Combining Computational and Information-Theoretic Security in Multi-Party Computation

Master’s Thesis

Christoph Lucas
c lucas@student.ethz.ch

Professor: Prof. Ueli Maurer
Assistant: Dominik Raub

ETH Zurich,
Department of Computer Science,
Institute of Theoretical Computer Science,
Clausiusstr. 59, CH-8092 Zurich
Abstract

Most proposed protocols for multi-party computation are designed to be secure either against information-theoretic or against computational adversaries. Each setting has its known advantages and disadvantages. In the computational setting we can achieve non-robust MPC for up to \( t < n \) corrupted parties, but relying on unproven intractability assumptions. In the information-theoretic setting we do not have any unproven assumptions, but only \( t < n/2 \) corrupted parties can be tolerated.

In [Cha89] Chaum sketches a hybrid protocol that brings together the best of both worlds: Information-theoretic security with abort (i.e. privacy and correctness, but no fairness or robustness) is achieved for \( t < n/2 \) corrupted parties. Beyond this threshold, for \( t < n \) corrupted parties, the protocol achieves computational security with abort.

In this work, we first formalize the protocol idea described in [Cha89] and prove its security. In a second step, we extend the protocol to provide fairness and robustness. Whereas fairness comes “for free”, robustness requires a trade-off similar to the one in [IKLP06]. Then, we describe a construction for realizing the protocol in the universally-composable setting as proposed in [Can01].

For his protocol, Chaum assumed that a broadcast channel is given. In the last part of this work we discuss the implications of dropping this assumption, and obtain an interesting new result: It is possible to achieve the same bounds as in [Cha89], extended with fairness, without a broadcast channel.
## Contents

1 Introduction 4
   1.1 Secure multi-party computation 4
   1.2 Hybrid security 4
   1.3 Previous work 5
   1.4 Contributions 5

2 Definitions and notations 7
   2.1 Definition of security 7
      2.1.1 Ideal vs. real model 7
      2.1.2 Security in the stand-alone setting 7
      2.1.3 Security in the universally composable setting 9
   2.2 Notions of security 10
      2.2.1 Security with abort 10
      2.2.2 Security with a designated party 11
      2.2.3 Ideal model without security 12
   2.3 Symmetric vs. asymmetric functionalities 12

3 The protocol 13
   3.1 An intuition 13
   3.2 Broadcast channels 13
   3.3 Protocol $\pi^N$: Simulating the “Neutral Zone” 14
      3.3.1 Initialization 14
      3.3.2 Calculation 14
      3.3.3 Sending and receiving private messages 15
      3.3.4 Sending and receiving broadcast messages 15
   3.4