrm{MULT}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ & $\mathrm{CP}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$	$\mathrm{CP}_{\mathrm{MULT}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ & $\mathrm{CP}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$
Stat.	$\mathrm{CS}_{\mathrm{MULT}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ & $\mathrm{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$	$\mathrm{CS}_{\mathrm{MULT}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ & $\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$
Com.	$\mathrm{CS}_{\mathrm{NREC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$	$\mathrm{CS}_{\mathrm{REC}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$

Figure 3.1: Exact bounds for SFE and MPC assuming a Broadcast channel

Chapter 4

Byzantine Agreement

In this chapter we study the problem of Byzantine Agreement (BA) in the presence of a mixed general adversary who can actively corrupt, passively corrupt, and fail-corrupt players, simultaneously. As in the previous chapter, the adversary is characterized by an adversary structure $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$. Furthermore, without loss of generality, we assume that for all $(A, E, F) \in \mathcal{Z} \ : \ E \subseteq A$ and $F \subseteq A$. We assume secure channels and a static adversary, i.e., the adversary chooses the class to corrupt, denoted as $(A^\star, E^\star, F^\star)$, at the beginning of the protocol. We prove exact feasibility bounds for BA for all three security levels, i.e., perfect, statistical, and computational. The results in this chapter are from [BFH$^+$08, HZ10b]. We point out that, especially for the current chapter, the privacy of the channels is not necessary for our results, as the security statements hold even if we assume only authenticated (instead of secure) channels.

For statistical and computational security we assume that the players can generate and verify digital signatures. As mentioned in Section 2.8, this assumption can be modeled by assuming a PKI or a digital signatures functionality. In fact, for statistical security, one could use the information-theoretic signatures which were suggested in [SHZI02]. Roughly speaking, the security of these signatures ensures that they are unforgeable even for a computationally unbounded adversary.

4.1 Outline

In Section 4.2 we give the definitions of Broadcast and Consensus and discuss the relation between the two primitives. Subsequently, in Sections 4.3 and 4.4 we prove exact feasibility bounds for each primitive for all three security definitions: perfect security without a setup is handled in Section 4.3, whereas statistical and computational security assuming a trusted setup are handled jointly in Section 4.4. We close the chapter by a discussion on some composability considerations that arise from the definitions.

4.2 Consensus and Broadcast

Byzantine Agreement (BA) comes in two flavors, called Broadcast and Consensus. In the following we discuss the properties of each primitive and give security definitions. In consistence with most of the existing BA literature, we give *property-based* definitions for both primitives. As we point out in Section 4.5, there might be simulatability/composability issues for protocols satisfying (only) such a property-based definition. This does not (necessarily) mean that the primitives are insecure, but rather that one should be aware of this issues when using the primitives within a higher level protocol. Nevertheless, we postpone this discussion for the end of this chapter (Section 4.5) and refer to Chapter 6 for approaches on how to cope with such composability issues.

Usually, in BA literature an adversary who can only actively corrupt players is considered. With such an adversary, a protocol is said to achieve Consensus if every player outputs the same value y (consistency), where if every honest player has input x, then $y = x$ (persistency). Similarly, a protocol is said to achieve Broadcast if every player outputs the same y (consistency), where y equals the sender's input unless he is corrupted (validity).[1]

Remark 4.1. Some definitions of Consensus (for threshold adversaries), require consistency on the output *only* as long as there is a majority of honest players in $\mathcal{P}$. Intuitively, the reason is that if more than half of the players can be corrupted, then there are situations where it is impossible to decide which value to output unless one takes into account the identity of the corrupted players. In the setting of general adversaries, the honest majority requirement would translate to requiring that no two (actively) corruptible player

[1] As in all synchronous protocols a termination condition is also required, stating that every player eventually terminates the protocol.

sets A_1 and A_2 fill up the player set $\mathcal{P}$, in other words, for any two (actively) corruptible sets A_1 and A_2 the condition $A_1 \cup A_2 \neq \mathcal{P}$ holds. In this chapter we adopt a different approach: we define Consensus for an arbitrary adversary structure, and then show that there are conditions (including the above) which are necessary for the existence of a protocol realizing it.

We extend the standard (active-corruption only) definitions of BA in the natural way to capture the effects of adding in the model passive and fail-corruption: For passively corrupted players we require that their (correct) inputs are considered as if they were honest. For any fail-corrupted player p, me make sure that p never gives a wrong input to the computation, but he might give no input, in case he crashes before completing his protocol instructions. More precisely, if p is correct (i.e., he has not crashed) at the end of the protocol, then his (correct) input is considered (in this case p's behavior is indistinguishable from an honest player's); if p has crashed before starting the execution of the protocol, then a default value $\perp$ is taken for his input; finally, if p crashes while executing the protocol, then either his correct input is considered, or $\perp$ is taken for it. The formal definitions follow:

We say that 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 $p_i \in \mathcal{P}$ outputs the same value y.
- (persistency) If every non-actively corrupted $p_i \in \mathcal{P}$ who is alive at the beginning of the protocol has input x then 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.[2]

Along the same lines, we give the definition of Broadcast:

We say that a protocol perfectly *$\mathcal{Z}$-securely realizes Broadcast* with sender p (where we denote p's input by x) among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a $\mathcal{Z}$-adversary:

- (consistency) Every $p_i \in \mathcal{P}$ outputs the same value y.
- (validity) If p is not actively corrupted then $y \in \{x, \perp\}$, where $y = x$ if p is correct until the end of the protocol, and $y = \perp$ if p has crashed before the invocation of the protocol.

[2]When p_i is fail-corrupted and is forced to crash before completing the protocol then, by default, the protocol terminates for p_i at the crashing point.

- (termination) For every non-actively corrupted $p_i \in \mathcal{P}$ the protocol terminates after a finite number of rounds.

The definitions of both Consensus and Broadcast for statistical (resp. computational) security are obtained from the above by requiring that the corresponding properties are satisfied except with negligible probability in the presence of an unbounded (resp. efficient) adversary.

4.2.1 Implementing Broadcast from Consensus

The standard approach for building a Broadcast protocol when a Consensus protocol is given is the following: In a first step, the sender sends his input to every player, and, in a second step, the players invoke Consensus on the received values. The consistency of Consensus ensures that the output will be the same for all correct players, whereas if the sender correctly (i.e, consistently) distributes his input in the first step, the persistency property of Consensus guarantees that the output equals this input.

Unfortunately the above approach does no yield a protocol which satisfies the definition of Broadcast from the previous section, when all three corruption types are considered. The reason is that it gives no guarantees when a fail-corrupted sender crashes before distributing all the messages in the first step. Indeed, in this case the output can be any value, as Consensus is invoked without pre-agreement on the input among the non-actively corrupted players.[3]

To fix the above problem we need to ensure that the players can find out whether or not a non-actively corrupted sender is alive at the end of the first step. For this purpose we construct protocol CDP (CDP stands for Crash Detection Protocol). The idea is the following: first p sends a heart-bit bit to every player indicating that he is alive; subsequently the players invoke Consensus on their view of p's "aliveness". Clearly, when p has crashed before the invocation of CDP then every (correct) player considers him as crashed in Consensus and the output is "crashed" (persistency). Similarly, when p correctly executes the protocol and is alive at the end, then every (correct) player considers him as alive in Consensus and the output is "alive". In the following we include a detailed description of CDP and state the achieved security.

[3]Observe that if we assumed atomic multi-send, then such a behavior would not be possible. Nevertheless, as we argue in Chapter 6, such an assumption is too strong.

Protocol $\mathsf{CDP}(\mathcal{P}, p)$

1. p sends a 1-bit to every $p_j \in \mathcal{P}$; p_j denotes the received bit as b_j ($b_j := 0$ if no bit was received).
2. The players in $\mathcal{P}$ invoke Consensus on inputs $b_1, \ldots, b_n$.
3. Every $p_j \in \mathcal{P}$ outputs "alive" if the output of Consensus is 1, and "crashed" otherwise.

Lemma 4.1. *Assuming Consensus, the protocol* $\mathsf{CDP}(\mathcal{P}, \mathcal{Z}, p)$ *has the following properties: (consistency) The (correct) players agree on the output which is either "*alive*" or "*crashed*". (validity) If* p *is correct until the end of* CDP*, then every (correct) player outputs "*alive*" and if* p *has crashed* before *the invocation of* CDP*, then every (correct) player outputs "*crashed*".*

The protocol CDP can be used to reduce Broadcast to Consensus as follows: Same as before, start by the sender p sending his input x to every player; subsequently, invoke Consensus on the received values, and finally, invoke CDP to check the aliveness of p; if CDP outputs "crashed" then set the output of Broadcast to $\perp$, otherwise ignore the output of CDP. We refer to this reduction/protocol as $\mathsf{Broadcast}^{\mathsf{Cons}}$. For completeness, we include a detailed description of protocol $\mathsf{Broadcast}^{\mathsf{Cons}}$ and state its security in a lemma.

Protocol $\mathsf{Broadcast}^{\mathsf{Cons}}(\mathcal{P}, p, x)$

1. Sender p sends x to every $p_j \in \mathcal{P}$ who denotes the received value as x_j.
2. Invoke Consensus on inputs $x_1, \ldots, x_n$; denote the output by y.
3. Invoke $\mathsf{CDP}(\mathcal{P}, p)$; for every $p_i \in \mathcal{P}$ if the output of CDP is "alive" then output y otherwise output $\perp$.

Lemma 4.2. *Assuming Consensus, the protocol* $\mathsf{Broadcast}^{\mathsf{Cons}}$ *perfectly* $\mathcal{Z}$*-securely realizes Broadcast for any (non-trivial) adversary structure* $\mathcal{Z}$*.*[4] *The statement holds also for statistical and computational security.*

4.2.2 Implementing Consensus from Broadcast

Although Consensus implies Broadcast, the opposite is not always true. In particular, a reduction of Consensus to Broadcast can be secure only when the

[4] A non-trivial adversary structure is one which does not allow all players to be simultaneously corrupted.

adversary structure satisfies the condition described in the following lemma.

Lemma 4.3. *Assuming Broadcast, there exists a protocol which $\mathcal{Z}$-securely realizes Consensus if and only if the following condition holds:*

$$\forall (A_1, E_1, F_1), (A_2, E_2, F_2) \in \mathcal{Z} : \; A_1 \cup A_2 \cup (F_1 \cap F_2) \neq \mathcal{P}$$

The statement holds for all three security levels, i.e., perfect, statistical, and computational.

Proof. The sufficiency can be directly proved through the following Consensus protocol: Every p_i broadcasts his input; the players search among the broadcasted values for some x such that $\exists (A^{(\bar{x})}, E^{(\bar{x})}, F^{(\bar{x})}) \in \mathcal{Z}$: $\{p_i \mid p_i \text{ broadcasted } x_i \notin \{x, \perp\}\} \subseteq A^{(\bar{x})}$ AND $\{p_j \mid p_j \text{ broadcasted } x_j = \perp \} \subseteq F^{(\bar{x})}$; if such an x exists and is unique then the players output x otherwise they output 0. The consistency of this protocol is directly implied by the consistency of the broadcast protocol (the decision is taken on broadcasted values), whereas the persistency follows trivially from the assumption of the lemma (when there is pre-agreement on some x then for $(A^{(\bar{x})}, E^{(\bar{x})}, F^{(\bar{x})}) = (A^{\star}, E^{\star}, F^{\star})$ the condition in the described protocol holds; the uniqueness follows trivially from the assumption).

For the necessity, first we show impossibility of Consensus for the case of two players p_1 and p_2, where any one of them can be actively corrupted, and then, using a simple player-simulation argument, we extend it to the general case:

The two-players case: Assume, towards contradiction that a Consensus protocol for the above two-players case exists, and denote by π_1 and π_2 the programs of p_1 and p_2, respectively. We consider the following three scenarios, where in all scenarios p_1 has input 0, p_2 has input 1, and the adversary allows the corrupted player to correctly execute his protocol (i.e., p_1 and p_2 run π_1 and π_2, respectively, even when they are corrupted):

Scenario 1: p_1 is actively corrupted: Because p_2 is the only honest player and has input 1 he should output 1 with overwhelming probability (persistency).

Scenario 2: p_2 is actively corrupted: Because p_1 is the only honest player and has input 0 he should output 0 with overwhelming probability (persistency).

Scenario 3: No player is actively corrupted. Due to the consistency property of Broadcast, with overwhelming probability p_1 and p_2 should output the same value.

To complete the impossibility proof for the two-players case, observe that in all three scenarios we have exactly the same configuration of the programs π_1

and π_2, hence the distribution of the outputs should be identical, which leads to a contradiction.

Extension to n players: Assume, towards contradiction, that $\exists(A_1, E_1, F_1)$, $(A_2, E_2, F_2) \in \mathcal{Z}$ s.t. $A_1 \cup A_2 \cup (F_1 \cap F_2) = \mathcal{P}$, and there exists a $\mathcal{Z}$-secure Consensus protocol π. Wlog, we assume that $A_1 \cap A_2 = \emptyset$ (otherwise we can consider a strictly weaker adversary structure, where some of the sets A_1 and A_2 is reduced and still the condition holds). Because one possible strategy of a $\mathcal{Z}$-adversary is to crash all the players in $F_1 \cap F_2$ before the protocol starts, π can be trivially transformed to a secure Consensus protocol π' for the players in $\mathcal{P}' = \mathcal{P} \setminus (F_1 \cap F_2)$, where $A_1 \cup A_2 = \mathcal{P}'$. Clearly π' tolerates an adversary who can actively corrupt all the players in any one of the sets A_1 and A_2. But this π' can be used from two players $\hat{p}_1$ and $\hat{p}_2$ to achieve Consensus against an adversary who can corrupt any one of them as follows: $\hat{p}_1$ and $\hat{p}_2$ play for the players in A_1 and A_2, respectively, (every $p_i \in A_j$ is give as input the input of $\hat{p}_j$) and output what they output. The existence of such a protocol contradicts our impossibility proof for the two-players case. □

4.3 Perfect Security (no setup)

We start our investigation for exact feasibility bounds with the case of perfect security without a setup. In fact, our impossibility results hold also for statistical and computational security when no setup is assumed. In other words, the bounds described in the current section are sufficient for perfectly secure BA and necessary even for statistical and computational security without a setup.

We first look at the case of (only) active corruption in Section 4.3.1, and subsequently in Sections 4.3.2-4.3.3 we generalize the results to include also fail-corruption and passive corruption. The absence of some of the three corruption types is modeled, as in the previous chapter, by assuming that in all adversary classes the corresponding corruption-set is empty, e.g., in the model with only active corruption we have that $\forall(A, E, F) \in \mathcal{Z} : E = \emptyset$ AND $F = \emptyset$.

4.3.1 Active corruption

We prove the necessary and sufficient bound for Consensus when only active corruption is considered. Because $E = F = \emptyset$ for every

$(A, E, F) \in \mathcal{Z}$, for simplicity we write $\mathcal{Z} = \{A_1, \ldots, A_m\}$ instead of $\mathcal{Z} = \{(A_1, E_1, F_1), \ldots, (A_m, E_m, F_m)\}$. The bound is given in the following theorem.

Theorem 4.4. *In the model with (only) active corruption if no setup is assumed a set $\mathcal{P}$ of players can $\mathcal{Z}$-securely realize Consensus or Broadcast if and only if the conditions $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$ holds where,*

$$\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z}) \iff \forall A_1, A_2, A_3 \in \mathcal{Z} \ : \ A_1 \cup A_2 \cup A_3 \neq \mathcal{P}$$

The statement holds for all three security levels, i.e., perfect, statistical, and computational security.

The sufficiency is proved by constructing a Consensus protocol. Using this protocol within protocol $\mathsf{Broadcast}^{\mathsf{Cons}}$ yields a Broadcast protocol. For sake of simplicity, we only describe a Consensus protocol with binary input/output domain. Extending such a bit-protocol to a corresponding protocol for arbitrary inputs is straight-forward: Represent the inputs as bit-strings of appropriate (fixed) length and invoke the bit-Consensus protocol for each bit. The necessity of the above theorem is proved by showing that the condition is necessary for Broadcast. The necessity for Consensus follows then from Lemma 4.2 which implies that any conditions which is necessary for Broadcast is also necessary for Consensus.

The Consensus protocol

For the constructions of our consensus protocol we follow the approach of [BGP89]: The idea is to construct weaker consensus primitives, and then compose them in a clever way to achieve Consensus. We construct three such weaker primitives called *Weak Consensus*, *Graded Consensus*, and *King Consensus*. All three provide a persistency guarantee similar to Consensus. However, the consistency guarantee is different for each primitive.

WEAK CONSENSUS Informally, weak consensus guarantees that there are no inconsistencies among the outputs of the (alive) non-actively corrupted players, but some of them might have no output (we say that they output a special symbol "$\mathsf{n/v}$"). Moreover, we have the same persistency guarantee as in Consensus: if the non-actively corrupted players *pre-agree* on some value x, i.e., all they have the same input x, then we get *post-agreement* on x, i.e., they output $y = x$.

To achieve Weak Consensus the idea is the following: first every p_i sends his input x_i to every p_j, and p_j checks whether the received values can be justified by some class in the adversary structure. In particular, p_j checks whether there is a value x such that, according to $\mathcal{Z}$, all players p_i who sent $x_i \neq x$ could have been simultaneously actively corrupted; if such a value exists then p_j adopts it, otherwise p_j outputs "n/v". The protocol WeakConsensus is described in detailed and analyzed in the following where the input of each player p_i is denoted as x_i.

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

1. Each $p_i \in \mathcal{P}$ sends x_i to every p_j; p_j denotes the set of players who sent him 0 (resp. 1) as $P_j^{(0)}$ (resp. $P_j^{(1)}$).
2. Each p_j sets $y_j := \begin{cases} 0 & \text{if } \exists A \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(0)} \subseteq A), \text{ else} \\ 1 & \text{if } \exists A \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(1)} \subseteq A), \text{ else} \\ \text{"n/v"} & \end{cases}$

Lemma 4.5. *Assuming that the condition* $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$ *holds, the protocol* WeakConsensus *has the following properties: (weak consistency) There exists some* $y \in \{0,1\}$ *such that every* $p_j \in \mathcal{P} \setminus A^\star$ *outputs* $y_j \in \{y, \text{"n/v"}\}$*. (persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *has the same input* x *then they all output* $y = x$*.*

Proof. (weak consistency) Assume, towards contradiction, that two players $p_i, p_j \in \mathcal{P} \setminus A^\star$ output $y_i = 0$ and $y_j = 1$, respectively. Denote by $P_i^{(\perp)}$ the set of players that did not send a bit to p_i in Step 1. The set $P_j^{(\perp)}$ is defined accordingly. As $y_i = 0$: $\exists A_i \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus P_i^{(0)} \subseteq A_i$. Similarly, as $y_j = 1$: $\exists A_j \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus P_j^{(1)} \subseteq A_j$. The player set can be written as:

$$\mathcal{P} = P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(0)} = \underbrace{P_i^{(1)} \cup P_i^{(\perp)}}_{\subseteq A_i} \cup \underbrace{(P_i^{(0)} \cap (P_j^{(\perp)} \cup P_j^{(0)}))}_{\subseteq A_j} \cup (P_i^{(0)} \cap P_j^{(1)})$$

Moreover, the players in $P_i^{(0)} \cap P_j^{(1)}$ are clearly actively corrupted as they sent different values to p_i and p_j, i.e., $P_i^{(0)} \cap P_j^{(1)} \subseteq A^\star$. Combining the above we get $\mathcal{P} \subseteq A_i \cup A_j \cup A^*$ which contradicts $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$. (persistency) If all non-actively corrupted players have input 0 (the case of pre-agreement on 1 can be handled symmetrically) then for every $p_i : \mathcal{P} \setminus P_i^{(0)} \subseteq A^\star$. Hence, p_i outputs $y_i = 0$ (it follows from the weak consistency property that the decision of p_i is unique, i.e., the condition of Step 2 cannot hold for $y_i = 0$ and $y_i = 1$ simultaneously). □

GRADED CONSENSUS The next primitive we construct is Graded Consensus. Here, each $p_i \in \mathcal{P}$ outputs a pair (y_i, g_i), where y_i is p_i's actual output-bit and $g_i \in \{0,1\}$ is a bit, called p_i's *grade*. The grade g_i has the meaning of the confidence level of p_i on the fact that all non-actively corrupted players also output y_i. In particular, if $g_i = 1$ for some non-actively corrupted p_i then (p_i knows that) $y_j = y_i$ for every (alive) $p_j \in \mathcal{P} \setminus A^\star$. Moreover, when the non-actively corrupted players pre-agreed on a value x, then they all output x with grade 1.

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

1. The players invoke WeakConsensus$(\mathcal{P}, \mathcal{Z}, \vec{x})$; each p_i denotes his output by z_i (note that $z_i \in \{0, 1, \text{“n/v”}\}$).
2. Each $p_i \in \mathcal{P}$ sends z_i to every p_j. p_j denotes the sets of players who sent him $0, 1$, and “n/v” as $P_j^{(0)}, P_j^{(1)}$, and $P_j^{(\text{“n/v”})}$, respectively.
3. Each p_j sets
$$y_j := \begin{cases} 0 & \text{if } \exists A \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus (P_j^{(0)} \cup P_j^{(\text{“n/v”})}) \subseteq A) \\ 1 & \text{otherwise} \end{cases}$$
$$g_j := \begin{cases} 1 & \text{if } \exists A \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(y_j)} \subseteq A) \\ 0 & \text{otherwise} \end{cases}$$

Lemma 4.6. *Assuming that the condition* $\text{CP}^{(A,\emptyset,\emptyset)}_{\text{CONS}}(\mathcal{P}, \mathcal{Z})$ *holds, the protocol* GradedConsensus *has the following properties: (graded persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *has the same input* $x_i = x$ *then they all output* x *with grade* 1*. (graded consistency) If some* $p_i \in \mathcal{P} \setminus A^\star$ *outputs* $(y_j, g_j) = (y, 1)$ *then every* $p_j \in \mathcal{P} \setminus A^\star$ *outputs* $y_j = y$ *with some* $g_j \in \{0, 1\}$*.*

Proof. Before proving the properties we observe that for any $p_j \in \mathcal{P} \setminus A^\star$, $y_j = 1$ only when $\exists A \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus (P_j^{(1)} \cup P_j^{(\text{“n/v”})}) \subseteq A$. Indeed, as $y_j = 1$, for every $p_i \in \mathcal{P} \setminus A^\star$ it must hold $z_i \in \{1, \text{“n/v”}\}$ (otherwise we would have $\mathcal{P} \setminus (P_j^{(0)} \cup P_j^{(\text{“n/v”})}) \subseteq A^\star$ and p_j would output 0). Hence, the players who sent a value not in $\{1, \text{“n/v”}\}$ are in $A^\star \in \mathcal{Z}$. We next prove the two properties from the lemma: (graded consistency) Assume, towards contradiction, that some $p_i \in \mathcal{P} \setminus A^\star$ outputs $(y_i, g_i) = (0, 1)$ and some $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = 1$ (the case $y_i = 1$ and $y_j = 0$ can be handled symmetrically using the observation at the beginning of this proof). The sets $P_i^{(\perp)}$ and $P_j^{(\perp)}$ are defined as in the proof of Lemma 4.5. As $(y_i, g_i) = (0, 1)$: $\exists A_i \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus P_i^{(0)} \subseteq A_i$. Additionally, as $y_j = 1$: $\exists A_j \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus (P_j^{(1)} \cup P_j^{(\text{“n/v”})}) \subseteq A_j$. The player set can be written as:

$$\begin{aligned}\mathcal{P} &= P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(\text{"n/v"})} \cup P_i^{(0)} \\ &= \underbrace{P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(\text{"n/v"})}}_{\subseteq A_i} \cup \underbrace{(P_i^{(0)} \cap (P_j^{(0)} \cup P_j^{(\perp)}))}_{\subseteq A_j} \cup (P_i^{(0)} \cap (P_j^{(1)} \cup P_j^{(\text{"n/v"})}))\end{aligned}$$

Also, clearly $P_i^{(0)} \cap (P_j^{(1)} \cup P_j^{(\text{"n/v"})}) \subseteq A^\star$, as those players sent different values to p_i and p_j in Step 2. Combining the above we get $\mathcal{P} \subseteq A_i \cup A_j \cup A^\star$ which contradicts $\text{CP}_{\text{CONS}}^{(A,\emptyset,\emptyset)}(\mathcal{P}, \mathcal{Z})$. (graded persistency) If all non actively corrupted players have input 0 (the case of pre-agreement on 1 can be handled symmetrically) then, by the persistency property of WeakConsensus, every $p_k \in \mathcal{P} \setminus A^\star$ has $z_k = 0$ hence for every $p_i : \mathcal{P} \setminus P_i^{(0)} \subseteq A^\star$, and, therefore, p_i outputs $(y_i, g_i) = (0, 1)$ (it follows from the graded consistency property that this decision of p_i is unique.). □

King Consensus. The last sub-primitive we construct is King Consensus. Here, there is a distinguished player $p_k \in \mathcal{P}$, called the *king* and the consistency of the output is guaranteed as long as the king correctly executes the protocol. More precisely, King Consensus guarantees that if the king is correct until the end of the protocol, then all non-actively corrupted players output the same value. Additionally, independent of the king's corruption, if the non-actively corrupted players pre-agreed on a value x, then they all output x. The protocol KingConsensus (see below) is described in the following.

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

1. Invoke GradedConsensus$(\mathcal{P}, \mathcal{Z}, \vec{x})$; each p_i denotes his output by (x_i', g_i).
2. The king p_k sends x_k' to every $p_j \in \mathcal{P}$.
3. Each $p_j \in \mathcal{P}$ sets $y_j := \begin{cases} x_j' & \text{, if } (g_j = 1) \text{ or } (x_k' = \perp) \\ x_k' & \text{, otherwise} \end{cases}$

Lemma 4.7. *Assuming that the condition* $\text{CP}_{\text{CONS}}^{(A,\emptyset,\emptyset)}(\mathcal{P}, \mathcal{Z})$ *holds, the protocol* KingConsensus *with king* p_k *has the following properties: (king consistency) If* $p_k \in \mathcal{P} \setminus A^\star$ *then every* p_j *outputs the same* $y_j = y$*. (persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *has input* $x_i = x$ *then they all output* $y = x$*.*

Proof. (Persistency) When the honest players pre-agree on some y then, by the graded persistency property, they all output y with grade 1 and do not change it. (King Consistency) When the king p_k is honest then he sends the same x_k' to every player. We consider two cases: (1) If some $p_i \in \mathcal{P} \setminus A^\star$

outputs some x'_i with grade $g_j = 1$, then all $p_j \in \mathcal{P} \setminus A^\star$ output $x'_j = x'_i$ in GradedConsensus, and therefore output either x'_i or x'_k in KingConsensus; but as the king is honest $x'_k = x'_i$. (2) If all players have grade 0 then they all output x'_k. □

CONSENSUS Building a Consensus protocol when a protocol for King Consensus is given is straight-forward: repeatedly invoke KingConsensus in sequence, once for each player $p_i \in \mathcal{P}$ being the king. As long as $\mathcal{Z}$ is non-trivial, in at least one of the invocations the king will be uncorrupted and consistency will be reached. As soon as consistency is reached, it is preserved in all future invocations independent of the king's corruption (due to the persistency property of King Consensus). For completeness we include the consensus protocol and state the achieved security.

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

0. Each $p_i \in \mathcal{P}$ sets $x'_i := x_i$.
1. For $k = 1, \ldots, n$ the following steps are run:
   (a) Invoke KingConsensus$(\mathcal{P}, \mathcal{Z}, p_k, (x'_1, \ldots, x'_n))$; each p_j denotes his output as $z_j^{(k)}$.
   (b) Each $p_i \in \mathcal{P}$ sets $x'_i := z_i^{(k)}$.
2. (Output) Each $p_i \in \mathcal{P}$ outputs $y_i := x'_i$.

Lemma 4.8. *Assuming that the condition* $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$ *holds the protocol* Consensus *perfectly $\mathcal{Z}$-securely realizes Consensus.*

Necessity of $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$ **for BA**

The necessity of $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}, \mathcal{Z})$ for Consensus follows trivially by the following lemma which was proved in [LSP82].

Lemma 4.9 ([LSP82])**.** *When no setup is assumed, there exists no secure Consensus protocol among $n = 3$ players which tolerates an adversary that can actively corrupt any one of the three players.*

We point out that the above lemma holds for all three security levels when no setup is assumed. By a simple player-simulation argument we can derive an impossibility proof for the case of an n-players set $\mathcal{P}$ and a $\mathcal{Z}$-adversary as follows:

Corollary 4.10. *When no setup is assumed, if the condition* $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ *is violated then there exists no protocol for* $\mathcal{Z}$*-securely realizing Consensus. The statement holds for all three security levels.*

Proof. Assume, towards contradiction, that $\exists A_1, A_2, A_3 \in \mathcal{Z}$ s.t. $A_1 \cup A_2 \cup A_3 = \mathcal{P}$ and there is a $\mathcal{Z}$-secure protocol π for Consensus. Wlog, we can assume that the sets A_1, A_2, and A_3 are disjoint (otherwise we can consider a strictly weaker adversary structure, where some of the sets A_1, A_2, and A_3 are reduced). Then three players $\hat{p}_1, \hat{p}_2$, and $\hat{p}_3$ can use π to achieve Consensus among them, tolerating an adversary who can actively corrupt any one of them as follows: for $i = 1, \ldots 3$, $\hat{p}_i$ plays for the players in A_i (where every $p_j \in A_i$ is given as input $\hat{p}_i$'s input) and outputs what they output. However, the existence of such a protocol contradicts Lemma 4.9. □

The above lemma establishes the necessity of $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ for Consensus. However, this impossibility does not directly imply that the condition is necessary for Broadcast. In fact, as we see in Section 4.2, there are settings where secure Broadcast protocols exist but secure Consensus protocol do not. In the following we provide a proof of necessity of $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ for Broadcast. The proof is based on a technique inspired by Karlin and Yao [KY84]. A similar approach was also used in [FHW04], but the current proof is more straight-forward. The idea is to show that if $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ is violated, then in any Broadcast protocol there are corruption scenarios, for which the outputs are incompatible with the Broadcast definition. As before, we show the impossibility for three players where any one of them can be actively corrupted. This can be extended to an impossibility proof for $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ along the lines of Corollary 4.10.

Lemma 4.11. *When no setup is assumed, there exists no secure Broadcast protocol among* $n = 3$ *players which tolerates an adversary that can actively corrupt any one of the three players. The statement holds for all three security levels.*

Proof. Assume, towards contradiction, that such a protocol exists, where wlog the sender is p_3, and denote by π_1, π_2, and π_3 the programs of the three player p_1, p_2, and p_3, respectively. We consider the following three scenarios which are also depicted in Figures 4.3.1-4.3.3. In all figures, the gray area correspond to the actively corrupted player, and inside this area the strategy of the adversary is depicted. The channels which appear to be cut have the meaning that the adversary will block all message sent through them.

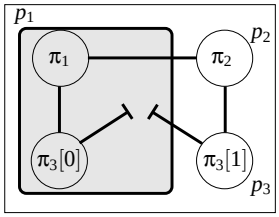

Figure 4.3.1

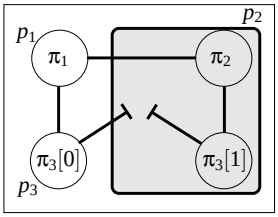

Figure 4.3.2

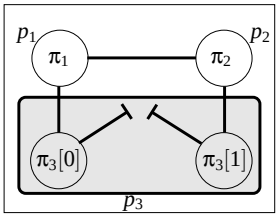

Figure 4.3.3

Scenario 1: Sender p_3 has input 1; the adversary (actively) corrupts p_1 and has him run the following strategy: the programs π_1 and π_3 are (locally) invoked but π_3 is run with input 0. Note that as the programs are given and there is no setup assumption (in particular there are no keys which are not known to the adversary), the adversary can locally run these programs. The adversary connects these programs with each other as shown in Figure 4.3.1. Because the sender p_3 has input 1 and is not actively corrupted, p_2 must output 1 with overwhelming probability (validity).

Scenario 2: Sender p_3 has input 0; the adversary (actively) corrupts p_2 and has him run the following strategy: the programs π_2 and π_3 are (locally) invoked but π_3 is run with input 1. (again, this is a valid strategy as there is no setup). The adversary connects the programs with each other as shown in Figure 4.3.2. Because the sender p_3 has input 0 and is not actively corrupted, p_1 must output 0 with overwhelming probability (validity).

Scenario 3: The adversary (actively) corrupts p_3 and has him run the following strategy: two instances of the program π_3 are locally invoked (simultaneously), one with input 0 and one with input 1. The adversary connects the instances with each other as shown in Figure 4.3.3. Due to the consistency property of Broadcast, with overwhelming probability the uncorrupted players p_1 and p_2 should output the same value.

To complete the proof, observe that in all three figures we have exactly the same configuration of the programs π_1, π_2, and π_3, hence the distribution of the outputs should be identical, which leads to a contradiction. □

Corollary 4.12. *When no setup is assumed, if the condition* $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ *is violated then there exists no protocol which* $\mathcal{Z}$*-securely realizes Broadcast. The statement holds for all three security levels.*

4.3.2 Active/fail corruption – The Dual-failure Model

The condition $\mathrm{CP}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ can be extended to a necessary condition for BA tolerating an active/fail adversary by requiring that for any set $\mathcal{P}' \subseteq \mathcal{P}$, if we consider the adversary structure $\mathcal{Z}' := \mathcal{Z}|^{\mathcal{P}' \subseteq F}_{\mathcal{P}\setminus\mathcal{P}'}$, i.e. the restriction of $\mathcal{Z}$ to the classes that are consistent to the players in $\mathcal{P}'$ being fail-corrupted, then the condition $\mathrm{CP}^{(A,E,\emptyset)}_{\mathrm{CONS}}(\mathcal{P}\setminus\mathcal{P}',\mathcal{Z}')$ should hold. Indeed, if this is not the case then the adversary can force all players in $\mathcal{P}'$ crash from the beginning, and make it impossible for any of the remaining players to broadcast his input (in the set of alive players, there are three actively corruptible sets that violate the necessary condition for broadcast). This leads naturally to necessity of the condition $\mathrm{CP}^{(A,\emptyset,F)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ for Consensus and Broadcast, where:[5]

$$\mathrm{CP}^{(A,\emptyset,F)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z}) \iff \begin{cases} \forall (A_1,F_1),(A_2,F_2),(A_3,F_3) \in \mathcal{Z}: \\ A_1 \cup A_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P} \end{cases}$$

This *dual failure* model, i.e., an adversary who can actively corrupt and fail-corrupt players, was considered by Altmann, Fitzi, and Maurer [AFM99] who showed that the above condition is sufficient and necessary for Broadcast and Consensus. We state this result in the following theorem.

Theorem 4.13 ([AFM99])**.** *In the model with active corruption and fail-corruption (dual-failure model) if no setup is assumed a set $\mathcal{P}$ of players can perfectly $\mathcal{Z}$-securely realize Consensus if and only if the conditions $\mathrm{CP}^{(A,\emptyset,F)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ holds.*

Similarly to the other theorems for the perfect security case, the above result holds also for statistical and computational security when no setup is assumed. We point out, that the Broadcast definition from [AFM99] is slightly weaker than the one used here, as it does not require that a fail-corrupted sender who crashes during the protocol broadcasts either his correct input or no input. Nevertheless, using the Consensus protocol from [AFM99] and the reduction $\mathsf{Broadcast}^{\mathsf{Cons}}$, we directly obtain a protocol which satisfies our Broadcast definition. Furthermore, similarly to many other results on (property-based) BA, the result from [AFM99] assumes only authenticated channels.

[5] Again, for simplicity we omit the (empty) sets of passively corrupted players from the notation of the adversary structure and write $\mathcal{Z} = \{(A_1,F_1),\ldots,(A_m,F_m)\}$ instead of $\mathcal{Z} = \{(A_1,E_m,F_1),\ldots,(A_m,E_m,F_m)\}$.

4.3.3 Adding passive corruption

Considering all three corruption types simultaneously is an extension of the dual-failure model, which has no effect on the bounds for BA when no setup is assumed. In particular, the BA protocols suggested in the previous sections achieve their specification even when every player in $\mathcal{P}$ is passively corrupted. Intuitively, this happens because, first, there is no setup which could potentially include information known exclusively to certain players, second, the designed protocols are secure even in the authenticated channels model, and, third, the (property-based) definition of BA has no privacy requirements;[6] hence the adversary gets no advantage by passively corrupting players and, thereby, obtaining access to their view. For completeness we state the exact bound in form of a theorem.

Theorem 4.14. *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 conditions $\mathrm{CP}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P},\mathcal{Z})$ holds where,*

$$\mathrm{CP}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P},\mathcal{Z}) \Longleftrightarrow \begin{cases} \forall (A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3) \in \mathcal{Z} \ : \\ A_1 \cup A_2 \cup A_3 \cup (F_1 \cap F_2 \cap F_3) \neq \mathcal{P} \end{cases}$$

The statement holds for all three security levels, i.e., perfect, statistical, and computational security.

4.4 Statistical and Computational Security (with setup)

In this section we prove necessary and sufficient conditions for Consensus and Broadcast with statistical and computational security. As already mentioned, the definition is the one described in Section 4.2, where the respective properties are satisfied *except with negligible probability*. In fact, our protocols assume a setup allowing generation and verification of digital signatures, and perfectly satisfy these properties unless some signature is forged. For simplicity, we assume that the signature on any message includes enough information to uniquely identify the point in the corresponding protocol in which the message was generated, e.g., a protocol ID, a round ID, and a message ID.

[6] As we shall see in the last section of this chapter, this is the main reason for certain simulatability/composability issues of the property-based definition of BA which we announced in Section 4.2.

We recall the reader that the reduction of Broadcast to Consensus described in Section 4.2.1, i.e., the protocol $\mathsf{Broadcast}^{\mathsf{Cons}}$, is valid also for computational and statistical security. Therefore, as in the previous section, in most cases we only construct protocols for Consensus and give impossibility proofs for Broadcast.

4.4.1 Active corruption

For the case of only active corruption, Dolev and Strong [DS82] (see also [Fit03] for a simpler description) described a Broadcast protocol which tolerates an arbitrary $t < n$ number of corrupted players. This implies that any non-trivial adversary structure can be tolerated for Broadcast. Interestingly, the same does not hold for Consensus which provably cannot be realized when the player set can be split in two sets each of which can be actively corrupted. In particular, the following theorem holds. The proof of the theorem follows directly from Lemma 4.3 and the fact that the condition of the theorem is sufficient for the Dolev-Strong broadcast protocol.

Theorem 4.15. *In the model with (only) active corruption, if a setup allowing statistically (resp. computationally) secure signatures is assumed, then a set of players $\mathcal{P}$ can statistically (resp. computationally) $\mathcal{Z}$-securely realize Consensus if and only if the condition $\mathrm{CS}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ holds, where*

$$\mathrm{CS}^{(A,\emptyset,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z}) \Longleftrightarrow \forall A_1, A_2 \in \mathcal{Z} \ : \ A_1 \cup A_2 \neq \mathcal{P}$$

4.4.2 Active/passive corruption

Adding passive corruption turned out to be trivial in the case of perfect security (Section 4.3.3), as the initial perfectly secure protocols tolerate arbitrary many passively corrupted players. Depending on the assumptions on the privacy of the setup, this might or might not be the case when statistical or computational security is considered. In particular, one can easily verify that if we assume that the adversary has no access (not even read-access) to the private signing keys of the passively corrupted players, then the Dolev-Strong Broadcast protocol can tolerate arbitrary many passively corrupted players in addition to the active corruption.

One could find justifications for the above model where the signing keys of a player are not given to the adversary who passively corrupts this player. For example, one could assume that the signing key is stored in some

intrusion-free cryptographic hardware. Nevertheless, it should be clear that this assumption is quite strong. In fact, because privacy of the inputs is not relevant in the property-based definitions of BA, this assumption essentially takes back any possible additional power that we give to the adversary by introducing passive corruption to the model.

To avoid making such a strong assumption, we assume in the following that the adversary can read all the internal data of passively corrupted players, including their signing keys. Clearly, in that case the adversary can trivially produce signatures of passively corrupted players on any message she wishes.[7] This renders the known protocols insecure when both active and passive corruption are considered simultaneously, as they heavily rely on the unforgeability of signatures generated by non-actively corrupted players. Note that if we model signatures by assuming a signatures functionality, the functionality would also need to allow the adversary to generate valid signatures on behalf of passively corrupted player.

Remark 4.2 (Insecure Channels). The authenticity of the channels is crucial in our model, as it cannot be obtained by insecure channels when the adversary is given access to all the internal data of passively corrupted players. Indeed, if we do not assume authenticated communication, then it is impossible to ensure that a passively corrupted player p broadcasts his correct input. The reason is that there is no way to distinguish between messages sent from p and messages sent from the adversary with p's signature.

In the following we give necessary and sufficient conditions for both Broadcast and Consensus for an adversary who can, simultaneously, actively and passively corrupt players. The bound for Consensus is stated in the following theorem.

Theorem 4.16. *In the model with active and passive corruption, if a setup allowing statistically (resp. computationally) secure signatures is assumed, then a set of players $\mathcal{P}$ can statistically (resp. computationally) $\mathcal{Z}$-securely realize Consensus if and only if the condition $\mathsf{CS}^{(A,E,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ holds, where*

$$\mathsf{CS}^{(A,E,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z}) \iff \begin{cases} \forall (A_1,E_1),(A_2,E_2),(A_3,E_3) \in \mathcal{Z} \ : \\ A_1 \cup A_2 \cup (E_1 \cap E_2 \cap A_3) \neq \mathcal{P} \end{cases}$$

The sufficiency of the condition is proved by constructing a Consensus protocol; using this protocol and the protocol/reduction $\mathsf{Broadcast}^{\mathsf{Cons}}$ one

[7] Producing such a fake signature is not considered a forgery; in the following we shall use the term "forge" only for signatures for which the adversary does not know the corresponding signing key.

can directly obtain a construction of a Broadcast protocol. The necessity is proved for Broadcast in Lemma 4.23.

The Consensus protocol

For constructing the Consensus protocol we follow the approach of Section 4.3.1: from Weak to Graded to King Consensus, which at the end is used to construct the Consensus protocol. The signatures are used in the (sub)protocols to detect whether the adversary has sent inconsistent values. More precisely, we do the following modification to the (sub)protocol from Section 4.3.1: whenever some p_i is instructed to send a message m to some p_j, he sends m and attaches his signature on it; furthermore, before deciding the output of the sub-protocols, the players bilaterally exchange all the signatures that they received during the protocol, and use them to detect inconsistencies introduced by the adversary. For example, if in round r player p_i is instructed to send a message m along with his signature on it, and, later on, some p_j sees p_i's signatures (for round r)[8] on two different messages m and m'', then p_j can safely deduce that p_i is passively or actively corrupted.

In each sub-protocol each player p_j keeps track of a set P_j^E of players p_k such that p_j received inconsistent signatures with signer p_k (i.e., signatures on different messages with the same round ID). Observe, that the set P_j^E is known locally to p_j, but, as the players exchange the received signatures, if some actively corrupted player p sends (and signs) inconsistent values to two correct players, then p will be included in the P_j^E set of every (correct) player p_j. The set P_j^E is used by p_j in the step where he decides his output, where he takes in consideration that $P_j^E \subseteq E^\star$.

In the following, we describe in detail the (sub)protocols WeakConsensus, GradedConsensus, and KingConsensus, and construct the Consensus protocol.

Lemma 4.17. *Assuming that the condition* $\mathsf{CS}_{\text{CONS}}^{(A,E,\emptyset)}(\mathcal{P},\mathcal{Z})$ *holds and no signature is forged, the protocol* WeakConsensus *has the following properties: (weak consistency) There exists some* $y \in \{0,1\}$ *such that every* $p_j \in \mathcal{P} \setminus A^\star$ *outputs* $y_j \in \{y, \text{“}\mathsf{n/v}\text{”}\}$. *(persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *has the same input* x *then they all output* $y = x$.

[8] Recall that we assume that the signature on a message includes enough information to uniquely identify the point of the protocol in which the message was generated, e.g., a round ID and a message ID.

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

1. Each $p_i \in \mathcal{P}$ sends x_i along with his signature on it to every p_j; p_j denotes the set of players who sent him 0 (resp. 1) with their valid signature on it as $P_j^{(0)}$ (resp. $P_j^{(1)}$).
2. Each $p_i \in \mathcal{P}$ forwards all the received values and the corresponding signatures to every $p_j \in \mathcal{P}$; p_j denotes by P_j^E the set of players p_ℓ such that p_j received p_ℓ's signature on two different messages (with the same round ID and message ID).
3. Each p_j sets
$$y_j := \begin{cases} 0 & \text{if } \exists (A,E) \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(0)} \subseteq A) \wedge (P_j^E \subseteq E), \text{ else} \\ 1 & \text{if } \exists (A,E) \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(1)} \subseteq A) \wedge (P_j^E \subseteq E), \text{ else} \\ \text{“n/v”} & \end{cases}$$

Proof. (weak consistency) Assume, towards contradiction, that two non-actively corrupted players p_i and p_j output $y_i = 0$ and $y_j = 1$, respectively. Denote by $P_i^{(\perp)}$ the set of players that did not send a well-formed pair to p_i in Step 1, i.e., $P_i^{(\perp)} := \mathcal{P} \setminus (P_i^{(1)} \cup P_i^{(0)})$. The set $P_j^{(\perp)}$ is defined accordingly. As $y_i = 0$: $\exists (A_i, E_i) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus P_i^{(0)} \subseteq A_i) \wedge (P_i^E \subseteq E_i)$. Similarly, as $y_j = 1$: $\exists (A_j, E_j) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus P_j^{(1)} \subseteq A_j) \wedge (P_j^E \subseteq E_j)$. Because p_i and p_j correctly exchange all the signatures that they receive in Step 1, we have $P_i^{(0)} \cap P_j^{(1)} \subseteq P_i^E \cap P_j^E$. The player set can be written as:

$$\mathcal{P} = P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(0)} = \underbrace{P_i^{(1)} \cup P_i^{(\perp)}}_{\subseteq A_i} \cup \underbrace{(P_i^{(0)} \cap (P_j^{(\perp)} \cup P_j^{(0)}))}_{\subseteq A_j} \cup \underbrace{(P_i^{(0)} \cap P_j^{(1)})}_{\subseteq E_i \cap E_j}$$

Moreover, the players in $P_i^{(0)} \cap P_j^{(1)}$ are clearly actively corrupted as they sent different values to p_i and p_j, i.e., $P_i^{(0)} \cap P_j^{(1)} \subseteq A^\star$. Combining the above we get $\mathcal{P} \subseteq A_i \cup A_j \cup ((E_i \cap E_j) \cap A^*)$ which contradicts $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$. (persistency) If all non actively corrupted players have input 0 (the case of pre-agreement on 1 can be handled symmetrically) then for every $p_i : \mathcal{P} \setminus P_i^{(0)} \subseteq A^\star$. Moreover, unless some signature is forged we have $P_i^E \subseteq E^\star$. Hence, p_i outputs $y_i = 0$ (it follows from the weak consistency property that the decision of p_i is unique, i.e., the condition of Step 3 cannot hold for $y_i = 0$ and $y_i = 1$ simultaneously). □

Lemma 4.18. *Assuming that the condition* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* GradedConsensus *has the following properties: (graded persistency) If every $p_i \in \mathcal{P} \setminus A^\star$ has the same input $x_i = x$ then they all output $y = x$*

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

1. The players invoke WeakConsensus$(\mathcal{P}, \mathcal{Z}, \vec{x})$; each p_i denote his output by z_i (note that $z_i \in \{0, 1, \text{“n/v”}\}$).
2. Each $p_i \in \mathcal{P}$ sends z_i along with his signature on it to every p_j; p_j denotes the sets of players who sent him $0, 1$, and “n/v” along with a valid signature on it as $P_j^{(0)}, P_j^{(1)}$, and $P_j^{(\text{“n/v”})}$, respectively.
3. Each $p_i \in \mathcal{P}$ forwards the values received in Step 2 and the corresponding signature to every $p_j \in \mathcal{P}$; p_j denotes by P_j^E the set of players p_ℓ such that p_j received p_ℓ's signature on two different messages (with the same round ID and message ID).
4. Each p_j sets
$$y_j := \begin{cases} 0 \text{ if } \exists (A, E) \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus (P_j^{(0)} \cup P_j^{(\text{“n/v”})}) \subseteq A) \wedge (P_j^E \subseteq E) \\ 1 \text{ otherwise} \end{cases}$$
$$g_j := \begin{cases} 1 \text{ if } \exists (A, E) \in \mathcal{Z} \text{ s.t. } (\mathcal{P} \setminus P_j^{(y_j)} \subseteq A) \wedge (P_j^E \subseteq E) \\ 0 \text{ otherwise} \end{cases}$$

with grade 1. *(graded consistency) If some $p_i \in \mathcal{P} \setminus A^\star$ outputs $(y_i, g_i) = (y, 1)$ then every $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y$ with some $g_j \in \{0, 1\}$.*

Proof. Before proving the properties we observe that for any $p_j \in \mathcal{P} \setminus A^\star$, $y_j = 1$ only when $\exists (A, E) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus (P_j^{(1)} \cup P_j^{(\text{“n/v”})}) \subseteq A) \wedge (P_j^E \subseteq E)$: indeed, as $y_j = 1$, for every $p_i \in \mathcal{P} \setminus A^\star$ it must hold $z_i \in \{1, \text{“n/v”}\}$ (otherwise we would have $(\mathcal{P} \setminus (P_j^{(0)} \cup P_j^{(\text{“n/v”})}) \subseteq A^\star) \wedge (P_j^E \subseteq E^\star)$ and p_j would output 0). Hence, the players who send a value not in $\{1, \text{“n/v”}\}$ are in $A^\star$. Moreover, because no signatures are forged $P_j^E \subseteq E^\star$, which completes the argument. We next prove the two properties from the lemma: (graded consistency) Assume, towards contradiction, that some $p_i \in \mathcal{P} \setminus A^\star$ outputs $y_i = 0$ with grade $g_i = 1$ and some $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = 1$ (the case $y_i = 1$ and $y_j = 0$ can be handled symmetrically using the observation at the beginning of this proof). The sets $P_i^{(\perp)}$ and $P_j^{(\perp)}$ are defined as in the proof of Lemma 4.17. As $(y_i, g_i) = (0, 1)$: $\exists (A_i, E_i) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus P_i^{(0)} \subseteq A_i) \wedge (P_i^E \subseteq E_i)$. Additionally, as $y_j = 1$: $\exists (A_j, E_j) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus P_j^{(1)} \cup P_j^{(\text{“n/v”})} \subseteq A_j) \wedge (P_j^E \subseteq E_j)$. Because p_i and p_j correctly exchange all the signatures that they receive in Step 2, we have $P_i^{(0)} \cap (P_j^{(1)} \cup P_j^{(\text{“n/v”})}) \subseteq P_i^E \cap P_j^E$. The player set can be written as:

$$\begin{aligned} \mathcal{P} &= P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(\text{“n/v”})} \cup P_i^{(0)} \\ &= \underbrace{P_i^{(1)} \cup P_i^{(\perp)} \cup P_i^{(\text{“n/v”})}}_{\subseteq A_i} \cup \underbrace{(P_i^{(0)} \cap (P_j^{(0)} \cup P_j^{(\perp)}))}_{\subseteq A_j} \cup \underbrace{(P_i^{(0)} \cap (P_j^{(1)} \cup P_j^{(\text{“n/v”})}))}_{\subseteq E_i \cap E_j} \end{aligned}$$

Also clearly $P_i^{(0)} \cap (P_j^{(1)} \cup P_j^{(\text{"n/v"})}) \subseteq A^\star$ as those players sent inconsistent values to p_i and p_j. Combining the above we get $\mathcal{P} \subseteq A_i \cup A_j \cup ((E_i \cap E_j) \cap A^*)$ which contradicts $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$. (graded persistency) If all non actively corrupted players have input 0 (the case of pre-agreement on 1 can be handled symmetrically) then, by the persistency property of WeakConsensus, every $p_k \in \mathcal{P} \setminus A^\star$ has $z_k = 0$ hence for every $p_i : \mathcal{P} \setminus P_i^{(0)} \subseteq A^\star$. Moreover, unless some signature is forged we have $P_i^E \subseteq E^\star$. Hence, p_i outputs $(y_i, g_i) = (0, 1)$ (it follows from the graded consistency property that the decision of p_i is unique, i.e., the condition of Step 4 cannot hold for $y_i = 0$ and $y_i = 1$ simultaneously). □

For achieving King Consensus we use the protocol KingConsensus from Section 4.3.1 which calls the above GradedConsensus protocol instead of the old one. The proof of the following lemma is along the lines of the proof of Lemma 4.7:

Lemma 4.19. *Assuming that the condition* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* KingConsensus *with king* p_k *has the following properties: (king consistency) If* $p_k \in \mathcal{P} \setminus A^\star$ *then every* p_j *outputs the same* $y_j = y$*. (persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *has input* $x_i = x$ *then they all output* $y = x$*.*

Similarly, the protocol Consensus from Section 4.3.1 realizes Consensus when it uses the sub-protocols described in the current section.

Lemma 4.20. *Assuming that the condition* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* Consensus *perfectly* $\mathcal{Z}$*-securely realizes Consensus among the players in* $\mathcal{P}$*.*

Because polynomially many signatures are generated/verified in any invocation of Consensus (in total $O(n^3)$ messages are exchanged), the overall probability that a signature is forged is negligible. This leads naturally to the following feasibility result for Consensus.

Corollary 4.21. *Assuming that the condition* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$ *holds and statistically (resp. computationally) secure signatures are used, the protocol* Consensus *statistically (resp. computationally)* $\mathcal{Z}$*-securely realizes Consensus among the players in* $\mathcal{P}$*.*

4.4.3 Necessity of $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$

The impossibility proof is constructed using an approach similar to the one for the perfect case. More precisely, first we construct a proof for a simpler

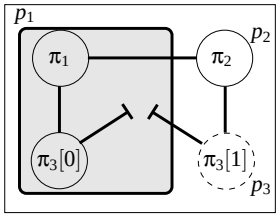

Figure 4.4.1

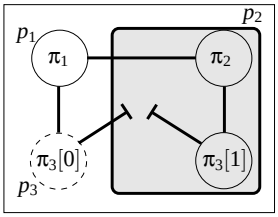

Figure 4.4.2

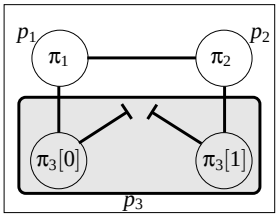

Figure 4.4.3

case, consisting of three players, and then generalize it using a player simulation argument.

Lemma 4.22. *Let $\mathcal{P} = \{p_1, p_2, p_3\}$ and $\mathcal{Z} := \{(\{p_1\}, \{p_3\}), (\{p_2\}, \{p_3\}), (\{p_3\}, \emptyset)\}$. Then there is no $\mathcal{Z}$-secure broadcast protocol with sender p_3. The statement holds for all three security levels, even when a setup allowing digital signatures is assumed.*

Proof. Assume, towards contradiction, that such a protocol exists, and denote by π_1, π_2, and π_3 the programs of player p_1, p_2, and p_3, respectively. We consider the following three scenarios which are also depicted in Figures 4.4.1-4.4.3 (observe that all three scenarios are allowed by $\mathcal{Z}$). In all figures, the gray area correspond to the actively corrupted player, and inside this area the strategy of the adversary is depicted. The channels which appear to be cut have the meaning that the adversary will block all message sent through them. Furthermore, the passively corrupted players are denoted by a dashed circle.

Scenario 1 Sender p_3 has input 1. The adversary passively corrupts p_3 and actively corrupts p_1 and has him run the following strategy: the programs π_1 and π_3 are (locally) invoked using the actual signing keys of the players but with the difference that π_3 is run with input 0. Note that the adversary can locally run these programs because the corresponding players are passively or actively corrupted. The adversary connects these programs with each other as shown in Figure 4.4.1. Because the sender p_3 has input 1 and is not actively corrupted, p_2 must output 1 with overwhelming probability (validity).

Scenario 2 Sender p_3 has input 0. The adversary passively corrupts p_3 and actively corrupts p_2 and has him run the following strategy: the programs π_2 and π_3 are (locally) invoked but π_3 is run with input 1 (again this is

a valid strategy as the corresponding players are passively or actively corrupted). The adversary connects the programs with each other as shown in Figure 4.4.2. Because the sender p_3 has input 0 and is not actively corrupted, p_1 must output 0 with overwhelming probability (validity).

Scenario 3 The adversary actively corrupts p_3 and has him run the following strategy: two instances of the program π_3 are locally invoked (simultaneously), one with input 0 and one with input 1. The adversary connects the instances with each other as shown in Figure 4.4.3. Due to the consistency property of Broadcast, with overwhelming probability the uncorrupted players p_1 and p_2 should output the same value.

To complete the proof, observe that in all three figures we have exactly the same configuration of the programs π_1, π_2, and π_3, hence the distribution of the outputs should be identical, which leads to a contradiction. □

Below we generalize the above lemma for the case of a player set with n players.

Lemma 4.23. *If $\exists (A_1, E_1), (A_2, E_2), (A_3, E_3) \in \mathcal{Z}$ s.t. $A_1 \cup A_2 \cup (E_1 \cap E_2 \cap A_3) = \mathcal{P}$ and $E_1 \cap E_2 \cap A_3 \neq \emptyset$ then for any $p \in E_1 \cap E_2 \cap A_3$ there exist no $\mathcal{Z}$-secure Broadcast protocol with sender p. The statement holds for all three security levels, even when a setup allowing digital signatures is assumed.*

Proof (sketch). Assume, towards contradiction, that such a secure Broadcast protocol π exists. Then π can be transformed into a $\hat{\mathcal{Z}}$-secure Broadcast protocol π' for the player set $\hat{\mathcal{P}} = \{\hat{p}_1, \hat{p}_2, \hat{p}_3\}$ with sender $\hat{p}_3$, where $\hat{\mathcal{Z}} := \{(\{\hat{p}_1\}, \{\hat{p}_3\}), (\{\hat{p}_2\}, \{\hat{p}_3\}), (\{\hat{p}_3\}, \emptyset)\}$: The players run π, with $\hat{p}_1$ and $\hat{p}_2$ playing for the players in A_1 and A_2, respectively, and $\hat{p}_3$ playing for the players in $(E_1 \cap E_2 \cap A_3)$, where the value which $\hat{p}_3$ wishes to broadcast is given as input to p. The existence of such a π' contradicts Lemma 4.22. □

Because Broadcast reduces to Consensus via the protocol/reduction $\mathsf{Broadcast}^{\mathsf{Cons}}$, the above impossibility result, combined with Theorem 4.15, implies the following for Consensus.

Corollary 4.24. *If $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,\emptyset)}(\mathcal{P}, \mathcal{Z})$ is violated then there exists no protocol which $\mathcal{Z}$-securely realizes Consensus among the players in $\mathcal{P}$. The statement holds for all three security levels, even when a setup allowing digital signatures is assumed.*

Proof. Assume that $\mathrm{CS}^{(A,E,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ is violated, i.e., there exist $(A_1,E_1),(A_2,E_2),(A_3,E_3)\in\mathcal{Z}$ such that $A_1\cup A_2\cup(E_1\cap E_2\cap A_3)=\mathcal{P}$. If $(E_1\cap E_2\cap A_3)\neq\mathcal{P}$, then Consensus is not possible, as otherwise we would be able to use $\mathsf{Broadcast}^{\mathsf{Cons}}$ to have any $p\in E_1\cap E_2\cap A_3$ broadcast his input, contradicting Lemma 4.23. Otherwise, we have $A_1\cup A_2=\mathcal{P}$, in which case Theorem 4.15 states that there is no $\mathcal{Z}$-secure protocol for Consensus. □

Remark 4.3 (Player-centric Broadcast). Our impossibility result for Broadcast provides a bound for existence of a Broadcast protocol which can be used with *any* player in p being the sender. However, the result does not exclude the possibility that, even when this bound is violated, there are still some players who *can* broadcast their input. In fact, in [HZ10a] it is proven that there exist adversary structures for which the above is true, and tight bounds are proved which characterize *exactly* when some player p can broadcast his input.

4.4.4 Adding fail-corruption

Augmenting the model with fail-corruption is also more complicated than in the perfect-security case. In fact, the additional complexity reflects also to the (unexpectedly complex) exact bound for BA. This extra complexity is generated by the fact that signatures cannot be used as proofs of the aliveness of a player. In particular, when some p_i is both passively and fail-corrupted, the adversary can force p_i to crash in round r, and introduce in the computation signatures with p_i's signing key for some round $r'>r$. When, later-on, the players use the signature to decide on the output, they will not be able to agree (using only the signature) on whether $p_i\in E^\star\cap F^\star$ or $p_i\in A^\star$.

The necessary and sufficient condition for Consensus in the most general model with all three corruptions is stated in the following theorem which is proved in the remaining of the current section.

Theorem 4.25. *If a setup allowing statistically (resp. computationally) secure signatures is assumed, then a set of players $\mathcal{P}$ can statistically (resp. computationally) $\mathcal{Z}$-securely realize Consensus if and only if the condition* $\mathrm{CS}^{(A,E,\emptyset)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z})$ *holds, where*

$$\mathrm{CS}^{(A,E,F)}_{\mathrm{CONS}}(\mathcal{P},\mathcal{Z}) \iff$$

$$\begin{cases} \forall (A_1,E_1,F_1),(A_2,E_2,F_2),(A_3,E_3,F_3)\in\mathcal{Z}: \\ A_1\cup A_2\cup\big(\big((E_1\cap F_1)\cup(E_2\cap F_2)\cup(E_1\cap E_2)\big)\cap A_3\big)\cup(F_1\cap F_2\cap F_3)\neq\mathcal{P} \end{cases}$$

The idea is to extend the protocol from the previous section so that each p_j keeps track of two sets P_j^E and P_j^F with the following properties: as before, P_j^E includes all the players from which p_j has seen inconsistent signatures; the set P_j^F includes players which p_j has (locally) detected to misbehave, e.g., they did not send him an expected message or they sent him a mal-formed message. The sets P_j^E and P_j^F are both taken in consideration when p_j determines the output of the sub-protocols, as it is clear that $P_j^E \subseteq E^\star$ and $P_j^F \subseteq F^\star$.[9]

The above modification is simple, but the properties that the sub-protocols achieve also need to be slightly modified. In particular, the persistency property of the new sub-protocols is the same as in the previous sections; but the consistency guarantee is "weaker" in all (sub)-protocols, as consistency on the output is only guaranteed *conditioned on the event that no fail-corrupted player crashes* during the execution of the protocol. This yields a King Consensus protocol with the following conditional king-consistency property: If the king is honest and *no player crashes during the execution of the protocol* then every player outputs the same value y.

Despite the fact that the above King Consensus protocol achieves less than the King Consensus protocols designed in the previous sections, we can still use it to to construct a Consensus protocol in the following way: we invoke the King Consensus protocol $n = |\mathcal{P}|$ times for each player being the king (in total n^2 invocations).[10] This way we make sure that in at least one invocation with each king, no player crashes. When the honest king's turn comes, consistency on the outputs will be achieved in the invocation where no player crashes, and (as the persistency is unconditional) it will be preserved in all future invocations.

In the following we describe the protocols WeakConsensus, GradedConsensus, and KingConsensus and state the achieved security.

Lemma 4.26. *Assuming that* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* WeakConsensus *has the following properties: (persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *who has not crashed before the beginning of the protocol has the same input* x *then every correct* p_j *outputs* $y_j = x$*. (conditioned weak consistency) Unless some player crashes during the protocol, there exists some* y *such that every correct* $p_j \in \mathcal{P} \setminus A^\star$ *outputs* $y_i \in \{y, \text{“}\mathsf{n/v}\text{”}\}$.

[9] Recall that we have assumed that for each $(A, E, F) \in \mathcal{Z} : \ F \supseteq A$ and $E \supseteq A$.

[10] In fact it would suffice to do logarithmically many invocations per player as shown in [AFM99].

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

0. Each $p_i \in \mathcal{P}$ sets $P_i^F := \emptyset$.
1. Each p_i sends x_i to every p_j, who denotes the received value as $x_i^{(j)}$; p_j denotes the sets of players who sent him $0, 1$, and $\perp$ by $P_j^{(0)}, P_j^{(1)}$, and $P_j^{(\perp)}$, respectively; p_j also sets $P_j^F := P_j^F \cup P_j^{(\perp)}$.
2. Each $p_i \in \mathcal{P}$ sends his signature on x_i to every p_j, who denotes it as $\sigma_i^{(j)}$. p_j includes in P_j^F the set of players that send him invalid signatures.
3. Each p_j forwards every pair $(x_i^{(j)}, \sigma_i^{(j)})$ to every $p_\ell \in \mathcal{P}$; for $b \in \{0,1\}$: p_ℓ denotes by P_ℓ^E the set of players p_i such that $p_i \in P_\ell^{(b)} \cup P_\ell^{(\perp)}$ but p_ℓ received a valid signature with signer p_i on the value $1-b$ which appears to have been generated in Step 2 (according to the round ID).
4. Each p_j sets

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

Proof. (persistency) If every $p_i \in \mathcal{P} \setminus A^\star$ has input 0 (the case of pre-agreement on 1 can be handled symmetrically) then for any correct p_i: $P_i^{(1)} = \mathcal{P} \setminus (P_i^{(0)} \cup P_i^{(\perp)}) \subseteq A^\star$. Moreover, clearly $P_i^E \subseteq E^\star$ (those are either players for whom inconsistent signatures were seen or players who crashed before Step 2 but still somebody came up with their signature), and $P_i^F \subseteq F^\star$. It remains to show that the condition of Step 5 cannot hold also for $x = 1$. Indeed, this would imply that $\exists (A_i, E_i, F_i) \in \mathcal{Z}$ s.t. $(\mathcal{P} \setminus (P_i^{(1)} \cup P_i^{(\perp)}) \subseteq A_i) \wedge (P_i^F \subseteq F_i)$, and therefore $\mathcal{P} = P_i^{(0)} \cup P_i^{(1)} \cup P_i^{(\perp)} \subseteq A_i \cup A^\star \cup (F_i \cap F^\star)$ contradicting $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$. (conditioned weak consistency). Assume that no player crashes. Assume also, towards contradiction, that two non-actively corrupted players p_i and p_j output $y_i = 0$ and $y_j = 1$, respectively. As $y_i = 0$: $\exists (A_i, E_i, F_i) \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus (P_i^{(0)} \cup P_i^{(\perp)}) \subseteq A_i) \wedge (P_i^E \subseteq E_i) \wedge (P_i^{(\perp)} \subseteq F_i)$. As $y_j = 1$: $\exists (A_j, E_j, F_j) \in \mathcal{Z}$ s.t. $\mathcal{P} \setminus (P_j^{(1)} \cup P_j^{(\perp)}) \subseteq A_j) \wedge (P_j^E \subseteq E_j) \wedge (P_j^{(\perp)} \subseteq F_j)$. The player set can be written as:

$$\mathcal{P} = \underbrace{P_i^{(1)}}_{\subseteq A_i} \cup \underbrace{((P_i^{(0)} \cup P_i^{(\perp)}) \cap P_j^{(0)})}_{\subseteq A_j} \cup$$
$$\cup \underbrace{(P_i^{(0)} \cap P_j^{(1)}) \cup (P_i^{(0)} \cap P_j^{(\perp)}) \cup (P_i^{(\perp)} \cap P_j^{(1)})}_{:=B} \cup \underbrace{(P_i^{(\perp)} \cap P_j^{(\perp)})}_{\subseteq F_i \cap F_j \cap F^\star}$$

Clearly, $B \subseteq A^\star$, as the players in B sent inconsistent values to p_i and p_j in Step 1, and we have assumed that no player crashed in the protocol. In the following, we show that for the set B defined above it holds $B \subseteq (E_i \cap E_j) \cup (E_i \cap F_i) \cup (E_j \cap F_j) \cup (F_i \cap F_j)$: Let $p_k \in B$. One of the following cases must hold. (1) $p_k \in \mathcal{P} \setminus (P_i^F \cap P_j^F)$, (2) $p_k \in P_i^F \cap P_j^F$, (3) $p_k \in P_i^F \cap (\mathcal{P} \setminus P_j^F)$, and (4) $p_k \in (\mathcal{P} \setminus P_i^F) \cap P_j^F$. We look at each case separately: In Case 1 both p_i and p_j have received p_k's signature on both 0 and 1 in Step 2, hence $p_k \in P_i^E \cap P_j^E \subseteq E_i \cap E_j$. In Case 2, $p_k \in F_i \cap F_j$. In Case 3, p_j received p_k's signature (in Step 2) on 1 and sent it to p_i in Step 3; because $p_k \in B \subseteq P_i^{(0)} \cup P_i^{(\perp)}$, p_i includes p_k in $P_i^E \subseteq E_i$, hence $p_k \in E_i \cap F_i$. For Case 4, it can be shown (similar to Case 3) that $p_k \in E_j \cap F_j$. From the above analysis it follows that $\mathcal{P} \subseteq A_i \cup A_j \cup (((E_i \cap E_j) \cup (E_i \cap F_i) \cup (E_j \cap F_j)) \cap A^\star) \cup (F_i \cap F_j \cap F^\star)$ which contradicts $\mathsf{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$. □

The sub-protocol GradedConsensus is described in the following (see next page).

Lemma 4.27. *Assuming that* $\mathsf{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* GradedConsensus *has the following properties: (graded persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *who has not crashed at the beginning has the same input* x *then they all output* $y = x$ *with grade* 1*. (conditioned graded consistency) Unless some player crashes during the protocol, if some* $p_i \in \mathcal{P} \setminus A^\star$ *outputs some* $(y_i, g_i) = (y, 1)$ *then every* p_j *outputs* $y_j = y$ *with some* $g_j \in \{0, 1\}$*.*

For King Consensus we can use the protocol KingConsensus from Section 4.4.2, where the above GradedConsensus is invoked instead of the one in that section. It is straight-forward to verify that this construction satisfies the properties of the following lemma.

Lemma 4.28. *Assuming that* $\mathsf{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds and no signature is forged, the protocol* KingConsensus *has the following properties: (persistency) If every* $p_i \in \mathcal{P} \setminus A^\star$ *who has not crashed at the beginning has the same input* x *then they all output* $y = x$*. (conditioned king consistency) Unless some player crashes during the protocol, if the king* p_k *is honest then there exists some* y *such that every* p_j *outputs* $y_j = y$*.*

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

1. The players invoke WeakConsensus$(\mathcal{P}, \mathcal{Z}, \vec{x})$; each p_i denote his output by z_i (note that $z_i \in \{0, 1, \text{“n/v”}\}$).
2. Each $p_i \in \mathcal{P}$ sends z_i to every p_j who denotes the received value as $z_i^{(j)}$; p_j denotes the sets of players who sent him $0, 1, \perp$, and “n/v” by $P_j^{(0)}, P_j^{(1)}, P_j^{(\perp)}$, and $P_j^{(\text{“n/v”})}$, respectively; p_j also sets $P_j^F := P_j^F \cup P_j^{(\perp)}$.
3. Each $p_i \in \mathcal{P}$ sends his signature on z_i to every p_j, who denotes it as $\sigma_i^{(j)}$. p_j includes in P_j^F the set of players that send him invalid signatures.
4. Each p_j forwards every pair $(z_i^{(j)}, \sigma_i^{(j)})$ to every $p_\ell \in \mathcal{P}$; for $b \in \{0, 1, \text{“n/v”}\}$: p_ℓ denotes by P_ℓ^E the set of players p_i such that $p_i \in P_\ell^{(b)} \cup P_\ell^{(\perp)}$ but p_ℓ received a valid signature with signer p_i on a value in $\{0, 1, \text{“n/v”}\} \setminus \{b\}$ which appears to have been generated in Step 2 (according to the round ID).
5. Each p_j sets

$$y_j := \begin{cases} 0 & \text{if } \exists (A, E, F) \in \mathcal{Z} \text{ s.t. } (P_j^{(1)} \subseteq A) \wedge (P_j^E \subseteq E) \wedge (P_j^F \subseteq F) \\ 1 & \text{otherwise} \end{cases}$$

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

The protocol Consensus is the same as the one described in Section 4.4.2 with the difference that Steps 1a-1b are repeated n^2 times, where in the first n iterations the king is p_1, in the next n iterations the king is p_2, and so on. The following lemma states the achieved security. The security follows from the above analysis and the fact that at most polynomially many signatures are generated within Consensus, which means that the forging probability is negligible.

Lemma 4.29. *Assuming that the condition* $\text{CS}_{\text{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *holds and and a statistically (resp. computationally) secure signature scheme is used, the protocol* Consensus *statistically (resp. computationally)* $\mathcal{Z}$*-securely realizes Consensus.*

Necessity of $\text{CS}_{\text{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$

The necessity of $\text{CS}_{\text{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ is proved along the lines of the necessity proof from the previous section. Note that the complexity of the condition $\text{CS}_{\text{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ also induces some additional complexity on the proofs.

Lemma 4.30. *Let $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5\}$ and $\mathcal{Z}$ be the following structure*

	p_1	p_2	p_3	p_4	p_5
Z_1	a		e/f		e
Z_2		a		e/f	e
Z_3			a	a	a

Then there is no $\mathcal{Z}$-secure Broadcast protocol with sender p_3, p_4, or p_5. The statement holds for all three security levels, even when a setup allowing digital signatures is assumed.

Proof. Assume, towards contradiction, that such a protocol exists, and denote by $\pi_1, \pi_2, \pi_3, \pi_4$, and π_5 the program of player p_1, p_2, p_3, p_4, and p_5 respectively. We consider the cases of Broadcast with sender each of the players p_3, p_4, and p_5 separately.

1) We first consider the case of Broadcast with sender p_5. Consider the following 3 scenarios (Scenarios 4.3.1.a–4.3.1.c) which are also depicted in Figures 4.3.1.a–4.3.1.c (observe that all three scenarios are allowed by $\mathcal{Z}$). In all figures, the gray area correspond to the actively corrupted player(s), and inside this area the strategy of the adversary is depicted. The channels which appear to be cut have the meaning that any message sent through them is ignored. Furthermore, the circles which appear to be filled with black correspond to a program which does nothing (imitates a crashed player)

(Scenario 1.a) Sender p_5 has input 1. The adversary passively corrupts p_3 and p_5, fail-corrupts p_3 and forces him to crash before the protocol starts, and actively corrupts p_1 and has him run the following strategy: the programs π_1, π_3, and π_5 are (locally) invoked using the real keys of the players, where π_5 is run with input 0. Note that the adversary can locally run those programs because the corresponding players are passively or actively corrupted. The programs are connected with each other as shown in Figure 4.3.1.a (observe that the adversary can trivially simulate a program which does nothing, corresponding to the filled black circle). Because the sender p_5 has input 1 and is not actively corrupted or fail-corrupted, p_2 should output 1 with overwhelming probability.

(Scenario 1.b) Sender p_5 has input 0. The adversary passively corrupts p_4, and p_5, fail-corrupts p_4 and forces him to crash before the protocol starts, and actively corrupts p_2 and has him run the following strategy: the programs π_2, π_4, and π_5 are (locally) invoked, where π_5 is run with input 1 (again this is a valid strategy as the corresponding players are passively or actively

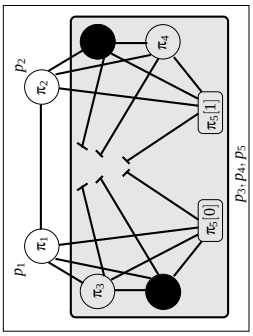

Figure 4.3.1.a

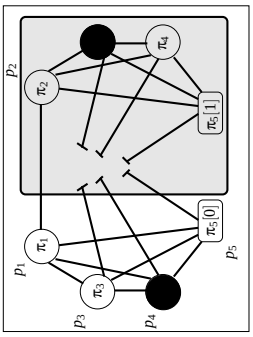

Figure 4.3.1.b

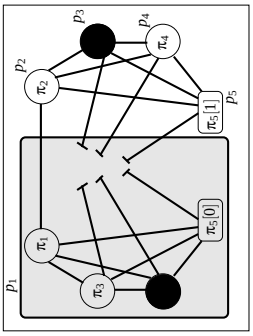

Figure 4.3.1.c

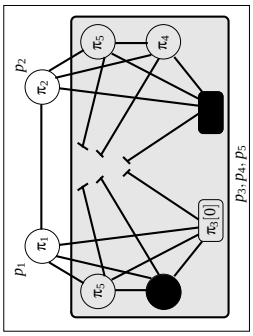

Figure 4.3.2.a

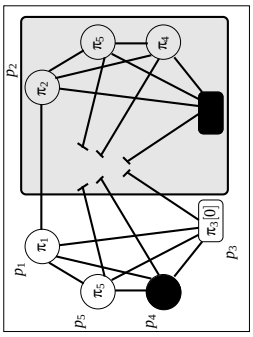

Figure 4.3.2.b

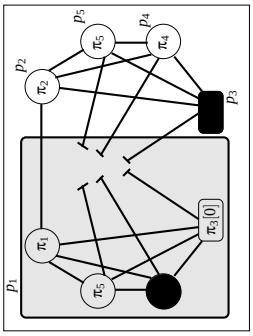

Figure 4.3.2.c

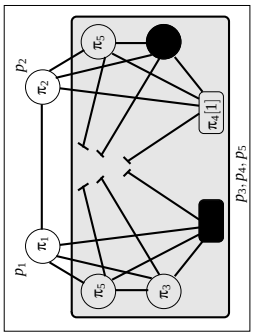

Figure 4.3.3.a

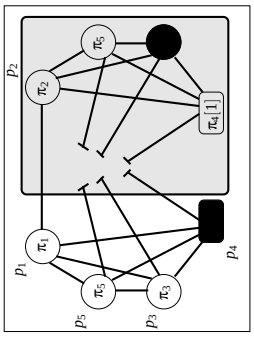

Figure 4.3.3.b

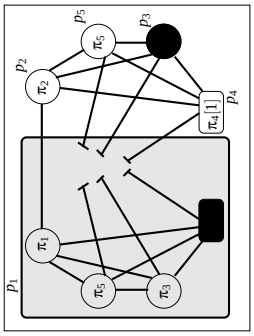

Figure 4.3.3.c

corrupted). The programs are connected with each other as shown in Figure 4.3.1.b. Because the sender p_5 has input 0 and is not actively corrupted or fail-corrupted, p_1 should output 0 with overwhelming probability.

(Scenario 1.c) The adversary actively corrupts p_3, p_4, and p_5 and has them run the following strategy: the programs π_3, π_4 and π_5 are locally invoked, where for π_5 two instances are (simultaneously) run, one with input 0 and one with input 1. The programs are connected with each other as shown in Figure 4.3.1.c. Due to the consistency property of Broadcast, with overwhelming probability the uncorrupted players p_1 and p_2 should output the same value.

However, observe that in all three figures we have exactly the same configuration of the programs $\pi_1, \pi_2, \pi_3, \pi_4$, and π_5, hence the distribution of the outputs should be identical, which leads to a contradiction.

2) We next consider the case of Broadcast with sender p_3. We again consider 3 scenarios (Scenarios 2.1–2.3) which are depicted in Figures 4.3.2.a–4.3.2.c.

(Scenario 2.a) The adversary passively corrupts p_3 and p_5, fail-corrupts p_3 and forces him to crash right before sending his first message, and actively corrupts p_1 and has him run the following strategy: the programs π_1, π_3, and π_5 are (locally) invoked using the real keys of the players, where π_3 is given input 0. The programs are connected with each other as shown in Figure 4.3.2.a (as above the adversary can trivially simulate a program which does nothing, corresponding to the filled black circle). Because the sender p_3 crashes before even sending his first message, p_2 should output $\perp$ with overwhelming probability.

(Scenario 2.b) Sender p_3 has input 0. The adversary passively corrupts p_4 and p_5, fail-corrupts p_4 and forces him to crash before the protocol starts, and actively corrupts p_2 and has him run the following strategy: the programs π_2, π_4, and π_5 are (locally) invoked. The programs are connected with each other as shown in Figure 4.3.2.b. Because the sender p_4 has input 0 and is not actively corrupted or fail-corrupted, p_1 should output 0 with overwhelming probability.

(Scenario 2.c) The adversary actively corrupts p_3, p_4, and p_5 and has them run the following strategy: the programs π_3, π_4, and π_5 are locally invoked, where for π_3 two instances are (simultaneously) run, one with input 0 and one which behaves as having crashed before even sending his first message. The programs are connected with each other as shown in Figure 4.3.2.c. Due to the consistency property of Broadcast, with overwhelming probability the uncorrupted players p_1 and p_2 should output the same value.

However, observe that in all three figures we have exactly the same configuration of the programs $\pi_1, \pi_2, \pi_3, \pi_4$, and π_5, hence the distribution of the outputs should be identical, which leads to a contradiction.

3) The case where the sender is p_4 is argued similarly, where the corresponding scenarios are depicted in Figures 4.3.3.a–4.3.3.c. □

The above lemma can be generalized for the case of a player set $\mathcal{P}$ with n players as follows:

Lemma 4.31. *If* $\exists (A_1, E_1), (A_2, E_2), (A_3, E_3) \in \mathcal{Z}$ *s.t.* $A_1 \cup A_2 \cup ((E_1 \cap F_1) \cup (E_2 \cap F_2) \cup (E_1 \cap E_2)) \cap A_3) = \mathcal{P}$, *and* $((E_1 \cap F_1) \cup (E_2 \cap F_2) \cup (E_1 \cap E_2)) \cap A_3 \neq \emptyset$ *then there for any* $p \in ((E_1 \cap F_1) \cup (E_2 \cap F_2) \cup (E_1 \cap E_2)) \cap A_3$ *there exist no* $\mathcal{Z}$*-secure broadcast protocol with sender* p*. The statement holds for all three security levels independent of whether or not a trusted setup is assumed.*

Proof (sketch). Assume, towards contradiction, that the above condition is violated and there exist a protocol π for p to broadcast his input. Then π can be transformed into a Broadcast protocol π' for the player set $\hat{\mathcal{P}} = \{\hat{p}_1, \hat{p}_2, \hat{p}_3, \hat{p}_4, \hat{p}_5\}$ as follows (wlog assume that the sets A_1, A_2, $(E_1 \cap F_1 \cap A_3)$, $(E_2 \cap F_2 \cap A_3)$, and $(E_1 \cap E_2 \cap A_3)$ are disjoint): $\hat{p}_1$ plays for the players in A_1, $\hat{p}_2$ plays for the players in A_2, $\hat{p}_3$ plays for the players in $(E_1 \cap F_1 \cap A_3)$, $\hat{p}_4$ plays for the players in $(E_2 \cap F_2 \cap A_3)$, and $\hat{p}_5$ plays for the players in $(E_1 \cap E_2 \cap A_3)$. The protocol π' has the following properties: If $p \in (E_1 \cap F_1 \cap A_3)$ then π' allows $\hat{p}_3$ to broadcast his input by giving it as input to the (simulated) sender p. Similarly, if $p \in (E_2 \cap F_2 \cap A_3)$ (resp. if $p \in (E_1 \cap E_2 \cap A_3)$) then π' allows $\hat{p}_4$ (resp. $\hat{p}_5$)to broadcast his input. In all three cases we arrive at a contradiction with Lemma 4.30. □

The impossibility of Consensus when the condition $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ is violated follows directly from the above lemma and Theorem 4.15.

Corollary 4.32. *If* $\mathrm{CS}_{\mathrm{CONS}}^{(A,E,F)}(\mathcal{P}, \mathcal{Z})$ *is violated then there exists no protocol which* $\mathcal{Z}$*-securely realizes Consensus among the players in* $\mathcal{P}$*. The statement holds for all three security level, even when a setup allowing digital signatures is assumed.*

4.5 Composability Considerations

We already mentioned that the property-based definitions of BA do not take into consideration the privacy of the players' inputs. It has been a common

belief that such a privacy consideration is not necessary in the case of BA primitives which are considered to be "inherently public", as the adversary might, by definition, learn the inputs of non-passively corrupted players. For example, in Broadcast with an honest sender, the value which is intended to be broadcast is the sender's input, and therefore is bound to become public. In fact, most, if not all, existing BA protocols are in the authenticated-channels model (no privacy) and use no encryption on the sent messages, which means that the adversary can see the full communication in plaintext.[11] Consistently with the past BA literature, in the current chapter we used the property-based approach. Nevertheless, very recent research [HZ10a] has shown that this might lead in protocols which have problems when used as tools within higher level protocols, e.g., within a general MPC protocol.

The main cause of the problems is the ability of the adversary to take "adaptive" decisions during the execution of the protocol based on her view so far.[12] In the following, we informally discuss a couple of such problems. We point out that many of these problems are not solved and are either open or work in progress at the time this thesis is written. For more details and a formal handling and solutions to problems of similar nature the reader is referred to Chapter 6.

Let us first consider Consensus: when the non-actively corrupted players pre-agree on a value, then this value is bound to become public (as the output of the protocol) independent of the adversary's strategy. However, when there is no pre-agreement, then it is not specified by the definition whether or not the adversary should be allowed to learn those inputs. In fact, in most existing protocols, in case of no pre-agreement an active adversary is allowed to decide the output arbitrarily even depending on the inputs of uncorrupted players. To see how this can result in counter-intuitive behaviors, consider the problem which has been the motivating example for Consensus, namely the Byzantine Generals Problem [LSP82]:"A group of generals of the Byzantine army are camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement." Using one of the suggested Consensus protocols for solving this problem (when the honest generals have no pre-agreement)

[11] As already mentioned, also the protocols described here satisfy the corresponding (property-based) specification, even when the secure channels are replaced by authenticated channels.

[12] Note that even a static adversary can have a strategy, where at any point the future messages of actively corrupted players depend on the values seen so far.

allows the "traitors" to impose an arbitrary battle-plan based on their knowledge about the forces/plans of the "honest" generals.

In the context of Broadcast, similar problems appear, e.g., when we consider active corruption and fail-corruption, simultaneously. More precisely, when the sender p is fail-corrupted and intends to broadcast his input x, then the definition does not exclude that the adversary can decide, depending on x, whether or not p is allowed to broadcast it. In fact, in older definitions of Broadcast with active and fail-corruption [AFM99], it is even allowed that the adversary can choose an arbitrary value which is broadcasted by a fail-corrupted player p who crashes during the protocol.[13]

Problems like the ones mentioned above not only "give a bad feeling" on the property-based definitions of BA, but they actually create problems with simulating invocations of BA protocols within higher level protocols. In other words, the property-based definitions of BA are not composable. This does not mean that existing BA protocols are insecure, but rather that one should be careful when using such protocols (which have only been proved secure according to property-based definition) within a higher level protocol. In Chapter 6 we discuss in depth such a composability/simulatability problem which occurs in the context of Broadcast when active corruption and an adaptive adversary is considered, and propose solutions. Many of the ideas used there can also be used to overcome some of the above problems.

[13]Recall that this is not the case in our definition, as we require that a fail-corrupted sender p broadcasts either his correct/intended input or $\perp$.

Part II

ALTERNATIVE CORRUPTION TYPES

Chapter 5

Omission-Corruption

In the previous chapters we considered two types of observable misbehavior of players, which were modeled by active corruption (arbitrary misbehavior) and fail-corruption (possibility of crashing). These are the types that have been considered in the biggest part of the multi-party cryptographic protocols literature. In this chapter we introduce *omission-corruption* in the model. When a player p is omission-corrupted, the adversary can selectively block messages sent from and to p, but without seeing the actual message (unless p is also passively or actively corrupted). We study omission-corruption in combination with active corruption, and fail-corruption. More concretely, we consider a mixed threshold adversary who can, simultaneously, actively corrupt t_a players, omission-corrupt t_ω players, and fail-corrupt t_f players. We shall refer to such an adversary as (t_a, t_ω, t_f)-*adversary*. We prove exact feasibility bounds for SFE and MPC tolerating such an adversary for all three security levels, i.e., perfect, statistical, and computational. The positive results (sufficiency proofs) are based on the approach of [ZHM09] (which is for perfect security only) and extend this approach to consider also statistical and computational security. The necessity proofs are further extensions of the results from [ZHM09]. As in the previous chapters, we assume a non-adaptive, also known as static, adversary and we work in the synchronous secure-channels model, but we do not assume a Broadcast primitive.

Some extra notation To simplify the description we denote the sets of actively corrupted, omission-corrupted, and fail-corrupted players by $A^\star$, $\Omega^\star$, and $F^\star$, respectively, and the set of uncorrupted players by $\mathcal{H}$ ($\mathcal{H}$ stands for

"honest"). Note that these sets are subsets of the player set $\mathcal{P}$ which are not known to the players and appear only in the security analysis.

5.1 Outline

We start our analysis by formalizing, in Section 5.2, which guarantees one expects from a protocol tolerating omission-corruption. In the subsequent sections, Sections 5.3-5.8, we look at feasibility of multi-party computation with perfect security. Subsequently, in Section 5.9 we sketch how the techniques developed in the previous sections can be used to extend the results to the cases of statistical and computational security.

5.2 The Functionality

Before constructing the protocols which tolerate omission-corruption, it is useful to explore some limitation on the level of security we can hope to achieve for omission-corrupted players. Motivated by the real-world/ideal-world paradigm, we want to guarantee that omission-corrupted players give either their (correct) inputs to the functionality, or give no input. The same should hold also for the outputs, and the adversary should be allowed decide which of the omission-corrupted players are allowed to receive which of their output(s). The strongest security one can hope for is to require that this decision of the adversary is taken independently of the inputs of non actively corrupted players and before seeing the outputs of actively corrupted players. More precisely, one would be interested in securely realizing the strong SFE functionality, denoted as $\mathcal{F}_{\text{SSFE}}$ (see next page). In fact, this functionality, without the fail-corruption, corresponds to the ideal-world evaluation process which was suggested in [Koo06].

Unfortunately, as stated in the following lemma, for any non-trivial choice of t_a and t_ω there exist functions which cannot be securely evaluated with respect to the functionality $\mathcal{F}_{\text{SSFE}}$. Furthermore, the impossibility holds for any security level, i.e., perfect, statistical, and computational. The idea is the following: the adversary might, with good probability, corrupt the player p_j who is the first (or among the first) to get the output, e.g., by randomly choosing whom to corrupt. In this case, she can decide, depending on her outputs, whether or not the omission-corrupted players get full information on the output. However, this behavior cannot be simulated, because in the

Functionality $\mathcal{F}_{\text{SSFE}}$

Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is denoted as $f(\cdot)$. The adversary decides which of the omission-corrupted players give input to or receive output from the functionality before *receiving the outputs of actively corrupted players.*

1. Every $p_i \in \mathcal{H}$ sends his input to $\mathcal{F}_{\text{SSFE}}$. Actively corrupted players might send $\mathcal{F}_{\text{SSFE}}$ arbitrary inputs as instructed by the adversary. For each $p_i \in \Omega^\star$ the adversary decides (without seeing p_i's input) whether p_i sends $\mathcal{F}_{\text{SSFE}}$ his input or a default value (e.g., 0). For each $p_i \in F^\star$ the adversary decides whether p_i is allowed to send his input to $\mathcal{F}_{\text{SSFE}}$ or p_i crashes before sending (in which case $\mathcal{F}_{\text{SSFE}}$ takes a default value for p_i's input). $\mathcal{F}_{\text{SSFE}}$ denotes the received values by $x'_1, \ldots, x'_n$.
2. $\mathcal{F}_{\text{SSFE}}$ computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ (if f is randomized then $\mathcal{F}_{\text{SSFE}}$ internally generates the necessary random coins). $\mathcal{F}_{\text{SSFE}}$ asks the adversary, which of the players $p_i \in \Omega^\star$ should receive their output y_i (without revealing any information about y_i).
3. For each $p_i \in \mathcal{P} \setminus \Omega^\star$, $\mathcal{F}_{\text{SSFE}}$ sends y_i to p_i. For each $p_i \in \Omega^\star$, $\mathcal{F}_{\text{SSFE}}$ sends y_i to p_i if the adversary allowed that p_i receives output in the previous step, otherwise $\mathcal{F}_{\text{SSFE}}$ sends nothing to p_i.

ideal-world the decision of which omission-corrupted players should receive their respective output has to be taken independently of the outputs of the corrupted players.

Lemma 5.1. *If $t_a > 0$ and $t_\omega > 0$, then there exist functions $f(\cdot)$ for which the functionality $\mathcal{F}_{SSFE}$ cannot be perfectly (t_a, t_ω, t_f)-securely realized independent of the choice of t_f. The statement holds also for statistical and computational security.*

Proof. We prove the statement for perfect security. The extension to statistical and computational security is straight-forward.

Consider the identity function, where every player $p_i \in \mathcal{P}$ inputs some value x_i, and the public output is the vector $\vec{x} = (x_1, \ldots, x_n)$. Towards contradiction, assume that there exists a perfectly secure SFE protocol for this function tolerating t_a actively corrupted and t_ω omission-corrupted players, where $t_a, t_\omega > 0$. This protocol implicitly defines for every $p_k \in \mathcal{P}$ a round in which p_k receives full information on the output. Let $\phi(k)$ denote this round. The

adversary has the following strategy: She picks two player p_i and p_j, such that $\phi(i) \leq \phi(j)$, and she actively-corrupts p_i and omission-corrupts p_j.[1] Up to round $\phi(i)$ the adversary instructs the players p_i and p_j to correctly follow their protocol instructions. In round $\phi(i)$ the adversary learns the output $\vec{x}$. Wlog we assume that $i, j \neq 1$. If $x_1 = 1$ then the adversary blocks all incoming communication towards p_j for the rest of the protocol including the messages sent to p_j in round $\phi(i)$. As $\phi(j) \geq \phi(i)$ with some non zero probability, if $x_1 = 1$ then p_j outputs some $\vec{x}' \neq \vec{x}$. However, the (ideal world) simulator does not know x_1 and cannot simulate this behavior. This creates a difference in the output distribution of the real and the ideal world, which contradicts the claimed perfect security of the protocol. □

To make reasonable statements about feasibility of SFE in this model, we relax the definition of the functionality to allow the adversary/simulator to decide which omission-corrupted players receive output, even after having seen the outputs of actively corrupted players (and possibly depending on these outputs). Our relaxation is minimal as Lemma 5.1 suggests. We call the resulting functionality $\mathcal{F}_{\text{SFE}}$ (see next page).

We say that a protocol Π (t_a, t_ω, t_f)-*securely* evaluates the functionality $\mathcal{F}_{\text{SFE}}$ if Π $\mathcal{A}$-securely realizes $\mathcal{F}_{\text{SFE}}$ in the presence of a (t_a, t_ω, t_f)-adversary $\mathcal{A}$.

5.3 The SFE Protocol – A High-Level Description

The main difference between omission-failures and (fail)crash-failures is the fact that *crashing is irreversible*: when a player is forced to crash he remains crashed forever. This irreversibility makes crash-failures easy to detect, as we can use the fact that a crashed player cannot send inconsistent messages.[2] Clearly, this does not hold when the player is omission-corrupted, as the adversary can selectively block messages sent by this player. In this section we describe, on a high-level, how one can deal with this problem, and sketch a construction of an SFE protocol. The protocol is explained in more details in the upcoming sections where a sufficient and necessary bound for its security is proved.

[1] The adversary does not need to know $\phi(\cdot)$, as by randomly selecting two players p_i and p_j, with noticeable probability $\phi(i) \leq \phi(j)$.

[2] In fact, this idea was used in Chapter 4 in the construction of the crash-detection protocol CDP.

Functionality $\mathcal{F}_{\text{SFE}}$

Each $p_i \in \mathcal{P}$ has input x_i. The function to be computed is denoted as $f(\cdot)$. The adversary decides which of the omission-corrupted players receive output from the functionality after *receiving the outputs of actively corrupted players.*

1. Every $p_i \in \mathcal{H}$ sends his input to $\mathcal{F}_{\text{SFE}}$. Actively corrupted players might send $\mathcal{F}_{\text{SFE}}$ arbitrary inputs as instructed by the adversary. For each $p_i \in \Omega^\star$ the adversary decides (without seeing p_i's input) whether p_i sends $\mathcal{F}_{\text{SFE}}$ his input or a default value (e.g., 0). For each $p_i \in F^\star$ the adversary decides whether p_i is allowed to send his input to $\mathcal{F}_{\text{SFE}}$ or p_i crashes before sending (in which case $\mathcal{F}_{\text{SFE}}$ takes a default value for p_i's input). $\mathcal{F}_{\text{SFE}}$ denotes the received values by $x'_1, \ldots, x'_n$.
2. $\mathcal{F}_{\text{SFE}}$ computes $f(x'_1, \ldots, x'_n) = (y_1, \ldots, y_n)$ (if f is randomized then $\mathcal{F}_{\text{SFE}}$ internally generates the necessary random coins). For each $p_i \in \mathcal{P} \setminus \Omega^\star$, $\mathcal{F}_{\text{SFE}}$ sends y_i to p_i.
3. For $p_i \in \Omega^\star$, $\mathcal{F}_{\text{SFE}}$ asks the adversary if p_i should receive his output y_i (without revealing any information about y_i); if the answer is yes then $\mathcal{F}_{\text{SFE}}$ sends y_i to p_i, otherwise it sends nothing to p_i.

An essential part of our construction is concentrated on making omission-corruption *detectable*. More precisely, the goal is to force the adversary to either allow the omission-corrupted players follow their protocol instructions, or make them misbehave in a publicly detectable manner. To this direction, we engineer the initial network to build a network of "detectable" secure channels, where either all sent messages are delivered with perfect privacy, or the protocol aborts, and every player notices it; furthermore, upon abortion all players obtain some (the same) information about the set of corrupted players. The main SFE protocol will be constructed on top of this network, and whenever a transmission aborts, the corresponding obtained corruption-information will be used by the players to recover from the abortion and proceed with the computation.

The road-map towards the construction of *detectable secure channels* is illustrated in Figure 5.1: In a first step we construct a network of *privately detectable authenticated channels*: In this network, whenever some player p_j does not receive an (expected) message from some p_i, p_j can decide whether this happened because he (p_j) is omission-corrupted or because p_i is corrupted

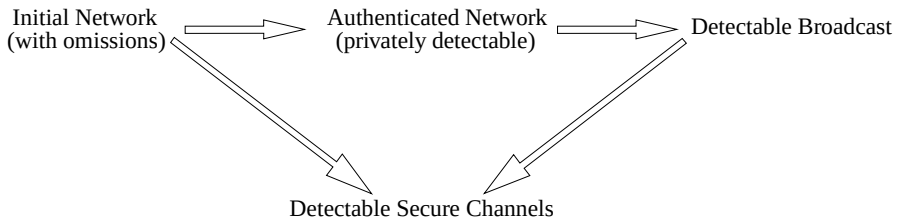

Figure 5.1: Engineering the network

and did not send the message. However, this decision is local to p_j, who cannot prove its validity to any other player. To allow for public detection, we build a *Broadcast* primitive, on top of the detectable authenticated network. The Broadcast primitive is a natural extension of the primitive defined in the previous chapter to include omission-corruption. In particular, it ensures that every non-actively corrupted player broadcasts either his intended input or $\perp$. Furthermore, it guarantees that only an incorrect sender might broadcast $\perp$, and therefore, when the output is $\perp$ then the sender is (publicly) detected as incorrect.[3] The network of (publicly) detectable secure channels is built on top of the initial network and uses the (detectable) Broadcast primitive. The steps of the above-sketched *network-engineering* process are described in detail in Section 5.4.

Having a detectable secure-channels network we construct an SFE protocol in two steps. In a first step, we construct a detectable SFE protocol with properties similar to the detectable secure channels as follows: We invoke one of the known perfectly secure SFE protocols for the secure-channels model, e.g., for perfect security the one from [BGW88], over the detectable secure channels network, and abort whenever some of the network-primitives aborts. Because upon abortion of our network-primitive useful information on the adversary's corruption choice is obtained, the same holds for the detectable SFE protocol. This first step is described in detail in Section 5.5.

In the second step, we use the detectable SFE protocol to achieve robust SFE. The idea for securely computing a given function/circuit C is the following: First, every player shares his input(s) for C, using a passively secure secret-sharing scheme (no consistency checks). Clearly some players might fail sending all the shares (when they are omission-corrupted or they crash), or might deliberately distribute inconsistent shares (when they are actively

[3] Recall that only a corrupted player who has misbehaved might become incorrect.

corrupted). Therefore, we invoke the detectable SFE protocol on input the sharings of the input values to compute a sharing of the output of C on these values, where for any input value which is not uniquely defined from the corresponding input-sharing a default value is taken. As the detectable SFE protocol might abort, this step is repeated until it succeeds (we make sure that it eventually succeeds by using, in an appropriate way, the corruption-information which the players obtain upon each abortion). As soon as it succeeds and a sharing of the output of C is computed, it is reconstructed towards every player. The detailed construction of robust SFE from detectable SFE can be found in Section 5.6.

5.4 Engineering the Network

We show how to construct the network-components described in the previous section. As already mentioned, the goal is to construct detectable secure channels which allow the players to detect when some player does not send a message he is supposed to. Note that when the detected player is omission-corrupted, then he also realizes that his message was not delivered, and therefore realizes that he is omission-corrupted. A player who detects that he is omission-corrupted stops participating in the protocol and sends a special symbol "I'm out" $\notin \mathbb{F}$ in all future rounds.[4] Observe that there is a fundamental difference between such a detected player and a fail-corrupted player who has crashed. More precisely, a detected omission-corrupted player is technically still running, but steps back from the computation to make it easier for the remaining players to complete it. We shall call such detected omission-corrupted players *zombies* or *zombie-players*. Note that such players are not considered alive, i.e., we slightly modify the definition of alive players to exclude zombie-players: we say that a player is *alive* at some point if he has not crashed and has not become a zombie up to that point. Furthermore, observe that a player becomes a zombie only if he has failed to send or receive some messages, which means that he is incorrect.

[4] Equivalently, we could have the detected player p_i send the message "I'm out" only once and then behave as having crashed, and instruct each player who receives such a message to behave as if he would receive "I'm out" from p_i in every future round.

5.4.1 (Privately) Detectable Authenticated Channels

The first primitive in our network-engineering process is the network of privately detectable authenticated channels. In such a channel, whenever a player p_i sends a message x to some player p_j, one of the following three cases might occur: (1) x is delivered, (2) p_j realizes that he is omission-corrupted (and becomes a zombie), and (3) p_j realizes that p_i is incorrect (and outputs $\perp$).

To have p_i send p_j a message x so that the above properties are satisfied, the idea is the following: p_i sends x to every $p_k \in \mathcal{P}$ who forwards it to p_j. If p_k did not receive a message from p_i, then he sends p_j a special message "n/v" $\notin \mathbb{F}$ indicating this fact. If p_j receives less messages then what he should (i.e., more that $t_a + t_\omega + t_f$ messages are missing) he becomes a zombie. The corresponding protocol, denoted as DetAuth, is described in the following (see below).

Protocol DetAuth$(\mathcal{P}, t_a, t_\omega, t_f, p_i, p_j, x)$

1. For each $p_k \in \mathcal{P}$, p_i sends x to p_k. p_k denotes the received value as x_k ($x_k = \perp$ if no value was received).
2. Each $p_k \in \mathcal{P}$ sends p_j a message x'_k, where $x'_k = x_k$ if $x_k \neq \perp$ and $x'_k =$ "n/v" otherwise. p_j denotes the value received from p_k as $x_k^{(j)}$ ($x_k^{(j)} = \perp$ if no value was received).
3. If $|\{x_k^{(j)} : x_k^{(j)} = \perp\}| > t_a + t_\omega + t_f$ then p_j becomes a zombie. Otherwise, if $\exists x' \in \mathbb{F}$ such that $|\{x_k^{(j)} : x_k^{(j)} = x'\}| > t_a$ then p_j output x', otherwise he outputs $\perp$.

Lemma 5.2. *If $3t_a + 2t_\omega + t_f < n$, then protocol* DetAuth *has the following properties: If p_j is alive at the end of the protocol then he outputs a value x', where $x' \in \{x, \perp\}$, unless $p_i \in A^\star$, and $x' = x$ if p_i is correct until the end of the protocol. Moreover, p_j might become a zombie only if he is omission-corrupted.*

The proof of the lemma follows by inspection of the protocol.

5.4.2 Broadcast

Having built the detectable authenticated channels network, we next construct (detectable) Broadcast. The definition of our Broadcast primitive is the

straight-forward adaptation of the Broadcast definition from Chapter 4 to include omission-corruption. In particular, it gives to omission-corrupted players similar guarantees as the Broadcast from Chapter 4 gives to fail-corrupted players (where becoming zombie corresponds to crashing). More precisely, an omission-corrupted sender p never broadcasts a value other than his input, but may broadcast $\perp$. Note that, unless p has become a zombie before completing the protocol, he also sees the output of Broadcast, and therefore if this output is $\perp$ he becomes a zombie as soon as he receives it. Furthermore, a sender who is already a zombie at the beginning of the protocol is treated similarly to a fail-corrupted player who has crashed, i.e., he broadcasts $\perp$. For completeness we describe the Broadcast definition in detail in the following.

We say that a protocol perfectly (t_a, t_ω, t_f)-*securely realizes Broadcast* with sender p (where we denote p's input by x) among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a (t_a, t_ω, t_f)-adversary:

- (consistency) Every player who is alive at the end of the protocol outputs the same value y.
- (validity) $y \in \{x, \perp\}$ unless the sender $p \in A^\star$, where $y = x$ when p is alive and correct at the end of the protocol, and $y = \perp$ if p has become a zombie or has crashed before the invocation of the protocol.
- (termination) For every $p_i \in \mathcal{P} \setminus A^\star$ the protocol terminates after a finite number of rounds.[5]

For the construction of the Broadcast protocol we follow the same approach as in Chapter 4. More precisely, we start by constructing a Consensus primitive and use it to achieve Broadcast. The definition of Consensus can be derived along the lines of the above Broadcast definition. For completeness we include it in the following.

We say that a protocol perfectly (t_a, t_ω, t_f)-*securely realizes Consensus* among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a (t_a, t_ω, t_f)-adversary:

- (consistency) Every $p_i \in \mathcal{P}$ who is alive at the end of the protocol outputs the same value y.
- (persistency) If every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of the protocol has input x then the output is $y = x$.

[5]Similarly to a fail-corrupted player who crashes, for an omission-corrupted player who becomes a zombie, by default, the protocol terminates at that point.

- (termination) For every $p_i \in \mathcal{P} \setminus A^\star$ the protocol terminates after a finite number of rounds.

The definitions for statistical (resp. computational) security for both Consensus and Broadcast are obtained from the above by requiring that the corresponding properties as satisfied except with negligible probability in the presence of an unbounded (resp. efficient) adversary.

Consensus

As in Chapter 4, we construct our Consensus protocol by constructing protocols for realizing weaker Consensus primitives, and then composing them to construct the desired Consensus primitive. The primitives are *Weak Consensus, Graded Consensus*, and *King Consensus*, which are natural adaptations of the corresponding primitives from Chapter 4 to tolerate omission-corruption in addition to active corruption and fail-corruption. In the following, we describe the protocols implementing these primitives and prove their security in corresponding lemmas. The input of each p_i to the protocols is denoted as x_i. Because our goal is to use the designed primitives within an MPC protocol, in the following we assume that the inputs and outputs of the Consensus and Broadcast protocols are values from the finite field $\mathbb{F}$; we point out, however, that the primitives can be used for Consensus and Broadcast on elements of an arbitrary set. The reader is referred to Chapter 4 (Section 4.3.1) for an informal description of the goals of each primitive.

Protocol WeakConsensus $(\mathcal{P}, t_a, t_\omega, t_f, \vec{x} = (x_1, \ldots, x_n))$

1. Each $p_i \in \mathcal{P}$ sends x_i, by invocation of DetAuth, to every $p_j \in \mathcal{P}$, who denotes the received value by $x_j^{(i)}$.
2. Each $p_j \in \mathcal{P}$ outputs y_j, where if $\exists$ unique $x \in \mathbb{F}$ s.t. $\big(|\{p_i : x_j^{(i)} = x\}| \geq n - (t_a + t_\omega + t_f)\big) \wedge \big(|\{p_i : x_j^{(i)} \notin \{x, \perp\}\}| \leq t_a\big)$ then $y_j := x$, otherwise $y_j :=$ "n/v".

Lemma 5.3. *If $3t_a + 2t_\omega + t_f < n$, the protocol* WeakConsensus *has the following properties: (weak consistency) There exists some $y \in \mathbb{F}$ such that every (alive) $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j \in \{y,$ "n/v"$\}$. (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 they all output $y = x$.*

Proof. (weak consistency) Assume that some $p_j \in \mathcal{P} \setminus A^\star$ outputs $y_j = y \in \mathbb{F}$. We show that, in this case, any (alive) $p_k \in \mathcal{P} \setminus A^\star$ outputs $y_k \in \{y, \text{“n/v”}\}$: Indeed, as p_j outputs y he received a value not in $\{y, \perp\}$ from at most t_a players. Among the remaining $\geq n - t_a$ players at most $t_a + t_\omega + t_f$ might be corrupted, hence at least $n - 2t_a - t_\omega - t_f > t_a$ are uncorrupted and sent y to both p_j and p_k (uncorrupted players never send $\perp$); therefore $y_k \in \{y, \text{“n/v”}\}$. (persistency) Every $p_j \in \mathcal{P} \setminus A^\star$ who is alive at the end of the protocol receives the value x from at least all uncorrupted players (i.e., $|\{p_i : x_i^{(j)} = x\}| \geq n - t_a - t_\omega - t_f$) and a value not in $\{x, \perp\}$ only from actively corrupted players and, therefore, x satisfies the condition of Step 2; the weak consistency property ensures that this x is unique, and therefore p_j output $y_j = x$. □

Protocol GradedConsensus $\left(\mathcal{P}, t_a, t_\omega, t_f, \vec{x} = (x_1, \ldots, x_n)\right)$

1. Invoke WeakConsensus $(\mathcal{P}, t_a, t_\omega, t_f, \vec{x}\,)$; p_i denotes his output by x_i'.
2. Each $p_i \in \mathcal{P}$ sends x_i', by invocation of DetAuth, to every $p_j \in \mathcal{P}$, who denotes the received value by $x_j^{(i)}$.
3. Each $p_j \in \mathcal{P}$ outputs (y_j, g_j) as follows:

   y_j: If $\exists$ unique $x \in \mathbb{F}$ s.t. $|\{p_i : x_j^{(i)} = x\}| > t_a$ then $y_j := x$, otherwise $y_j = 0$.

   g_j: If $(|\{p_i : x_j^{(i)} \in \{y_j, \perp\}\}| \geq n - t_a) \wedge (|\{p_i : x_j^{(i)} = y_j\}| \geq n - (t_a + t_\omega + t_f))$ then $g_j := 1$, otherwise $g_j := 0$.

Lemma 5.4. *If $3t_a + 2t_\omega + t_f < n$, protocol* GradedConsensus *has the following properties: (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 $(y_j, g_j) = (x, 1)$. (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 $g_j \in \{0, 1\}$.*

Proof. (graded consistency) Assume that some $p_i \in \mathcal{P} \setminus A^\star$ outputs $(y_i, g_i) = (y, 1)$ for some $y \in \mathbb{F}$. We show that in Step 2 any (alive) $p_j \in \mathcal{P} \setminus A^\star$ receives y more than t_a times, and receives some $y' \in \mathbb{F} \setminus \{y\}$ at most t_a times, hence p_j also outputs $y_j = y$: In deed, as p_i output y with grade 1, he received a value in $\{y, \perp\}$ from $\geq n - t_a$ players, out of which $\geq n - t_a - (t_a + t_\omega + t_f) > t_a$ are uncorrupted and sent y to both p_i and p_j (uncorrupted players never send $\perp$), and therefore we know that, first, every p_j receives y more than t_a times, and, second, because at least one uncorrupted player sent y as his output of WeakConsensus, the weak consistency property guarantees that only actively

corrupted (i.e., $\leq t_a$) players might send a value $y' \in \mathbb{F} \setminus \{y\}$ to p_j. (graded persistency): Assume that every $p_i \in \mathcal{P} \setminus A^\star$ who is alive at the beginning of GradedConsensus has input $x_i = x$. Then by the correctness property of WeakConsensus every non-actively corrupted p_j outputs $x'_j = x$ in Step 1, hence, in Step 2, each p_j gets a value in $\{x, \perp\}$ at least $n - t_a$ times and gets the value x at least $n - t_a - t_\omega - t_f$ times (i.e., from at least all uncorrupted players) and therefore outputs x with grade 1 (the uniqueness of x follows from the graded consistency property). □

Protocol KingConsensus $(\mathcal{P}, t_a, t_\omega, t_f, p_k, \vec{x} = (x_1, \ldots, x_n))$

1. Invoke GradedConsensus$(\mathcal{P}, t_a, t_\omega, t_f, \vec{x})$; p_i denotes his output by (x'_i, g_i).
2. The king p_k sends x'_k to every $p_j \in \mathcal{P}$ by invocation of DetAuth.
3. Each $p_j \in \mathcal{P}$ sets
$$y_j := \begin{cases} x'_j & \text{if } (g_j = 1) \text{ or } (p_k \text{ sent } x'_k \notin \mathbb{F}) \\ x'_k & \text{otherwise} \end{cases}$$

Lemma 5.5. *If $3t_a + 2t_\omega + t_f < n$, the protocol* KingConsensus *has the following properties: (king consistency) If the king p_k is uncorrupted, 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 proof of the above lemma is similar to the proof of Lemma 4.7 from Chapter 4 and is therefore omitted. A Consensus protocol is constructed from KingConsensus in the straight-forward way as in Chapter 4.

Protocol Consensus $(\mathcal{P}, t_a, t_\omega, t_f, \vec{x} = (x_1, \ldots, x_n))$

0. Each $p_i \in \mathcal{P}$ sets $x'_i := x_i$.
1. For $k = 1, \ldots, t_a + t_\omega + t_f + 1$ the following steps are run:
   (a) Invoke KingConsensus$(\mathcal{P}, t_a, t_\omega, t_f, p_k, (x'_1, \ldots, x'_n))$; each p_j denotes his output by $z_j^{(k)}$.
   (b) Each $p_i \in \mathcal{P}$ sets $x'_i := z_i^{(k)}$.
2. Each $p_j \in \mathcal{P}$ outputs $y_j := x'_j$.

Lemma 5.6. *If $3t_a + 2t_\omega + t_f < n$, the protocol* Consensus *perfectly (t_a, t_ω, t_f)-securely realizes Consensus.*

Broadcast

As in Chapter 4, the standard reduction of Broadcast to Consensus, i.e., have the sender p send his input to every player and then invoke Consensus on the received values, does not work. Indeed, such an approach provides no guarantees when an omission-corrupted sender p fails to send his input to some uncorrupted players, or when a fail-corrupted sender crashes during the execution of the protocol. In Chapter 4, where no omission-corruption was considered, we solved this problem by appending at the end of the above construction a crash-detection procedure which checks whether the sender has crashed before the invocation of Consensus. However, such a crash detection is not sufficient for detecting omission-corruption, as the adversary can always allow an omission-corrupted sender to correctly executes all his instructions *during the detection*. Therefore, to allow for detection of omission-corrupted players the messages that are relevant for the computation need to be used.

To guarantee that an omission-corrupted or fail-corrupted sender p never broadcasts a wrong value (but might broadcast $\perp$ when he is incorrect) we use the following construction of a Broadcast protocol which is along the lines of the Broadcast protocol from [Koo06]: As typically, we start by having p send his input x to everybody, and then invoking Consensus on the received values (let y denote the output). Subsequently, instead of appending a crash-detection procedure we do the following: p sends a confirmation bit (instead of an "aliveness" bit) to every player, i.e., a bit b where $b = 1$ if p agrees with the output of Consensus and $b = 0$ otherwise;[6] subsequently, the players invoke Consensus on the received bits to establish a consistent view on the confirmation-bit and they accept the output y only if this bit equals 1, otherwise they output $\perp$. The trick is that the players interpret receiving no bit or an "I'm out"-message instead of a bit for b as 0, i.e., as if the sender would have rejected. This way, we ensure that when the sender is not actively corrupted and the output y of Consensus does not equal his input, then the players will always reject, as independent of whether or not the sender is alive they will all set $b = 0$. This results in protocol Broadcast which is described in details in the following.

Lemma 5.7. *If $3t_a + 2t_\omega + t_f < n$, protocol* Broadcast *perfectly (t_a, t_ω, t_f)-securely realizes Broadcast.*

[6] Observe that the consistency property of Consensus ensures that if p is alive at the end of the protocol then his output is consistent with the output of the other alive players.

Protocol Broadcast $(\mathcal{P}, t_a, t_\omega, t_f, p, x)$

1. Sender p sends x to every $p_j \in \mathcal{P}$ (by DetAuth), who denotes the received value by x_j (p_j sets $x_j := 0$ if a values outside $\mathbb{F}$ was received).
2. The players invoke Consensus on inputs $x_1, \ldots, x_n$; for each $p_j \in \mathcal{P}$ let y_j denote p_j's output.
3. p sends a confirmation bit b to every $p_i \in \mathcal{P}$ (by DetAuth), where $b = 1$ if p's output in Consensus equals x and $b = 0$ otherwise; p_i denotes the received bit by b_i (p_i sets $b_i := 0$ if a bit was not received).
4. Invoke Consensus on inputs $b_1, \ldots, b_n$. For each $p_i \in \mathcal{P}$, if p_i's output in Consensus is 1 then p_i outputs y_i, otherwise he outputs $\perp$.

5.4.3 Detectable Secure Channels

The final step of our network-engineering technique is to construct detectable secure channels, or, stated differently, perfectly secure message transmission with unanimous abort. More precisely, we construct a protocol which allows a sender p_i to send a message to some receiver p_j such that either p_j receives the message, or the protocol aborts. The decision of weather or not the protocol aborts is public; furthermore, upon abortion every (alive) player learns a set $B \subseteq \mathcal{P}$ of players which either consists of one corrupted player, or consists of two players of which at least one is in $A^\star \cup \Omega^\star$. As we show in the following sections, when we invoke a protocol over this network and a message-transmission aborts, then we can use the above corruption-information to allow the protocol recover and go on with its computation.

For simplicity, we describe the protocol DetSMT which allows for detectable secure message transmission, as described above, in the model where a Broadcast primitive is assumed, satisfying all the properties of the definition from Section 5.4.2.

Lemma 5.8. *Assuming that Broadcast is given, the protocol* DetSMT *has the following properties: (correctness) Either it outputs x' towards p_j, where $x' = x$ unless $p_i \in A^\star$, or it aborts with a non-empty set $B \subseteq \{p_i, p_j\}$, such that either $|B| = 2$ and $B \cap (A^\star \cup \Omega^\star) \neq \emptyset$ or $|B| = 1$ and the player in B is incorrect. (privacy) No information about x leaks to the adversary.*

Proof. (correctness) DetSMT aborts with $B = \{p_i\}$ (resp. $B = \{p_j\}$) only when p_i (resp. p_j) broadcast $\perp$ which means that he is incorrect (validity of Broadcast); furthermore DetSMT aborts with $B = \{p_i, p_j\}$ when p_i complains

Protocol DetSMT(p_i, p_j, x)

1. p_j selects a one-time pad key k uniformly at random and sends it to p_i over the bilateral secure channel.
2. p_j broadcast a heart-bit $b = 1$.
3. If p_i received a key $k \in \mathbb{F}$ in Step 1 then he computes $c := x + k$ and broadcast c, otherwise p_i broadcasts a special fail-message "fail". If p_i broadcast "fail" then DetSMT aborts with $B = \{p_i, p_j\}$.
4. p_j outputs $x := c - k$.
5. If at any step p_i (resp. p_j) broadcasts $\perp$, then DetSMT aborts with $B = \{p_i\}$ (resp. $B = \{p_j\}$).

for not receiving a key k, by broadcasting "fail", which might happen only when p_i or p_j is actively corrupted or omission-corrupted. When the protocol does not abort then p_i broadcasts a one-time pad encryption c, where, unless $p_i \in A^\star$, the encrypted message is x and the key is the value k which p_j sent to him in Step 1, in which case $c - k = x$. (privacy) When p_i or p_j is actively corrupted then privacy is no issue. Otherwise, privacy follows directly from the privacy of the bilateral secure channels and the fact that k is uniformly chosen and perfectly blinds the message x. □

5.5 Detectable SFE

The next step towards the final SFE protocol is a detectable SFE protocol. Similarly to the underlying network, the detectable SFE either securely computes any given circuit or it aborts with a non-empty set $B \subseteq \mathcal{P}$ with properties as in Lemma 5.8. Furthermore, when it aborts, the adversary gets no more information than what she can compute from the respective inputs and outputs of actively corrupted players (i.e., from the inputs and outputs she should get if the protocol would not abort).

A protocol with the above properties can be constructed as follows: Let Π_{BGW} denote the MPC protocol from [BGW88] which achieves perfect security and tolerates up to $t < n/3$ actively corrupted players (but no omission-corruption). We run this protocol over the engineered detectable secure-channels network. In particular, every player $p_i \in \mathcal{P}$ executes his instructions of Π_{BGW}, with the difference that whenever he is instructed to broadcast a message he uses the Broadcast primitive, as specified in Section 5.4.2, and

whenever he is instructed to send a message over the bilateral secure-channel to some p_j, the protocol DetSMT is invoked. If p_i broadcasts $\perp$ or DetSMT aborts with B the protocol aborts with B (in the first case we set $B = \{p_i\}$). It follows directly from the perfect security of Π_{BGW} for $t < n/3$ that the above protocol, denoted as DetSFE, has the desired properties when there are at most $t < n/3$ actively corrupted players (independent of the number of omission-corrupted or fail-corrupted players). We state this in the following lemma.

Lemma 5.9. *Assuming that Broadcast is given, if there are at most $t < |\mathcal{P}|/3$ actively corrupted players in $\mathcal{P}$, then protocol $\mathsf{DetSFE}(\mathcal{P}, t, C)$ has the following properties: (correctness) Either it perfectly $(t, \cdot, \cdot)$-securely evaluates the circuit C, or it aborts with a non empty set $B \subseteq \mathcal{P}$. When it aborts with B, then either $|B| = 2$ and $B \cap (A^\star \cup \Omega^\star) \neq \emptyset$ or $|B| = 1$ and the player in B is incorrect. (privacy) The adversary gets no more information than what he can compute from the specified inputs and outputs of actively corrupted players.*

5.6 Robust SFE

We can now describe the robust SFE protocol. As in the previous chapters, the function to be computed is represented by a circuit C. The protocol proceeds in three phases. In the first phase, called the INPUT PHASE, every player shares his input using a passive Shamir-sharing of degree t_a. The second phase, called the EVALUATION PHASE, takes as input the sharings created in the input phase and uses DetSFE to determine the shared values and compute a sharing of the output of C on input these values. The computed sharing is reconstructed in the third phase called the OUTPUT PHASE. Before describing in detail each of the phases we introduce the secret-sharing scheme which is used, and prove some useful results on its reconstructibility.

SECRET-SHARING SCHEME As usual in the threshold-adversary literature, we use Shamir-sharings: A secret s is *t-shared* among the players in $\mathcal{P}$ when there exists a degree-t polynomial $q(\cdot)$ with $q(0) = s$, and each (alive) $p_i \in \mathcal{P} \setminus A^\star$ holds $s_i = q(i)$. The value s_i is p_i's *share* of s. We refer to the vector of shares, denoted by $\langle s \rangle = (s_1, \ldots, s_n)$, as a *$t$-sharing* of s.

Because omission-corrupted players might fail to send/receive some of their respective shares, we introduce the following characterizations for sharings: We say that $\langle s \rangle$ is a *t-consistent* sharing of s among the players in $\mathcal{P}$ if

there exists a degree-t polynomial $q(\cdot)$ with $q(0) = s$ such that each (alive) $p_i \in \mathcal{P} \setminus A^\star$ holds share $s_i \in \{q(i), \perp\}$. Furthermore we say that $\langle s \rangle$ is a t-*robust* sharing of s among the players in $\mathcal{P}$, if $\langle s \rangle$ is t-consistent and for some degree-t polynomial $q(\cdot)$ with $q(0) = s$, each alive $p_i \in \mathcal{H} \cup F^\star$ holds share $s_i = q(i)$.

In the following we describe sufficient conditions for reconstructibility of t-consistent and t-robust sharings. To this direction consider the simple reconstruction protocol Reconstruct described in the following. Reconstruct is given as input a degree-t sharing $\langle s \rangle$ of some value s, as well as the degree t and an upper-bound t' on the number of actively corrupted players in $\mathcal{P}$.

Protocol Reconstruct $(\mathcal{P}, t, t', p, \langle s \rangle)$

1. Each $p_i \in \mathcal{P}$ sends his share s_i to p.
2. p finds, using standard polynomial interpolation techniques, a degree-t polynomial $f(\cdot)$ with the property that more than $t + t'$ of the received shares lie on $f(\cdot)$ and outputs $s' = f(0)$. If no such polynomial exists then p_j outputs $\perp$.

Lemma 5.10. *Assume that there exists t_c such that there are at most t_c corrupted players in $\mathcal{P}$, of whom at most t' are actively corrupted and the condition $t + t' + t_c < |\mathcal{P}|$ holds. Then the protocol* Reconstruct *outputs a value s' towards player p, where $s' \in \{s, \perp\}$ if $\langle s \rangle$ is a t-consistent sharing of s among the players in $\mathcal{P}$, and $s' = s$ if $\langle s \rangle$ is t-robust.*

Proof. The interpolation algorithm in Step 2 outputs $s' = f(0)$ for some degree-t polynomial $f(\cdot)$, if and only if the values sent by more than $t + t'$ players lie on $f(\cdot)$. As non-actively corrupted players never sends wrong values, this implies that the shares of more than t non actively corrupted players lie on $f(\cdot)$. Hence, if $\langle s \rangle$ is a t-consistent sharing of s, i.e., there exists a degree-t polynomial $q(\cdot)$ with $q(0) = s$ such that each non actively corrupted $p_i \in \mathcal{P}$ holds share $s_i \in \{q(i), \perp\}$, then clearly $q(\cdot) = f(\cdot)$. When, in addition, $\langle s \rangle$ is t-robust then all uncorrupted players hold shares that lie on $f(\cdot)$; because $t + t' + t_c < |\mathcal{P}|$ and at most t_c players are corrupted, there are more than $t + t'$ uncorrupted players who correctly send their share and therefore the interpolation algorithm outputs $f(0) = s$. □

We next describe the three phases of the protocol in detail. As in Chapter 3 we assume, without loss of generality (see Page 54), that each player has one input and there is a global output.

INPUT PHASE Each player $p_i \in \mathcal{P}$ t_a-shares his input $s^{(i)}$ by a passive Shamir sharing protocol: p_i chooses a uniformly random degree-t_a polynomial $f_i(\cdot)$ with $f_i(0) = s^{(i)}$ and sends (for $j = 1, \ldots, n$) $f_i(j)$ to p_j. For future reference, we denote this simple sharing protocol as Share. It is easy to verify that Share results in a t_a-consistent sharing of $s^{(i)}$ when $p \in \mathcal{P} \setminus A^\star$, which is even t_a-robust when $p \in \mathcal{P} \setminus (A^\star \cup \Omega^\star)$ and is alive at the end of this phase. Furthermore, because the polynomial is of degree t_a the adversary gets no information about s from the (at most t_a) shares of the actively corrupted players.

EVALUATION PHASE The goal of this phase is to evaluate the circuit C on input the values which were shared in the input phase. For this purpose we use the protocol DetSFE. There are two issues to be resolved. First, the inputs to DetSFE are sharings of the input-values which might not even uniquely define a shared value when the corresponding input-player is corrupted, and, second, DetSFE might abort. To deal with these issues we use the following trick, based on an idea from [IKLP06]: instead of C we evaluate the circuit $\langle C_{\langle\cdot,\cdot\rangle}\rangle$ which implements the following computation: on input the shares distributed in the input phase, $\langle C_{\langle\cdot,\cdot\rangle}\rangle$ first reconstructs them as in protocol Reconstruct (if the reconstruction of some input returns $\perp$ then a default value is taken for this input), and subsequently evaluates C on the reconstructed inputs; denote the output by y. Furthermore, instead of outputting y, $\langle C_{\langle\cdot,\cdot\rangle}\rangle$ computes and outputs a uniformly random t_a-sharing of y (along the line of protocol Share used in the input phase), where the output of each player is his respective share.[7] If DetSFE aborts with set B then the players in B are eliminated from $\mathcal{P}$ and DetSFE is invoked in a smaller setting, i.e., among the players in $\mathcal{P} \setminus B$. In the following we argue that forcing the protocol to abort gives no advantage to the adversary.

Because the output of DetSFE is a uniformly random t_a-sharing, even if the adversary receives her outputs, she gains no information on the inputs of non-actively corrupted players. As the adversary cannot learn more that her outputs, forcing the protocol DetSFE to abort does not give the adversary any advantage. Hence, when the evaluation is repeated due to abortion of DetSFE, the adversary might be able to change her inputs,[8] but has to do this independent of the inputs of actively corrupted players. Furthermore, it is

[7] From the initial circuit C, such a circuit $\langle C_{\langle\cdot,\cdot\rangle}\rangle$ can be computed with size polynomial in the size of C [IKLP06].

[8] In fact the adversary can not only change the inputs of actively corrupted players, but might also be able re-decide which omission-corrupted are allowed to give the input to the computation.

straight-forward to verify that the t_a-robustness and t_a-consistency property of the shares created in the input phase cannot be destroyed by eliminating players.

But there is an additional issue to be resolved. In particular, although the condition $3t_a + 2t_\omega + t_f < n$ ensures that the sufficient conditions for DetSFE and Reconstruct are satisfied in the initial player set $\mathcal{P}$, by eliminating players we destroy the ratio of corrupted vs. uncorrupted players. To resolve this, we use the idea of *player elimination* [HMP00]: Lemma 5.9 ensures that when DetSFE aborts with B then one of the following two cases holds: (1) $|B| = 1$ and the player in B is incorrect, or (2) $|B| = 2$ and $B \cap (A^\star \cup \Omega^\star) \neq \emptyset$. In both cases, eliminating the players in B cannot destroy the ration of corrupted vs. uncorrupted players. More concretely, assume that $t_a^{(\mathcal{P})}, t_\omega^{(\mathcal{P})}$, and $t_f^{(\mathcal{P})}$ are upper bounds on the number of actively corrupted and omission-corrupted players, respectively, in $\mathcal{P}$ which satisfy $3t_a^{(\mathcal{P})} + 2t_\omega^{(\mathcal{P})} + t_f^{(\mathcal{P})} < |\mathcal{P}|$. Then in the updated player set $\mathcal{P}' = \mathcal{P} \setminus B$, there are corresponding thresholds $t_a^{(\mathcal{P}')}$ and $t_\omega^{(\mathcal{P}')}$, and $t_f^{(\mathcal{P}')}$ which satisfy $3t_a^{(\mathcal{P}')} + 2t_\omega^{(\mathcal{P}')} + t_f^{(\mathcal{P}')} < |\mathcal{P}'|$. These thresholds cannot always be computed, but we can compute a "good" upper bound on the number of actively corrupted players, i.e., one that is sufficient for DetSFE and Reconstruct. The idea is to keep track of how many times DetSFE has aborted with one player and how many times with two players in two variables $t^{(1)}$ and $t^{(2)}$, respectively. These variables are used to compute a good upper bound on the number of actively corrupted players by the following simple rule: consider as many of the eliminated players as you can ($\leq t_f$) as fail-corrupted; when you reach the maximum number of fail-corruptions start considering any additional eliminations as omission-corruptions, and, if there are more eliminations, start considering them as active corruptions and reducing the corresponding threshold. This leads to the computation of t'_a as shown in the protocol SFE (see next page). In the corresponding lemma we show that the bound is sufficient for the invoked sub-protocols.

OUTPUT PHASE In the output phase the sharing of the output of C which is generated in the computation phase is reconstructed towards every player by invocation of Reconstruct. Note that as DetSFE has succeeded, every alive player has received his output, i.e., his respective share; hence, the sharing generated by the computation phase is t_a-valid and can be robustly reconstructed by invocation of Reconstruct.

The protocol SFE is described in the following. Each player p_i has input x_i.

Lemma 5.11. *Assuming that Broadcast is given, protocol* SFE *is perfectly* (t_a, t_ω, t_f)*-secure if* $3t_a + 2t_\omega + t_f < |\mathcal{P}|$.

Protocol SFE $(\mathcal{P}, t_a, t_\omega, t_f, C)$

0. Set $\mathcal{P}' := \mathcal{P}$, $t'_a := t_a$, and $t^{(1)} := t^{(2)} := 0$.
1. For each $p_i \in \mathcal{P}$ invoke Share with $t = t_a$ to have p_i t_a-share his input. Each $p_j \in \mathcal{P}$ denotes the vector of all shares he received by $\vec{x}^{\,(j)}$.
2. The players in $\mathcal{P}'$ invoke DetSFE$(\mathcal{P}', t'_a, \langle C_{\langle\cdot\rangle}\rangle)$, where each $p_i \in \mathcal{P}'$ has input $\vec{x}^{\,(i)}$. If DetSFE aborts with B, then set $\mathcal{P}' = \mathcal{P}' \setminus B$, set $t^{(|B|)} := t^{(|B|)} + 1$, set $t'_a := t_a - \max\{(t^{(2)} + \max\{t^{(1)} - t_f, 0\} - t_\omega),\ 0\}$ and repeat this step; otherwise denote by $\langle y\rangle$ the output sharing.
3. For each $p_j \in \mathcal{P}$ invoke Reconstruct$(\mathcal{P}', t_a, t'_a, p_j, \langle y\rangle)$.

Proof. As long as the sufficient conditions for DetSFE and Reconstruct are satisfied, the security of SFE follows directly from the security of the corresponding sub-protocols and the fact that the sharings which are computed are of degree t_a and, therefore, leak no information to the adversary. Because before the output phase the adversary gets no information on the inputs, and as soon as the output phase kicks off the output is guaranteed to be robustly reconstructed, the adversary can perfectly simulate all the sent messages. Termination is guaranteed, because each time the protocol aborts an additional player is eliminated, hence, there can be at most $|\mathcal{P}|$ aborting executions of DetSFE. In the following, we show that the sufficient conditions for DetSFE and Reconstruct are indeed satisfied for the computed threshold t'_a. At any point, denote by $\mathcal{E} := \mathcal{P} \setminus \mathcal{P}'$ the set of eliminated players. Because there are at least $t^{(1)}$ corrupted players in $\mathcal{E}$, at least $\max\{t^{(1)} - t_f, 0\}$ of them have to be in $A^\star \cup \Omega^\star$. In addition to these, there are at least $t^{(2)}$ more players in $\mathcal{E}$ who are in $A^\star \cup \Omega^\star$, hence in total there are at least $t^{(2)} + \max\{t^{(1)} - t_f, 0\}$ players in $\mathcal{E} \cap (A^\star \cup \Omega^\star)$, and therefore at least $\max\{(t^{(2)} + \max\{t^{(1)} - t_f, 0\} - t_\omega), 0\}$ of the players in $\mathcal{E}$ have to be in $A^\star$. Hence, t'_a is an upper bound for the number of actively corrupted players in $\mathcal{P}'$. We show that for $t = t_a, t' = t'_a$, and $t_c = t_a + t_\omega + t_f - t^{(1)} - t^{(2)}$ the condition $t + t' + t_c < |\mathcal{P}'|$ which is sufficient for Reconstruct is satisfied. We consider two cases: (1) $t_\omega \geq t^{(2)} + \max\{t^{(1)} - t_f, 0\}$, and (2) $t_\omega < t^{(2)} + \max\{t^{(1)} - t_f, 0\}$. In Case 1 we have $t'_a = t_a$ hence $t_a + t'_a + t_c = 3t_a + t_\omega + t_f - t^{(1)} - t^{(2)} < n - t_\omega - t^{(1)} - t^{(2)} \leq n - t^{(1)} - 2t^{(2)} = \mathcal{P}'$. In Case 2 we have $t_a + t'_a + t_c = 3t_a + t_\omega + t_f - t^{(1)} - t^{(2)} - (t^{(2)} + \max\{t^{(1)} - t_f, 0\} - t_\omega) = 3t_a + 2t_\omega + t_f - t^{(1)} - 2t^{(2)} - \max\{t^{(1)} - t_f, 0\} \leq 3t_a + 2t_\omega + t_f - t^{(1)} - 2t^{(2)} < n - t^{(1)} - 2t^{(2)} = \mathcal{P}'$. In a similar way we can show that $3t'_a < |\mathcal{P}'|$ which is sufficient for DetSFE: we again consider the same 2 cases. In Case 1, we have $t'_a = t_a$, hence $3t'_a = 3t_a < n - 2t_\omega - t_f \leq n - 2(t^{(2)} + \max\{t^{(1)} - t_f, 0\}) - t_f \leq n - 2t^{(2)} - \max\{t^{(1)} - t_f, 0\} - t_f \leq n - 2t^{(2)} - (t^{(1)} - t_f) - t_f = n - 2t^{(2)} - t^{(1)} = |\mathcal{P}'|$.

In Case 2, we have $3t'_a = 3t_a - 3\max\{(t^{(2)} + \max\{t^{(1)} - t_f, 0\} - t_\omega),\ 0\} \leq 3t_a - 2\max\{(t^{(2)} + \max\{t^{(1)} - t_f, 0\} - t_\omega),\ 0\} \leq 3t_a - 2t^{(2)} - 2\max\{t^{(1)} - t_f, 0\} + 2t_\omega \leq 3t_a + 2t_\omega - 2t^{(2)} - \max\{t^{(1)} - t_f, 0\} \leq 3t_a + 2t_\omega - 2t^{(2)} - t^{(1)} - t_f < n - 2t^{(2)} - t^{(1)}$. □

5.7 Necessity

In this section we show that the condition $3t_a + 2t_\omega + t_f < n$ is necessary for SFE. To this direction, we show that $3t_a + 2t_\omega < n$ is necessary for SFE, and derive the necessity of the initial condition as a trivial corollary.

To prove the necessity of the condition $3t_a + 2t_\omega < n$ for SFE, we show that if this condition is violated then the natural secure message transmission functionality, $\mathcal{F}_{\text{SMT}}$, which takes as input a message $s \in \mathbb{F}$ from a sender p_i and forwards it to a receiver p_j without leaking any information to the adversary, cannot be securely realized. We assume that $\mathcal{F}_{\text{SMT}}$ sends a default message "n/v" to p_j when no input from p_i was received. Also, we assume that when p_j is omission-corrupted and the adversary does not allow him to receive the sent value, then he outputs a special symbol $\perp$, where $\perp \neq$ "n/v".[9] This implies, that when p_j does not receive a valid message $s \in \mathbb{F} \setminus \{\text{"n/v"}, \perp\}$, then he can distinguish between the following two cases: (1) $p_i \in \Omega^\star$ and $p_j \in \mathcal{H}$ (p_j receives "n/v"), and (2) $p_i \in \mathcal{H}$ and $p_j \in \Omega^\star$ (p_j receives $\perp$).

Lemma 5.12. *If $3t_a + 2t_\omega \geq n$ then there are functions which cannot be perfectly (t_a, t_ω, t_f)-securely evaluated for any $t_f \geq 0$.*

Proof (sketch). We prove the lemma for $t_a = 1$ and $t_\omega = 1$, and $n = 5$; using a player-simulation argument (along the lines of the proof of Corollary 4.10) one can easily extend the proof to the desired bound. Let $\mathcal{P} = \{p_1, p_2, p_3, p_4, p_5\}$, and assume, towards contradiction, that there exists a protocol Π allowing p_1 to perfectly securely send a message s to p_2. Consider the following two scenarios:

1. p_1 is omission-corrupted and one of the players p_4 and p_5 is actively corrupted. The adversary permanently blocks all bilateral communication between p_1 and any of the players p_2 and p_3. The actively corrupted player is allowed to follow his protocol instructions. Because p_1 is omission-corrupted and p_2 is honest, p_2 should output either s or "n/v".

[9] Note that both these assumptions are consistent with the behavior of the general SFE functionality $\mathcal{F}_{\text{SMT}}$ described in Section 5.2.

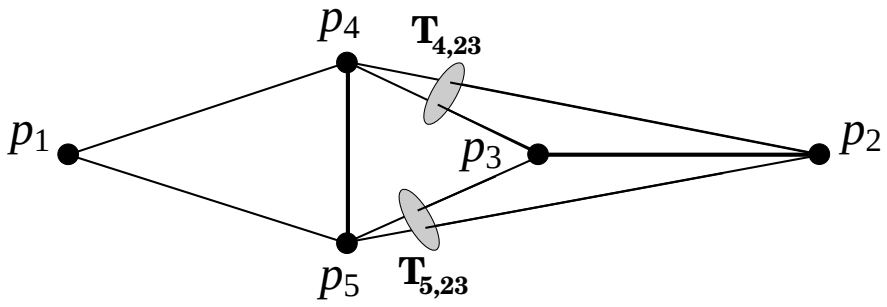

Figure 5.2: The adversarial strategy

2. p_2 is omission-corrupted and p_3 is actively corrupted. Again, the adversary permanently blocks all bilateral communication between p_1 and any of the players p_2 and p_3. In any other way the corrupted players are allowed to follow their protocol instructions. Because p_1 is honest and p_2 is omission-corrupted, p_2 should output either s or $\perp$.

Both adversarial behaviors result in the same situation which is depicted in Figure 5.2, where every player executes exactly his protocol instructions. The bilateral channels that are missing from the configuration in Figure 5.2, i.e., the channels connecting p_1 to each of the players p_2 and p_3, have the meaning that the adversary will block any communication through these channels (it is easy to verify that in both scenarios the adversary can do so). Observe that even if we allow p_2 access to all the messages received from p_3, it is impossible for p_2 to distinguish between these two corruption scenarios. Therefore p_2 cannot output $\perp$ or "n/v", and should output s. Let $T_{4,23}$ (resp. $T_{5,23}$) denote the "conversation", i.e., the full set of messages exchanged, between p_4 (resp. p_5) and the players p_2 and p_3 (see Figure 5.2). Because we want perfect privacy, both $T_{4,23}$ and $T_{5,23}$ should be independent of s. This implies that for each $s' \in \mathbb{F}$ and each $T_{5,23}$ there exists a possible conversation $T'_{4,23}$ which leads p_2 to output s' (otherwise an adversary who corrupts p_5 could try all possible conversations $T'_{4,23}$ and deduce that $s \neq s'$). But if the adversary replaces the program of p_4 with a protocol sending random messages, then with strictly positive (possibly negligible) probability the conversation $T'_{4,23}$ is produced and p_2 outputs s' which violates the assumed perfect security. □

The necessity of $3t_a + 2t_\omega + t_f < n$ follows from the above result as a

simple corollary.

Corollary 5.13. *If $3t_a + 2t_\omega + t_f \geq n$ then there are functions which cannot be perfectly (t_a, t_ω, t_f)-securely evaluated.*

Proof (sketch). Assume, towards contradiction that $3t_a + 2t_\omega + t_f \geq n$ holds and there is a protocol π for secure message transmission. Because the adversary can fail-corrupt the players $p_{n-t_{f+1}}, \ldots, p_n$ and crash them from the beginning, π can be trivially used from the players in $\mathcal{P}' = \{p_1, \ldots, p_{n-t_f}\}$, as an SMT protocol. But, as $3t_a + 2t_\omega < |\mathcal{P}'|$, this contradicts Lemma 5.12. □

5.8 (Reactive) MPC

Having constructed an SFE protocol, a (reactive) MPC protocol can be constructed along the lines of Chapter 3 (Section 3.2): For each gate which is to be evaluated, invoke protocol SFE to compute a t_a-robust uniformly random sharing of the output of the gate among the players in $\mathcal{P}$ (along the lines of protocol Share from the input phase of SFE). We point out that as we only consider threshold corruption, the tight bounds for SFE and MPC are identical, i.e., perfectly (t_a, t_ω, t_f)-secure general MPC is possible if and only if $3t_a + 2t_\omega + t_f < n$. We state this fact in the following lemma.

Lemma 5.14. *Perfectly (t_a, t_ω, t_f)-secure (even reactive) general MPC is possible if and only if $3t_a + 2t_\omega + t_f < n$.*

Remark 5.1. Note that with the above described MPC protocol, we can even tolerate an adversary who can change his choice of omission-corrupted players after the evaluation of each gate with the restriction that during the evaluation of each gate he sticks to the same $\leq t_\omega$ omission-corrupted players. This would require that at the beginning of the evaluation of each gate the zombie-players stop being zombies and return to their initial state, i.e., become alive again. Of course, if a player p is omission-corrupted when a gate receiving input from him (resp. providing output to him) is evaluated, then he might be prevented from providing his input (resp. receiving his output) for this gate.

5.9 Statistical and Computational Security

Our SFE (and MPC) construction is generic and, with small modifications, it can be tuned to achieve statistical or computational security under the

weaker bound $2t_a + 2t_\omega + t_f < n$. In this section we sketch the modifications which are needed for our protocol to achieve this goal, and prove the necessity of this condition. We state our results in the model where Broadcast is assumed which is as secure as the security we wish for our SFE protocol, i.e., for computational (resp. statistical) security we assume computationally (resp. statistically) secure Broadcast. Furthermore, for simplicity we also assume a setup allowing digital signatures with corresponding security. We point out that, because we assume Broadcast, it would not be necessary to assume such a setup, as in the case of computational security we could use the assumed Broadcast to establish a Public-key Infrastructure (by having every player broadcasting his public key), and in the case of statistical security we could use Information Checking [RB89, CDD+99].[10] However, assuming such a setup makes the description much simpler and straight-forward.

There are three modifications that we need on the protocol designed in the previous sections, and have mostly to do with the secret sharing, which is tuned to include digital signatures, in order to ensure robust reconstructibility under the weaker bound $2t_a + 2t_\omega + t_f < n$. In the following we describe in detail these modifications.

The first modification has to do with the sharing created in the input phase: each player p_i who shares his input $s^{(i)}$ according to some polynomial $f_i(\cdot)$, sends to each player p_j, in addition to his share $s_j^{(i)}$, his signature on $s_j^{(i)}$. If some p_j does not receives a share along with p_i's signature then he sets $s_j^{(i)} = \perp$. It is easy to verify that, as before, for each $p_i \in \mathcal{P} \setminus A^\star$ the created input-sharing is a t_a-consistent sharing of $s^{(i)}$, and it is even t_a-robust when $p_i \in \mathcal{P} \setminus A^\star \cup \Omega^\star$ and is alive at the end of the input-sharing phase. Of course, the reconstruction algorithm needs also to be modified accordingly to consider the signatures. In particular, a share is taken into account for the reconstruction only if it is accompanied with a valid signature from the dealer. Furthermore, the reconstruction algorithm does not depend on the number of actively corrupted players; it takes as input the degree t of the sharing, and searches for a degree-t polynomial $f(\cdot)$ such that at least $t+1$ of the accounted shares, i.e., those accompanied with valid signatures from p_i, lie on $f(\cdot)$; if no such polynomial is found the the algorithm outputs $\perp$, otherwise it outputs $f(0)$. It is easy to verify that for the above mentioned secret sharing scheme with dealer p_i, if the condition $t + t_c < |\mathcal{P}|$ holds, where $|A^\star \cup \Omega^\star \cup F^\star| \leq t_c$, then the reconstruction algorithm yields a value $s' \in \mathbb{F} \cup \{\perp\}$, where $s' \in \{s^{(i)}, \perp\}$ when $p_i \in \mathcal{P} \setminus A^\star$, and $s' = s$ when

[10] The Information Checking technique would have to be revised to consider also omission-corruption and fail-corruption.

$p_i \in \mathcal{P} \setminus (A^\star \cup \Omega^\star)$ and has not crashed before sending all the shares to the players.

The second modification has to do with protocol DetSFE and, in particular, with the choice of protocol which is invoked over the detectable secure channels network. Recall that in the perfect-security case, for constructing DetSFE, we used the perfectly secure MPC protocol from [BGW88]. As we are interested in statistical and computational security, it suffices to use a protocol which achieves the corresponding security level. Such protocols are known to exist for both cases: for statistical security we can use the MPC protocol from [RB89] and for computational security the one from [GMW87]. In both cases, the corresponding protocol can tolerate $t < n/2$ players being actively corrupted.

The third modification has to do with the output of the circuit $\langle C_{(\cdot)} \rangle$ which we compute in the evaluation phase. As in the perfect case, the circuit $\langle C_{(\cdot)} \rangle$ should output a degree-t_a uniformly random sharing of the output of the circuit C which we wish to evaluate. However, in order for this sharing to be robustly reconstructible under the weaker bound $2t_a + 2t_\omega + t_f < n$, we need to make sure that actively corrupted players cannot announce false shares. To achieve this, we augment the sharing which $\langle C_{(\cdot)} \rangle$ computes by adding some unforgeable authenticity-tag to each share. More precisely, $\langle C_{(\cdot)} \rangle$ computes a sharing of the outcome, and outputs towards each p_i his share s_i along with a vector containing a signature on s_i from every player.[11] When later on, in the output-generation phase, the output of $\langle C_{(\cdot)} \rangle$ is reconstructed, both the shares and the signature-vectors are announced, and only shares which have a valid signature from every player will be accounted in the reconstruction. This makes sure that, unless the adversary forges a signature of some non-actively corrupted player, no false share will be considered in the reconstruction. Because, in total, at most polynomially many signatures are generated, the forging probability is negligible.

One can verify that with the above modifications, if $2t_a + 2t_\omega + t_f < n$ holds, then the protocol SFE achieves statistical or computational security depending on the security of the setup. In the following we show that this bound is also necessary.

Lemma 5.15. *If $2t_a + 2t_\omega + t_f \geq n$ then there are functions which cannot be (t_a, t_ω, t_f)-securely evaluated. The statement holds for all three security levels.*

[11] For simplicity, we can assume that $\langle C_{(\cdot)} \rangle$ also computes and outputs a new setup for the verification of these signatures where the verification keys are included in the players' respective outputs.

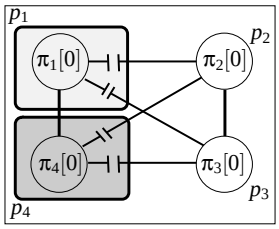

Figure 5.9.1

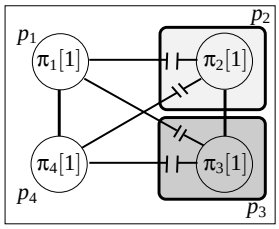

Figure 5.9.2

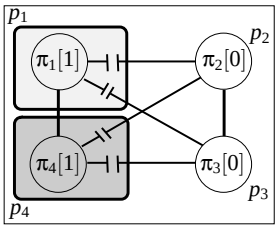

Figure 5.9.3

Proof (sketch). We prove impossibility of securely realizing Consensus for $n = 4$ players out of which $t_a = 1$ can be actively corrupted and, simultaneously, $t_\omega = 1$ can be omission-corrupted. By a simple player-simulation argument we can extend this to an impossibility proof for Consensus when $2t_a + 2t_\omega \geq n$. Extending this further to an impossibility for Consensus when $2t_a + 2t_\omega + t_f \geq n$ is straight-forward, along the lines of the proof of Corollary 5.13.

In the following we concentrate on the four-players case ($n = 4, t_a = 1, t_\omega = 1$): Let $\mathcal{P} := \{p_1, p_2, p_3, p_4\}$. Assume, towards contradiction, that there is a protocol which computationally securely realizes Consensus (the argument for statistical security is the same). Let π_1, π_2, π_3, and π_4 denote the program of p_1, p_2, p_3, and p_4, respectively. We consider the following three scenarios which are also depicted in Figures 5.9.1-5.9.3. In all figures, the dark-gray area corresponds to the actively corrupted player and the light-grey area to the omission-corrupted player; inside each area the strategy of the adversary is depicted. The channels which appear to be cut have the meaning that the adversary will block all message sent through them in any direction.

Scenario 1: p_1, p_2, and p_3 all have input 0; the adversary omission-corrupts p_1 and blocks all his communication with p_2 and p_3, and actively corrupts p_4 and has him run the following strategy: the program π_4 is run with input 0. The adversary connects π_4 with the other programs as shown in Figure 5.9.1. Because all non actively corrupted players have input 0, p_2 must output 0 with overwhelming probability (persistency).

Scenario 2: p_1, p_2, and p_4 all have input 1; the adversary omission-corrupts p_2 and blocks all his communication with p_1 and p_4, and actively corrupts p_3 and has him run the following strategy: the program π_3 is run with input 1. The adversary connects π_3 with the other programs as shown in Figure 5.9.2. Because all non actively corrupted players have input 1, p_1 must output 1

with overwhelming probability (persistency).

Scenario 3: p_1 has input 1, whereas p_2 and p_3 have input 0; the adversary omission-corrupts p_1 and blocks all his communication with p_2 and p_3, and actively corrupts p_4 and has him run the following strategy: the program π_4 is run with input 1. The adversary connects π_4 with the other programs as shown in Figure 5.9.3. In this scenario there is no pre-agreement among the non actively corrupted players; nevertheless the consistency property of Consensus requires that they all output the same value (where p_1 could also output no value as he is omission-corrupted).

Observe that for p_2 Scenario 3 is indistinguishable from Scenario 1, therefore, with overwhelming probability p_2 should output 0. But, for p_1 Scenario 3 is indistinguishable from Scenario 2, therefore, with overwhelming probability p_1 should output 1. Hence, with noticeable probability p_1 and p_2 output inconsistent values, which leads to contradiction. □

Part III

ADAPTIVE CORRUPTION

Chapter 6

Adaptively Secure Broadcast

In the previous chapters we assumed a static, aka non-adaptive adversary. This is a restriction of the adversary's power, which requires her to choose the set of corrupted players at the beginning of the protocol, before seeing any of the exchanged messages. In contrast, an adaptive adversary is allowed to chose the players to corrupt during the execution of the protocol. Until recently, it has been a folklore belief that an adaptive adversary is stronger than a non-adaptive. This belief was formally proved in [CDD+01]. Nevertheless, many of the known MPC protocols are proved secure against a static adversary but are believed (or even implicitly assumed) to be secure even when the adversary is adaptive.

In this chapter we consider the problem of securely realizing Broadcast against an adaptive adversary. We show that the property-based definition of Broadcast does not compose naturally when an adaptive adversary is considered. This is "bad news", as most (if not all) Broadcast protocols in the literature are proved secure according to such a property-based definition. For simplicity we state most of the results in a model where bilateral synchronous secure channels and an (adaptive) adversary who can (only) actively corrupt players is considered. In the last section we briefly discuss other models and additional corruption types. The results in the current chapter are from [HZ10a].

6.1 Outline

In Section 6.2 we discuss the property-based definition of Broadcast and the simulatability problems that come with it. In Section 6.3 we discuss some important aspects of the model. Subsequently, in Section 6.4 the ideal functionality of Broadcast is introduced and compared with what is achievable by most known Broadcast protocols. In Sections 6.5 and 6.6 exact feasibility bounds for adaptively secure Broadcast are proved for all three security levels: (perfect) security without a setup is handled in Section 6.5, whereas statistical and computational security assuming a setup are dealt with jointly in Section 6.6. Finally, in Section 6.7 we briefly discuss how the simulatability issues we deal with occur also in some other of the most commonly used models.

6.2 (In)Composability of the Property-Based Definition

One can think of Broadcast as a megaphone that a sender can use to publicly announce his input and every player listens to. Typically, in the literature Broadcast is defined by enumerating the properties which a Broadcast primitive is expected to satisfy. In Chapter 4 we described such a property-based definition for the case of a mixed adversary who can actively corrupt, passively corrupt, and fail-corrupt players, simultaneously. Most Broadcast protocols in the literature consider only active corruption and satisfy the definition which can be derived from the one in Chapter 4 in the straight-forward way. Because most of the current chapter will be for the active-only case, we describe in the following the definition of Broadcast for an active-only adversary.

We say that a protocol perfectly *t-securely realizes Broadcast* with sender p (denote p's input by x) among the players in $\mathcal{P}$ if it satisfies the following properties in the presence of a t-adversary:

- (consistency) Every $p_i \in \mathcal{P}$ outputs the same value y.
- (validity) If p is uncorrupted then $y = x$.
- (termination) For every honest player the protocol terminates after a finite number of rounds.

The definition for statistical (resp. computational) security is obtained from the above by requiring that the properties are satisfied except with negligible probability in the presence of an unbounded (resp. efficient) adversary.

Clearly, a megaphone-like functionality which takes the input from p and sends it to every player satisfies all the above properties. Moreover, the hope is that the above properties capture in a tight way the behavior of such a functionality, and, in particular, that a protocol proved to satisfy the above properties can be used to realize it. Unfortunately this is not the case, as there are simple settings where the above definition fails to capture the behavior one expects from a Broadcast primitive. Intuitively, the reason is that the property-based definition does take into account the point in time when the sender might get corrupted. In fact, the definition does not exclude that the adversary can corrupt the sender *depending on the message he intends to broadcast*. This can lead to unexpected situations which are inconsistent with the megaphone intuition. We demonstrate two such situations in the following examples.

First, consider the following process for 10 players: Each player p_i ($i = 1, \ldots, 10$) in turn chooses a bit $b_i \in_R \{0,1\}$ uniformly at random and announces it using ideal Broadcast (e.g., using a megaphone), i.e., first p_1 selects $b_1 \in_R \{0,1\}$ and announces it, subsequently p_2 selects $b_2 \in_R \{0,1\}$ and announces it, etc. Consider an adversary who can corrupt at most three players, and her goal is to have only 1's broadcasted. Clearly, the probability that the output sequence consists only of 1's is at most 2^{-7}, as each of the seven bits chosen by the honest players are 1 with probability $1/2$. However, when we replace the ideal Broadcast with some Broadcast protocol satisfying the above three properties, then the adversary might be able to bring this probability to $46 \cdot 2^{-9}$, which is more than ten times bigger. She can achieve this by only corrupting these players p_i who intend to broadcast 0. With the mentioned probability, there are at most three such players, and the adversary can corrupt each of them and make him broadcast 1.

A more cryptography-related example is the following: Consider a prover p who uses the Fiat-Shamir (interactive) protocol to publicly prove to n players (verifiers) that he knows the square root of some publicly known y (in an RSA group). In order to do so, p executes one round of the Fiat-Shamir protocol with each verifier p_i in sequence. All executions are public, in the sense that all the messages are exchanged using ideal Broadcast. Each verifier accepts if all rounds are accepting. Assume that the adversary can corrupt up to $t = n/2$ of the verifiers. Then in this protocol the probability that a malicious prover can make the players accept when he does not know the square

root is negligible in n. However, along the lines of the above example, when the ideal Broadcast is replaced by a Broadcast protocol satisfying (only) the property-based definition, a malicious prover might be able to corrupt only those verifiers who intend to challenge the bit the prover is not prepared for, which allows a malicious prover to cheat with probability $1/2$!

6.3 The Model

Although the model which we consider is the same as the one considered in the previous chapter (with the difference that the adversary is adaptive) and has been discussed in Chapter 2, we choose to discuss some of its aspects in more details here. The reason is that understanding these aspects is the first step for understanding the source of the composability issues that arise.

The players are connected with each-other by a complete network of bilateral secure channels. The communication is synchronous, i.e., there is a known upper bound in the delivery time of any sent message and the players have synchronized clocks. The protocols proceed in rounds, where in each round, every player can send a message to every other player. The messages which are sent in some round are delivered by the beginning of the next round (if some player does not receive an expected message within some round, then he can deduce that the message was not sent).

As already mentioned, we consider an active adaptive t-adversary, i.e., the adversary can actively corrupt players during the execution of the protocol depending on the messages seen so far, with the restriction that the total number of players she corrupts is at most t. Note that we only assume bilateral communication and, in particular, *we do not assume simultaneous multi-send*, i.e., when a player p is instructed to send his messages for some round, then there is no guarantee that all these messages are sent as an atomic multi-send operation. More precisely, it might happen that some of these messages is sent and delivered to its corresponding recipient, say p', before p finishes sending all his messages. In that case, an adversary who has corrupted p' receives the message sent from p for that round, before p has finished sending all his messages for this round, and can still corrupt p and force him change the remaining messages which he intended to send in that round. We emphasize, that this is not a result of some additional assumption on the adversary's power, but rather a result of considering an adaptive adversary in the communication model. In particular, we do not assume that the adversary *can always* perform the above attack, but we simply do not exclude that she *might*

be able to do so. Note that the traditional notion of a *rushing* adversary, as defined in [Can00], implicitly assumes that the adversary *cannot* perform such an attack, or, equivalently, assumes atomic multi-send; but such an assumption is hard to motivate and does not follow from the standard assumptions on the communication network that are required for synchronous computation, i.e., synchronized clocks and channels with upper-bounded delay.

DIGITAL SIGNATURES For the cases of statistical and computational security, we assume digital signatures. For simplicity, we take the approach of assuming a digital signatures functionality $\mathcal{F}_{\text{SIG}}$. Such a functionality was described in [Can03].

6.4 The Broadcast Functionality

The ideal functionality for Broadcast $\mathcal{F}_{\text{BC}}$ when synchronous secure channels are assumed is simple and matches the "megaphone" intuition. Nevertheless, for completeness, in the following we give a description. To keep the intuition simple, we only give a high-level description of the functionality $\mathcal{F}_{\text{BC}}$. However, one can derive a UC-type functionality from our description in a straight-forward manner.

Functionality $\mathcal{F}_{\text{BC}}$

1. p_s sends his input x_s to the functionality $\mathcal{F}_{\text{BC}}$.
2. $\mathcal{F}_{\text{BC}}$ sends x_s to every $p \in \mathcal{P}$.

Despite its simplicity, the above Broadcast functionality is not realized by known Broadcast protocols which are proved secure only according to the property-based definition. In particular, all protocols which start by the sender sending his input to everybody and then invoke a Consensus protocol on the received values, e.g., [CW89, BGP89, BDDS92], are of the above type. But even the protocols where the second phase is not a self-contained Consensus protocol, e.g., the Broadcast protocols from [DS82, PW92], also have the same problem, as they also start by the sender sending his input to everybody and then try to achieve a consistent input on the sent values.

It is not hard to see that a protocol following the above paradigm does not realize $\mathcal{F}_{\text{BC}}$: in any such protocol there is a good probability that a corrupted

player is the first to receive the input x_s from p_s and the adversary can, depending on the received value, decide whether or not to corrupt the sender p_s (and possibly change the broadcasted value). However, this behavior cannot be simulated, as by the time the simulator learns x_s from the functionality it is already too late to change it (the functionality also sends it to all honest players).

In order to come up with a functionality which is realized by known Broadcast protocols, one can consider a relaxation of $\mathcal{F}_{\text{BC}}$ as follows: after receiving the sender's input, the functionality shows it to the adversary (even when p_s is honest) who is then allowed, depending on the received value, to decide whether or not she wants to corrupt p_s and possibly modify the broadcasted value. We denote this weaker functionality as $\mathcal{F}_{\text{WBC}}$ (see below).

Functionality $\mathcal{F}_{\text{WBC}}$

1. p_s sends his input x_s to the functionality $\mathcal{F}_{\text{WBC}}$.
2. $\mathcal{F}_{\text{WBC}}$ sends x_s to the adversary.
3. If p_s is corrupted then the adversary sends a value to $\mathcal{F}_{\text{WBC}}$; $\mathcal{F}_{\text{WBC}}$ denotes the received value by x'_s (if p_s is not corrupted then $\mathcal{F}_{\text{WBC}}$ sets $x'_s := x_s$).
4. $\mathcal{F}_{\text{WBC}}$ sends x'_s to every $p \in \mathcal{P}$

In the following, we show that for each of the three security levels, there exist some protocol in the literature which securely realizes $\mathcal{F}_{\text{WBC}}$. For this purpose we consider two of the known Broadcast protocols from the literature: (1) for (perfect) security without a setup we consider the protocol from [BGP89] which we denote as BC_{BGP}, and (2) for security in the $\mathcal{F}_{\text{SIG}}$-hybrid model we consider the protocol from [DS82] which we denote as BC_{DS}. Both these protocol are deterministic and follow the paradigm described above. Although our arguments are for the above two protocols, one can verify that most Broadcast protocols in the literature securely realize the ideal functionality $\mathcal{F}_{\text{WBC}}$.

Lemma 6.1. *Protocol BC_{BGP} perfectly t-securely realizes the functionality $\mathcal{F}_{\text{WBC}}$ for $t < n/3$.*

Proof. (sketch) As shown in [BGP89], the protocol BC_{BGP} satisfies the property-based definition of Broadcast (i.e., it satisfies validity, consistency, and termination). We show that it perfectly securely realizes $\mathcal{F}_{\text{WBC}}$. Let $\mathcal{A}$ be

an adversary attacking BC_{BGP}; a corresponding simulator $\mathcal{S}$ can be built as follows: First S waits for the input x_s from $\mathcal{F}_{\text{WBC}}$. Note that, because BC_{BGP} is deterministic and the players in $\mathcal{P} \setminus \{p_s\}$ have no input, knowing x_s allows $\mathcal{S}$ to perfectly simulate all the messages sent by honest players.[1] $\mathcal{S}$ invokes $\mathcal{A}$ and does the following:

1. $\mathcal{S}$ simulates all the players in the computation.
2. Whenever $\mathcal{A}$ requests to corrupt some $p_i \in \mathcal{P}$, $\mathcal{S}$ corrupts p_i and sends (the simulated) internal state of p_i to $\mathcal{A}$. From that point on, $\mathcal{S}$ has (the simulated) p_i follow $\mathcal{A}$'s instruction.
3. Whenever $\mathcal{A}$ sends a message to the environment Ψ, $\mathcal{S}$ forwards this message to Ψ.
4. At the end of the simulation, if some (simulated) uncorrupted player p_i outputs $y_i = x_s$, then in the ideal evaluation p_s sends x_s to $\mathcal{F}_{\text{WBC}}$ (even when he is corrupted). Otherwise, i.e., if $y_i \neq x_s$, $\mathcal{S}$ instructs p_s to send y_i to the functionality $\mathcal{F}_{\text{WBC}}$ in Step 3 (the correctness property of BC_{BGP} implies $y_i \neq x_s$ only when p_s is actively corrupted.).

It is easy to verify that $\text{IDEAL}_{\mathcal{F}_{\text{WBC}},\mathcal{S},\Psi} \equiv \text{EXEC}_{\pi,\mathcal{A},\Psi}$, i.e., the protocol perfectly securely realizes $\mathcal{F}_{\text{WBC}}$. □

The proof of the following lemma is along the same lines.

Lemma 6.2. *Protocol BC_{DS} perfectly t-securely realizes $\mathcal{F}_{\text{WBC}}$ for $t < n$ in the $\mathcal{F}_{SIG}$-hybrid model, where the signatures are replaced by calls to an ideal signature functionality $\mathcal{F}_{SIG}$.*

Because many known protocols realize the functionality $\mathcal{F}_{\text{WBC}}$, one might be willing to take a compromise and accept this functionality as a tight description of what one would expect from Broadcast. However, we point out that this functionality allows counter-intuitive behaviors such as the one demonstrated in Section 6.2. Furthermore, the security of protocols which assume Broadcast should be (re-)analyzed with this functionality in mind.

Although the functionality $\mathcal{F}_{\text{WBC}}$ cannot be used "de-facto" to instantiate a Broadcast primitive, such an instantiation is, as we show, possible under certain conditions. More precisely, if a protocol Π (which assumes Broadcast)

[1] In fact, for any adversary $\mathcal{A}$, the simulator $\mathcal{S}$ can generate exactly the same messages as the uncorrupted players would if the protocol would be run with this adversary.

satisfies an appropriate pre-condition, then it is safe to instantiate Broadcast in Π by calls to $\mathcal{F}_{\text{WBC}}$. We stress that this pre-condition is not proved to be necessary; nevertheless, it will allow us to use the known Broadcast protocols (which realizes $\mathcal{F}_{\text{WBC}}$) to achieve secure realizations of $\mathcal{F}_{\text{BC}}$. The pre-condition is the following: For any value v which is supposed to be broadcasted in Π, the adversary *"knows v in advance"*, i.e., there exists a *deterministic* strategy for this adversary to compute v based on the contents of her view *before* the call to the Broadcast primitive. We formalize this in the following lemma which holds for all three security levels.[2]

Lemma 6.3. *Let Π be an $\mathcal{F}_{BC}$-hybrid protocol which securely realizes a given functionality $\mathcal{F}$, and let Π' denote the protocol which results by replacing in Π all the calls to $\mathcal{F}_{BC}$ with calls to $\mathcal{F}_{WBC}$. If Π uses calls to $\mathcal{F}_{BC}$ only to broadcast values which the adversary knows in advance, then Π' securely realizes $\mathcal{F}$.*

Proof. (sketch) Let $\mathcal{A}'$ be an adversary attacking Π' in the $\mathcal{F}_{\text{WBC}}$-hybrid model. We show how to construct an adversary $\mathcal{A}$ attacking Π in the $\mathcal{F}_{\text{BC}}$-hybrid model such that $\text{EXEC}_{\Pi',\mathcal{A}',\Psi} \equiv \text{EXEC}_{\Pi,\mathcal{A},\Psi}$. This is sufficient as then we can use the simulator for $\mathcal{A}$ (which is guaranteed to exist by the security of Π) as a simulator for $\mathcal{A}'$. $\mathcal{A}$ behaves exactly as $\mathcal{A}'$ except in the invocations of $\mathcal{F}_{\text{WBC}}$: when $\mathcal{F}_{\text{WBC}}$ is to be called, in order to simulate the first message of $\mathcal{F}_{\text{WBC}}$ towards $\mathcal{A}'$ (corresponding to the value which the sender intends to broadcast) $\mathcal{A}$ computes the value to be broadcasted (using the deterministic strategy on her view which is guaranteed to exist by the fact that she knows the broadcasted value in advance)[3] and sends this value to $\mathcal{A}'$. $\mathcal{A}'$ is now allowed to corrupt the sender and (possibly) change the value he intends to broadcast. $\mathcal{A}$ acts accordingly and then invokes $\mathcal{F}_{\text{BC}}$. It is straight-forward to verify that $\text{EXEC}_{\Pi',\mathcal{A}',\Psi} \equiv \text{EXEC}_{\Pi,\mathcal{A},\Psi}$. □

Remark 6.1 (Static adversary). One can easily verify that when the adversary is static, i.e., non-adaptive, then we can replace $\mathcal{F}_{\text{BC}}$ by $\mathcal{F}_{\text{WBC}}$ in any protocol, even when the pre-condition of Lemma 6.3 is not satisfied. Indeed, such a static adversary chooses the set of corrupted players at the beginning of the protocol; hence, either the sender p is actively corrupted from the beginning of the execution or he is honest and remains honest until the end of the protocol (all players learn his input). Clearly, in both cases the use of $\mathcal{F}_{\text{WBC}}$ instead of $\mathcal{F}_{\text{BC}}$ gives no advantage to the adversary.

[2] For the case of computational security we will have to require that the strategy of the adversary to compute the value which is to be broadcasted is efficient.

[3] Wlog we can assume that $\mathcal{A}'$ forwards his entire view to $\mathcal{A}$ [Can00, Can01].

6.5 Perfect Security (without setup)

We start the analysis with the easy case, namely perfect security. It has been shown in [LSP82, KY84, FLM86] that a protocol which perfectly t-securely realizes Broadcast according to the property-based definition exists if and only if $t < n/3$ (for a nice proof see also [Fit03]). Because a protocol which t-securely realizes $\mathcal{F}_{\text{BC}}$ has to satisfy the property-based definition of Broadcast, the bound $t < n/3$ is trivially also necessary for the secure realization of $\mathcal{F}_{\text{BC}}$.

Lemma 6.4. *For $t \geq n/3$ there exists no protocol which perfectly t-securely realizes the functionality $\mathcal{F}_{\text{BC}}$. The statement holds even for statistical and computational security, unless some setup is assumed.*

In the remaining of this section we show that the bound $t < n/3$ is also sufficient for Broadcast. Observe that, although protocols which are perfectly secure according to the property-based definition are known to exist, e.g., protocol BC_{BGP}, this does not imply existence of a protocol realizing $\mathcal{F}_{\text{BC}}$.

The idea is the following: for a sender p to broadcast his input x, we have p secret-share x among the players in $\mathcal{P}$ using a perfectly secure Verifiable Secret Sharing (VSS), where, subsequently the sharing of s is reconstructed towards every player. Note that a VSS scheme is a secret-sharing scheme which – in addition to guaranteeing that the adversary learns nothing about the shared value from the shares of actively corrupted players – guarantees that the sharing is robustly reconstructible. For our purposes, we can use the VSS scheme from [BGW88], which is proved to be perfectly secure when at most $t < n/3$ players are corrupted. It is straight-forward to see that this approach yields a secure Broadcast protocol.

It might look as if we are done with the Broadcast protocol, but there is still one issue to be addressed. The [BGW88] VSS (and all other known VSS protocols with perfect security) assumes a Broadcast primitive. However, as we already argued, none of the known Broadcast protocols can "de facto" be used to instantiate such a primitive, as they do not securely realize the Broadcast functionality $\mathcal{F}_{\text{BC}}$. Nevertheless, one can easily verify that the [BGW88] VSS satisfies the pre-conditions of Lemma 6.3. Indeed, in [BGW88] only accusations/disputes and answers to them are broadcasted, which might occur only when the dealer or one of the disputing players is corrupted; hence the adversary knows all the broadcasted values in advance. Using Lemma 6.3 we can instantiate the broadcasting of messages in [BGW88] by calls to $\mathcal{F}_{\text{WBC}}$ without loss of security. Because, according to Lemma 6.1 protocol BC_{BGP}

perfectly t-securely realizes $\mathcal{F}_{\text{WBC}}$ when $t < n/3$, it follows that if we instantiate the broadcasting of messages in the above VSS-based protocol by calls to BC_{BGP}, the resulting protocol, denoted as $\text{VSS}_{\text{BGW}}^{(\text{BGP})}$, perfectly t-securely realizes $\mathcal{F}_{\text{BC}}$ for $t < n/3$. We state this in the following lemma.

Lemma 6.5. *Protocol $VSS_{BGW}^{(BGP)}$ perfectly t-securely realizes the functionality $\mathcal{F}_{BC}$ for $t < n/3$.*

Remark 6.2 (Simultaneous Broadcast). The idea to use VSS to get more out of a Broadcast protocol was used in the context of simultaneous Broadcast [CGMA85, CR87, Gen95, Gen00, HM05, Hev06]. However, the goal of these works is to satisfy an additional property, namely to allow different parties to broadcast values in parallel while guaranteeing mutual independence of the broadcasted values. This does not imply simulation-based composable security. In fact, all these protocols use "normal" Broadcast as sub-primitive, and the security analysis relies on the hope that the protocol realizing this sub-primitive securely composes, which, as we showed, in general is not the case.

6.6 Statistical and Computational Security (with setup)

In this section we look at feasibility of adaptively secure Broadcast with statistical and computational security. It follows from Lemma 6.4 that in order to "beat" the $n/3$ bound one needs to assume a trusted setup. In the current section, we assume a setup allowing generation and verification of digital signatures, which is modeled by assuming a digital signatures functionality $\mathcal{F}_{\text{SIG}}$. In particular, our protocols are described in the $\mathcal{F}_{\text{SIG}}$-hybrid model where generation and verification of signatures is done by calls to $\mathcal{F}_{\text{SIG}}$. Recall that we say that a signature scheme is statistically (resp. computationally) secure is it statistically (resp. computationally) securely realizes the functionality $\mathcal{F}_{\text{SIG}}$.

Unlike the perfect-security case, the exact feasibility bounds for Broadcast are different when an adaptive and a static adversary is considered. In fact, it follows from Lemma 6.2 that when a static adversary and statistically (resp. computationally) secure signatures are assumed, then the protocol BC_{DS} statistically (resp. computationally) t-securely realize $\mathcal{F}_{\text{WBC}}$ and $\mathcal{F}_{\text{BC}}$ (see also Remark 6.1) for any $t < n$. The bound for an adaptive t-adversary is quite different and is given in the following theorem.

Theorem 6.6. *Assuming a statistically (resp. computationally) secure signature scheme, there exist a protocol which perfectly t-securely realizes the functionality $\mathcal{F}_{BC}$ in the presence of an adaptive adversary if and only if $t \leq n/2$.*

In the remaining of this section we prove the above theorem. The sufficiency is proved by constructing a secure Broadcast protocol, whereas the necessity is proved independently in Lemma 6.11 at the end of the section.

Our approach for building a Broadcast protocol follows similar steps as the construction from the previous section. More concretely, first, we construct a protocol which is secure in the $\{\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{SIG}}\}$-hybrid model and satisfies the pre-conditions of Lemma 6.3; this guarantees that our protocol is also secure in the $\{\mathcal{F}_{\text{WBC}}, \mathcal{F}_{\text{SIG}}\}$-hybrid model where all the calls to $\mathcal{F}_{\text{BC}}$ are replaced by calls to $\mathcal{F}_{\text{WBC}}$. Because the Broadcast protocol BC_{DS} perfectly t-securely realizes $\mathcal{F}_{\text{WBC}}$ in the $\mathcal{F}_{\text{SIG}}$-hybrid model (Lemma 6.2), we can replace the calls to $\mathcal{F}_{\text{WBC}}$ in our protocol by invocations of BC_{DS} without loss of security.

We use as starting point of our construction the VSS from [CDD+99]. In [CDD+99] IC-signatures are used to ensure that some p_j who receives a value v from some p_i can, at a later point, publicly prove that p_i indeed send him v.[4] Because IC-signatures are secure only when $t < n/2$ and we construct a Broadcast protocol for $t \leq n/2$, we use digital signatures for the same purpose. As already mentioned the signatures are generated and verified by calls to the functionality $\mathcal{F}_{\text{SIG}}$. We point out that, as in the previous chapters, we assume that each signatures includes enough information to uniquely identify for which message in the flow of the protocol it was generated (e.g., a unique message ID associated with every message sent in the protocol).

In the following, we first describe our sharing, which is along the lines of [CDD+99], and specify some useful security properties, and then we describe and analyze our Broadcast protocol.

Secret Sharing We introduce the following terminology which is consistent with [CDD+99]: we say that a vector $v = (v_1, \ldots, v_m) \in \mathbb{F}^m$ is d-consistent, if there exists a polynomial $p(\cdot)$ of degree d such that $p(i) = v_i$ for $i = 1, \ldots, m$. A value s is said to be *d-shared* among the players in $\mathcal{P}$ when every (honest) player $p_i \in \mathcal{P}$ holds a degree-d polynomial $g_i(\cdot)$ and for each $p_j \in \mathcal{P}$ p_i also holds p_j's signature on $g_i(j)$, where the following condition holds: there exists a degree-d polynomial $q(\cdot)$ with $q(0) = s$ and $g_i(0) = q(i)$ for

[4] We refer to Section 3.5 for a more detailed description of IC-signatures and their goal.

all p_i. The polynomials $g_1(\cdot), \ldots, g_n(\cdot)$ along with the corresponding signatures constitute a *d-sharing* of s, which we denote as $\langle s \rangle$. Note that the vector $(g_1(0), \ldots, g_n(0))$ of the zero-points of the share-polynomials $g_1(\cdot), \ldots, g_n(\cdot)$ is d-consistent, as they all lie on polynomial $q(\cdot)$.

The protocol Share (see next page) is along the lines of the sharing protocol from [CDD+99]. The main difference from a standard sharing protocol is that the correctness of the output-sharing is guaranteed only when the dealer is honest until the end of the protocol.[5] We describe protocol Share (see next page) in the $\{\mathcal{F}_{\text{SIG}}, \mathcal{F}_{\text{BC}}\}$-hybrid model, i.e., Share invokes $\mathcal{F}_{\text{BC}}$ for broadcasting values and $\mathcal{F}_{\text{SIG}}$ for signature generation and verification. To ensure that the output of Share matches the form of our sharing, i.e., every honest p_i outputs a degree-d polynomial $g_i(\cdot)$ and signatures from all other players, we do the following: for every message-transmission, the receiver p_j confirms when he receives a well-formed message from p_i or, otherwise, p_j complains and p_i is expected to answer the complaint by broadcasting the message. If some p_i is publicly caught to misbehave, e.g., by broadcasting a malformed message, then p_i is disqualified. Because corrupted players cannot be forced to sign the messages they send, we make the following convention: when p_i is disqualified, then every player takes a default value, denoted as $\perp$, to be p_i's signature on any message ($\perp$ will always be accepted as valid signature of disqualified players on any message).

Lemma 6.7. *Protocol* Share *invoked in the $\{\mathcal{F}_{BC}, \mathcal{F}_{SIG}\}$-hybrid model achieves the following: The view of any d-adversary attacking the protocol can be perfectly simulated (privacy);*[6] *when the dealer is honest until the end of* Share *then the output is a d-sharing of s (honest-dealer correctness). Furthermore, in all calls to $\mathcal{F}_{BC}$, the adversary "knows in advance" (in the sense of Lemma 6.3) the value to be broadcasted.*

Proof (sketch). Clearly the adversary "knows in advance" all the values to be broadcasted, as these are accusations and replies which might only occur if at least one of the disputing players is corrupted. Furthermore, by inspection of the protocol one can verify that when the dealer is honest until the end of the protocol, then the output is a d-sharing of s, as p_D correctly sends all shares and answers all complaints. Honest-dealer correctness is proved as follows: When the dealer is honest at the end of Share, then only values which lie on the actual polynomial $f(\cdot, \cdot)$ appear in the output of honest players. Moreover, every honest p_i holds all the signatures he should hold, as otherwise p_i would have complained in Step 5 and exposed the inconsistency. Therefore,

[5] Note that such a statement makes sense only when the adversary is adaptive.

[6] This ensures that no information about s leaks to a d-adversary.

Protocol Share$(\mathcal{P}, d, p_D, s)$

1. Dealer p_D chooses a uniformly random bivariate polynomial $f(\cdot,\cdot)$ of degree d in each variable, such that $f(0,0) = s$. For each $p_i \in \mathcal{P}$:
   (a) For $j = 1, \ldots, n$: p_D sends p_i the values $s_{i,j} = f(i,j)$ and $s_{j,i} = f(j,i)$ along with his signature on them; p_i denotes the received values as $s^{(i)}_{i,j}, s^{(i)}_{j,i}$, $\text{sig}_D(s^{(i)}_{i,j})$, and $\text{sig}_D(s^{(i)}_{j,i})$.
   (b) p_i broadcasts a complaint if any of the vectors $\left(s^{(i)}_{i,1}, \ldots, s^{(i)}_{i,n}\right)$ and $\left(s^{(i)}_{1,i}, \ldots, s^{(i)}_{n,i}\right)$ is not d-consistent or if for some value no valid signature was received.
   (c) p_D answers each complaint by broadcasting the values he sent to p_i in Step 1a. If p_D broadcasts a malformed message or invalid signatures then p_D is disqualified; otherwise p_i adopts the broadcasted messages as the messages he should have received in Step 1a.
2. For each $p_i \in \mathcal{P}$:
   (a) For $j = 1, \ldots, n$: p_i sends $s^{(i)}_{i,j}$ to p_j along with his signature $\text{sig}_i(s^{(i)}_{i,j})$ and the dealer's signature $\text{sig}_D(s^{(i)}_{i,j})$.
   (b) Each $p_j \in \mathcal{P}$ broadcasts a complaint if he did not receive a message along with p_i's and p_D's signature on it in Step 2a.
   (c) p_i answers each complaint by broadcasting $(s^{(i)}_{i,j}, \text{sig}_D(s^{(i)}_{i,j}), \text{sig}_i(s^{(i)}_{i,j}))$. If p_i does not broadcast a message or any of the signatures is invalid then p_i is disqualified, every player replaces all p_i's signatures by $\perp$, and p_j adopts $s^{(j)}_{i,j}$ as the value he should have received in Step 2a; otherwise p_j adopts the broadcasted messages as the messages he should have received in Step 2a.
3. Every p_i checks if he received an $s^{(j)}_{i,j}$ from some p_j in Step 2 which is inconsistent with his own view of $s_{i,j}$, i.e., $s^{(j)}_{i,j} \neq s^{(i)}_{i,j}$, and if so, broadcasts $\left(s^{(i)}_{i,j}, s^{(j)}_{i,j}, \text{sig}_D(s^{(i)}_{i,j}), \text{sig}_D(s^{(j)}_{i,j})\right)$; every player verifies that $s^{(j)}_{i,j} \neq s^{(i)}_{i,j}$ and that the signatures are valid and if so p_D is disqualified.[a]

[a] Recall that signatures include information to uniquely identify for which message in the flow of the protocol it was generated, e.g. a unique message ID and session ID.

the output will be a d-sharing of the dealer's value. The privacy of Share can

be argued along the lines of [CDD+99]. For completeness, we sketch how the simulator $\mathcal{S}$ simulates the view of the adversary $\mathcal{A}$: The simulation of the signatures is trivial, as the functionality $\mathcal{F}_{\text{SIG}}$ allows $\mathcal{S}$ to choose the actual signature. As long as $\mathcal{A}$ does not corrupt p_D, the simulator proceeds as follows: whenever $\mathcal{A}$ requests to corrupt some p_i, $\mathcal{S}$ creates a simulated view for p_i (up to the current simulated round) by choosing all the values in p_i's view uniformly at random except those that have already appeared in the view of $\mathcal{A}$ (e.g., if p_j has been already corrupted then the values $s_{i,j}^{(i)} = s_{i,j}^{(j)}$ and $s_{j,i}^{(i)} = s_{j,i}^{(j)}$ have been already given to the adversary). Note that, because the sharing polynomial is of degree d, from the point of view of the d-adversary $\mathcal{A}$ the simulated views are distributed as in a real run of the protocol Share. If at some point $\mathcal{A}$ requests to corrupt p_D, then at that point $\mathcal{S}$ learns p_D's input s and, can simulate the view of all the remaining players while making sure that all simulated values are consistent with some degree-d polynomial $f'(\cdot,\cdot)$ with $f'(0,0) = s$. □

To reconstruct a sharing the protocol Reconstruct is invoked (see below). The idea is the following: every p_i broadcasts his share and the corresponding signatures from the players in $\mathcal{P}$; if some signature is invalid or the announced share is not d-consistent, then p_i is excluded from the reconstruction, otherwise his share-polynomial is interpolated; The zero-coefficients of the share-polynomials of the players that have not been excluded are used to reconstruct the shared value. Depending on the actual choice of d and the number of corrupted parties, the sharing might not uniquely define a value. In any case the players adopt the value which is output by the interpolation algorithm. The consistency of the output is guaranteed as it is decided on publicly seen values.

Protocol Reconstruct$(\mathcal{P}, d, \langle s \rangle)$

1. Each $p_i \in \mathcal{P}$ broadcasts $(s_{i,1}, \ldots, s_{i,n})$ along with the corresponding signatures $\text{sig}_1(s_{i,1}), \ldots, \text{sig}_n(s_{i,n})$; if any of the broadcasted signatures is invalid or if the broadcasted vector is not d-consistent, then p_i is disqualified. Otherwise a polynomial $g_i(\cdot)$ is defined by interpolating the components of the vector.
2. Let $P_{\text{"ok"}} = \{p_{i_1}, \ldots, p_{i_\ell}\}$ denote the set of non-disqualified players. The values $g_{i_1}(0), \ldots, g_{i_\ell}(0)$ are used to interpolate a polynomial $g'(\cdot)$ and every player outputs $g'(0)$.

Lemma 6.8. *Protocol* Reconstruct *invoked in the* $\{\mathcal{F}_{BC}, \mathcal{F}_{SIG}\}$*-hybrid model outputs*

(the same) $y \in \mathbb{F}$ towards every player. Furthermore, if $d < n - t$ (where t is the number of corrupted players) and the input is a d-sharing of some s, then $y = s$.

Proof (sketch). As the output is decided based on values which are agreed upon using $\mathcal{F}_{\text{BC}}$, all players output the same value y. Furthermore, when $d < n - t$ then there are at least $d + 1$ honest players. When additionally the input is a d-sharing then the values which the honest players have signed uniquely define all the share-polynomials $g_i(\cdot)$. Because $\mathcal{F}_{\text{SIG}}$ never verifies as valid a signature on a value which was not signed by the corresponding player, the adversary cannot announce a polynomial other than $g_i(\cdot)$ for any $p_i \in \mathcal{P}$. Hence, every corrupted p_i either announces the correct value or is disqualified. However, the honest players always announce the correct values, hence the correct polynomials are interpolated. Because there are at least $d+1$ honest players there will always be at least $d+1$ values to interpolate the correct $g'(\cdot)$ and recover the shared value. □

We next describe our Broadcast protocol for $t \leq n/2$. The idea is to have the sender p_s share his input x_s by a degree-$(t-1)$ sharing, using Share with $d = t - 1$, and subsequently invoke Reconstruct on the output of Share. The intuition is the following: When p_s is honest until the end of Share, then Share outputs a $(t-1)$-sharing of x_s (Lemma 6.7: honest-dealer correctness); because $t \leq n/2$ implies $d = t-1 < n-t$, Lemma 6.8 guarantees that Reconstruct will output x_s. Hence, the only way the adversary can change the output to some $s' \neq s$ is by corrupting p_s during (or before) protocol Share. As there are at most t corrupted players, if the adversary wishes to corrupt p_s, then she might corrupt at most $t - 1$ of the remaining players; but, because a $(t-1)$-adversary gets no information on x_s (Lemma 6.7: privacy), the decision whether or not to corrupt p_s has to be taken independently of x_s, which is a behavior that can be easily simulated.

In the above, we managed to tweak the VSS protocol from [CDD+99] (which is secure if and only if $t < n/2$) so that we can use it for Broadcast when $t \leq n/2$. However, as already mentioned, both Share and Reconstruct use calls to $\mathcal{F}_{\text{BC}}$ for broadcasting. In order to replace $\mathcal{F}_{\text{BC}}$ by $\mathcal{F}_{\text{WBC}}$, we need to make sure that the precondition of Lemma 6.3 is satisfied (i.e., the adversary "knows in advance" all broadcasted values). For protocol Share this is guaranteed by Lemma 6.7. But the values which are broadcasted in Reconstruct are not necessarily known to the adversary in advance. We resolve this by a technical trick, namely we introduce a *dummy* step between Share and Reconstruct, where every player sends to every other player his output from protocol

Share. Observe that such a modification could potentially give an advantage to the adversary. But this might only happen in case the adversary has not corrupted p_s by the end of Share, as otherwise she knows all the outputs. However, even in this bad case, because p_s is honest until the end of Share, by the time the dummy step is executed the output is already fixed to x_s, and the adversary cannot change it even with access to the full transcript. For completeness we include a description of our Broadcast protocol and state its achieved security.

Protocol Broadcast (p_s, x_s)

1. Invoke Share$(\mathcal{P}, t-1, p_s, x_s)$; if p_s is disqualified then every player outputs a default value, e.g., 0 and halts.
2. Every $p_i \in \mathcal{P}$ sends his output from Share to every $p_j \in \mathcal{P}$.
3. Invoke Reconstruct on the output of Share.

Lemma 6.9. *Protocol* Broadcast *perfectly t-securely realizes the functionality $\mathcal{F}_{BC}$ in the $\{\mathcal{F}_{WBC}, \mathcal{F}_{SIG}\}$-hybrid model, for $t \leq n/2$.*

Proof (sketch). By inspection of the protocol one can verify that the preconditions of Lemma 6.3 are satisfied for every value which is broadcasted, hence using $\mathcal{F}_{\text{WBC}}$ for broadcasting values in protocol Broadcast is as secure as using $\mathcal{F}_{\text{BC}}$. Therefore, it suffices to argue the security of Broadcast in the $\{\mathcal{F}_{\text{BC}}, \mathcal{F}_{\text{SIG}}\}$-hybrid model. We sketch the simulator $\mathcal{S}$ for a given adversary $\mathcal{A}$. During the execution of Share, the simulator behaves as the simulator in the proof of Lemma 6.7. In the subsequent steps, if $\mathcal{A}$ has already corrupted p_s before the end of the simulated run of Share, then at that point $\mathcal{S}$ has learned the sender's input x_s and can simulate the remaining transcript as in the proof of Lemma 6.7 ($\mathcal{S}$ can clearly simulate all the messages exchanged in Steps 2 and 3 as they appear in the transcript of Share). Otherwise, i.e., if by the end of the simulated run of Share the adversary $\mathcal{A}$ has not requested to corrupt p_s, then $\mathcal{S}$ allows for the invocation of $\mathcal{F}_{\text{BC}}$, where p_s gives his input x_s; $\mathcal{S}$ learns x_s from $\mathcal{F}_{\text{BC}}$ (as the output of any corrupted player) and can, same as before, simulate the remaining transcript. □

The final step to complete the protocol, and also the proof of sufficiency of the condition $t \leq n/2$ for Broadcast, is to instantiate in protocol Broadcast the calls to $\mathcal{F}_{\text{WBC}}$ by invocations to protocol BC_{DS}. Because BC_{DS} is a secure realization of $\mathcal{F}_{\text{WBC}}$ for any $t < n$ (Lemma 6.2), hence also for $t \leq n/2$, such and instantiation does not cause loss of security.

Corollary 6.10. *If $t \leq n/2$ and a statistically (resp. computationally) secure signature scheme is available then the above protocol statistically (resp. computationally) t-securely realizes the functionality $\mathcal{F}_{BC}$.*

To complete the proof of Theorem 6.6 we show that the condition $t \leq n/2$ is necessary for adaptively secure synchronous Broadcast for statistical and computational security. The idea of the proof is the following: Because the adversary can corrupt half of the players in $\mathcal{P} \setminus \{p_s\}$, she can be the first to learn noticeable information on the dealers input, before the honest players in $\mathcal{P} \setminus \{p_s\}$ jointly learn noticeable information. Depending on this information the adversary can corrupt the sender and, with overwhelming probability, change the output to some other value. However this behavior cannot be simulated.

Lemma 6.11. *There exists no protocol which computationally t-securely realizes the functionality $\mathcal{F}_{BC}$ for $t > n/2$. The statement holds also for statistical security.*

Proof. To arrive at a contradiction, assume that there exists a computationally (resp. statistically) t-secure Broadcast protocol π. Wlog, assume that the sender p_s uses π to broadcast a randomly chosen $x_s \in_R \mathbb{F}$. For every round i, protocol π implicitly assigns to every set $\mathcal{P}' \subseteq \mathcal{P}$ a probability $\Pr_{\mathcal{P}',x_s,\pi,i}$, which is the probability of the best efficient adversary corrupting $\mathcal{P}'$ to output x_s based only on her view in π up to round i. For all $\mathcal{P}' \subseteq \mathcal{P} \setminus \{p_s\}$ this probability is negligible if i is the first round of π and overwhelming if i is the last round of π. As the total number of rounds in π is polynomial, for each $\mathcal{P}' \subseteq \mathcal{P} \setminus \{p_s\}$ there exists a round, denoted as $i_{\mathcal{P}'}$, where this probability from negligible becomes noticeable, i.e., not negligible. The adversary corrupts the set $A \subseteq \mathcal{P} \setminus \{p_s\}$ with $|A| = t-1$ such that $i_A = \min\{i_{\mathcal{P}'} \mid \mathcal{P}' \subseteq \mathcal{P} \wedge |\mathcal{P}'| \leq t-1\}$. In round i_A, the adversary gets the values which are sent to corrupted players and runs the best (efficient) strategy to compute x_s on input the view of the players in A; denote by x' the output of this protocol (by our assumption, $x' = x_s$ with noticeable probability). Let $\mathbb{F}_{1/2}$ denote the set of the first $\lfloor |\mathbb{F}|/2 \rfloor$ (in any ordering) elements in $\mathbb{F}$. If $x' \in \mathbb{F}_{1/2}$ then the adversary acts as a passive adversary (i.e., all corrupted players are instructed to correctly execute their protocol). Otherwise, i.e., if $x' \in \mathbb{F} \setminus \mathbb{F}_{1/2}$, then the adversary actively corrupts p_s and forces all the actively corrupted players to crash before sending any message in round i_A; as $|\mathcal{P} \setminus A| \leq t-1$ we know that $i_{\mathcal{P} \setminus A} \geq i_A$, hence, because the players in $\mathcal{P} \setminus A$ do not get the messages from round i_A, with overwhelming probability the output of the honest players will be in $\mathbb{F} \setminus \{x_s\}$. With this strategy the adversary achieves that when $x_s \in \mathbb{F} \setminus \mathbb{F}_{1/2}$,

then the output of the honest players in π is different than x_s with noticeable probability. However the simulator cannot simulate this behavior as he has to decide whether or not to corrupt p_s and change the output independent of x_s. □

6.7 Other Models

We chose a relatively simple model to presented our solutions for the simulatability problems of the property-based definition of Broadcast. More precisely, we assumed an adaptive adversary who can (only) actively corrupt players, and a network of synchronous secure channels. Nevertheless, the problem does not only appear in this setting. In this section we discuss how similar simulatability issues arise in authenticated networks, even asynchronous, when the adversary is adaptive. Moreover, when a mixed adversary who can also fail-corrupt or omission-corrupt players is assumed, then, even when the adversary is static, similar simulatability issues occur. We point out that most of the cases discussed here are not solved and are material of on-going research.

Synchronous Authenticated Channels

The secure-channels model is convenient for our analysis, because it is clear that the only way the adversary can learn a transmitted message is by corrupting the sender or the receiver (after the message has been received). In particular, this implies that for a message sent to the functionality, as long as the sender is honest, the simulator cannot learn the sent message before the functionality learns it. However, the simulatability issues which are exposed in this chapter apply also to authenticated channels. Indeed, unless some kind of authenticated multi-send is assumed, it might happen that the adversary learns the message which the sender intends to broadcast and can still decide to corrupt the sender and the receiver and modify it. Of course, this is not an issue when we give the simulator additional power on the communication network. For example, if the simulator is allowed to read the sent message, delete it from the channel, and then corrupt the sender and resend the message, then the above problem disappears "by definition". It is arguable, however, how consistent such a model is with the synchronicity assumption on the communication network. Furthermore, when defining such an authenticated-communication model which eliminates the composability

issue presented in this work, one has to keep in mind that the described protocols typically are not composable when the authenticated channels are replaced by secure channels.

Asynchronous Communication

Also, in the case of asynchronous communication, the same problem appears. Take for example the asynchronous secure-channels model as defined in [BCG93, BKR94]. As in the synchronous case, unless we assume some kind of asynchronous atomic multi-send, when a player p is instructed to send a message to several other players,[7] then it might happen that the adversary first learns the message, by corrupting one of the receivers, and still is able to corrupt p and change it. In fact, this is the case in the UC framework [Can01] which is the most widely accepted framework for arguing about composable security of cryptographic protocols. Clearly this behavior cannot be simulated in the ideal world. Similarly to the synchronous case, one might be willing to take a compromise and accept an asynchronous version of $\mathcal{F}_{\text{WBC}}$ to be the desired ideal functionality for Broadcast.

Mixed Adversary

What is even more surprising than the above two cases, is that as soon as we consider fail-corruption and/or omission-corruption in addition to active corruption, a similar simulatability issue occurs, *even when a static adversary is considered*. Take for example Broadcast with a fail-corrupted sender p. In any of the known Broadcast protocols tolerating active corruption and fail-corruption simultaneously (including the ones described in Chapter 4), it is possible that the player p_i who is the first to receive the value x which the sender intends to broadcast is actively corrupted, and still the adversary can crash p before he sends x to any other player. In that case the adversary can trivially change the broadcasted value to $\perp$ by having p_i pretend that he also received no value from the sender. However, this behavior cannot be simulated, as the decision of the simulator whether or not to force the sender to crash before sending x to the Broadcast functionality has to be taken independently of the sender's input.

[7] Observe that this is the way in which most asynchronous Broadcast protocols start, e.g., [Bra84].

Bibliography

[AFM99] B. Altmann, M. Fitzi, and U. Maurer. Byzantine agreement secure against general adversaries in the dual failure model. In *DISC '99*, volume 1693 of *LNCS*, pp. 123–137, 1999.

[Alt99] B. Altmann. Constructions for efficient multi-party protocols secure against general adversaries. Diploma Thesis, ETH Zurich, 1999.

[BCG93] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC '93*, pp. 52–61, 1993.

[BDDS92] A. Bar-Noy, D. Dolev, C. Dwork, and H. Strong. Shifting gears: Changing algorithms on the fly to expedite Byzantine agreement. *Information and Computation*, 97(2):205–233, 1992.

[BFH+08] Z. Beerliova, M. Fitzi, M. Hirt, U. Maurer, and V. Zikas. MPC vs. SFE: Perfect security in a unified corruption model. In *TCC 2008*, volume 4948 of *LNCS*, pp. 231–250, 2008.

[BGP89] P. Berman, J. Garray, and J. Perry. Towards optimal distributed consensus. In *FOCS '89*, pp. 410–415, 1989. Full version in *Computer Science Research*, 1992.

[BGW88] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC '88*, pp. 1–10, 1988.

[BHR07] Z. Beerliova-Trubiniova, M. Hirt, and M. Riser. Efficient Byzantine agreement with faulty minority. In *ASIACRYPT 2007*, volume 4833 of *LNCS*, pp. 393–409, 2007.

[BKR94] M. Ben-Or, B. Kelmer, and T. Rabin. Asynchronous secure computations with optimal resilience (extended abstract). In *PODC '94*, pp. 183–192. ACM, 1994.

[BPW91] B. Baum-Waidner, B. Pfitzmann, and M. Waidner. Unconditional Byzantine agreement with good majority. In *STACS '91*, volume 480 of *LNCS*, pp. 285–295, 1991.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A universally composable cryptographic library. Cryptology ePrint Archive, Report 2003/015, 2003.

[Bra84] G. Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In *PODC '84*, pp. 154–162, 1984.

[BW98] D. Beaver and A. Wool. Quorum-based secure multi-party computation. In *EUROCRYPT '98*, volume 1403 of *LNCS*, pp. 346–360, 1998.

[Can00] R. Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[Can01] R. Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS 2001*, pp. 136–145, 2001.

[Can03] R. Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003.

[CCD88] D. Chaum, C. Crépeau, and I. Damgård. Multiparty unconditionally secure protocols (extended abstract). In *STOC '88*, pp. 11–19, 1988.

[CDD+99] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT '99*, volume 1592 of *LNCS*, pp. 311–326, 1999.

[CDD+01] R. Canetti, I. Damgård, S. Dziembowski, Y. Ishai, and T. Malkin. On adaptive vs. non-adaptive security of multiparty protocols. In *EUROCRYPT 2001*, volume 2045 of *LNCS*, pp. 262–279, 2001.

[CDM00] R. Cramer, I. Damgård, and U. Maurer. General secure multiparty computation from any linear secret sharing scheme. In *EUROCRYPT 2000*, volume 1807 of *LNCS*, pp. 316–334, 2000.

[CGMA85] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *FOCS '85*, pp. 383–395, 1985.

[CR87] B. Chor and M. Rabin. Achieving independence in logarithmic number of rounds. In *PODC '87*, pp. 260–268, 1987.

[CW89] B. Coan and J. Welch. Modular construction of nearly optimal Byzantine agreement protocols. In *PODC '89*, pp. 295–305, 1989. Full version in *Information and Computation*, 97(1):61–85, 1992.

[DM00] Y. Dodis and S. Micali. Parallel reducibility for information-theoretically secure computation. In *CRYPTO 2000*, volume 1880 of *LNCS*, pp. 74–92, 2000.

[DS82] D. Dolev and H. Strong. Polynomial algorithms for multiple processor agreement. In *STOC '82*, pp. 401–407, 1982. Full version in *SIAM Journal on Computing*, 12(4):656–666, 1983.

[FHM98] M. Fitzi, M. Hirt, and U. Maurer. Trading correctness for privacy in unconditional multi-party computation. In *CRYPTO '98*, volume 1462 of *LNCS*, pp. 121–136, 1998. Corrected version is available online.

[FHM99] M. Fitzi, M. Hirt, and U. Maurer. General adversaries in unconditional multi-party computation. In *ASIACRYPT '99*, volume 1716 of *LNCS*, pp. 232–246, 1999.

[FHW04] M. Fitzi, T. Holenstein, and J. Wullschleger. Multi-party computation with hybrid security. In *EUROCRYPT 2004*, volume 3027 of *LNCS*, pp. 419–438, 2004.

[Fit03] M. Fitzi. *Generalized Communication and Security Models in Byzantine Agreement*. PhD thesis, ETH Zurich, 2003.

[FLM86] M. Fischer, N. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1:26–39, 1986.

[Gen95] R. Gennaro. Achieving independence efficiently and securely. In *PODC '95*, pp. 130–136, 1995.

[Gen00] R. Gennaro. A protocol to achieve independence in constant rounds. *IEEE Trans. Parallel Distrib. Syst.*, 11(7):636–647, 2000.

[GGBS08] A. Gupta, P. Gopal, P. Bansal, and K. Srinathan. Authenticated byzantine generals strike again. Cryptology ePrint Archive, Report 2008/287, 2008.

[GL02] S. Goldwasser and Y. Lindell. Secure computation without agreement. In *DISC '02*, volume 2508 of *LNCS*, pp. 17–32, 2002.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *FOCS '86*, pp. 174–187, 1986.

[GMW87] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game — a completeness theorem for protocols with honest majority. In *STOC '87*, pp. 218–229, 1987.

[Gol03] O. Goldreich. *Foundations of Cryptography: Volume1, Basic Tools*. Cambridge University Press, New York, NY, USA, 2003.

[Gol04] O. Goldreich. *Foundations of Cryptography: Volume 2, Basic Applications*. Cambridge University Press, New York, NY, USA, 2004.

[GP92] J. Garay and K. Perry. A continuum of failure models for distributed computing. In *WDAG '92*, volume 647 of *LNCS*, pp. 153–165, 1992.

[Hev06] A. Hevia. Universally composable simultaneous broadcast. In *SCN 2006*, volume 4116 of *LNCS*, pp. 18–33, 2006.

[HM97] M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *PODC '97*, pp. 25–34, 1997. Full version appeared in *Journal of Cryptology*,13(1): 31–60, 2000.

[HM00] M. Hirt and U. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000. Extended abstract in *Proc. 16th of ACM PODC '97*.

[HM05] A. Hevia and D. Micciancio. Simultaneous broadcast revisited. In *PODC '05*, pp. 324–333, 2005.

[HMP00] M. Hirt, U. Maurer, and B. Przydatek. Efficient secure multi-party computation. In *ASIACRYPT 2000*, volume 1976 of *LNCS*, pp. 143–161, 2000.

[HMZ08] M. Hirt, U. Maurer, and V. Zikas. MPC vs. SFE: Unconditional and computational security. In *ASIACRYPT 2008*, volume 5350 of *LNCS*, pp. 1–18, 2008.

[HZ10a] M. Hirt and V. Zikas. Adaptively secure broadcast. In *EUROCRYPT 2010*, volume 6110 of *LNCS*, pp. 466–485, 2010.

[HZ10b] M. Hirt and V. Zikas. Player-centric Byzantine agreement. Manuscript, 2010.

[IKLP06] Y. Ishai, E. Kushilevitz, Y. Lindell, and E. Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *CRYPTO 2006*, volume 4117 of *LNCS*, pp. 483–500, 2006.

[Koo06] C. Koo. Secure computation with partial message loss. In *TCC 2006*, volume 3876 of *LNCS*, pp. 502–521, 2006.

[KY84] A. Karlin and A. Yao. Manuscript, 1984.

[LF82] L. Lamport and M. Fischer. Byzantine generals and transaction commit protocols. Technical Report Opus 62, SRI International (Menlo Park CA), TR, 1982.

[LLR02] Y. Lindell, A. Lysyanskaya, and T. Rabin. On the composition of authenticated Byzantine agreement. In *STOC 2002*, pp. 514–523, 2002.

[LSP82] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[Mau02] U. Maurer. Secure multi-party computation made simple. In *SCN 2002*, volume 2576 of *LNCS*, pp. 14–28, 2002. Full version appeared in *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[Mau06] U. Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.

[Mau09a] U. Maurer. Cryptography. Lecture Notes, ETH Zurich, 2009.

[Mau09b] U. Maurer. Information security. Lecture Notes, ETH Zurich, 2009.

[MS94] U. Maurer and P. Schmid. A calculus for secure channel establishment in open networks. In *ESORICS '94*, volume 875 of *LNCS*, pp. 175–192, 1994.

[PL80] M. Pease and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27:228–234, 1980.

[PR03] P. Parvedy and M. Raynal. Uniform agreement despite process omission failures. *Parallel and Distributed Processing Symposium, International*, 0:212b, 2003.

[PSW00] B. Pfitzmann, M. Schunter, and M. Waidner. Secure reactive systems. IBM Research Report RZ 3206, 2000.

[PT86] K. Perry and S. Toueg. Distributed agreement in the presence of processor and communication faults. *IEEE Trans. Softw. Eng.*, 12(3):477–482, 1986.

[PW92] B. Pfitzmann and M. Waidner. Unconditional Byzantine agreement for any number of faulty processors. In *STACS '92*, volume 577 of *LNCS*, pp. 339–350, 1992.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy*, pp. 184–200, 2001.

[Ray02] M. Raynal. Consensus in synchronous systems: A concise guided tour. In *PRDC '02: Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing*, p. 221, Washington, DC, USA, 2002. IEEE Computer Society.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC '89*, pp. 73–85, 1989.

[SHZI02] J. Shikata, G. Hanaoka, Y. Zheng, and H. Imai. Security notions for unconditionally secure signature schemes. In *EUROCRYPT 2002*, volume 2332 of *LNCS*, pp. 434–449, 2002.

[Yao82] A. Yao. Protocols for secure computations. In *FOCS '82*, pp. 160–164, 1982.

[ZHM09] V. Zikas, S. Hauser, and U. Maurer. Realistic failures in secure multi-party computation. In *TCC 2009*, volume 5444 of *LNCS*, pp. 274–293, 2009.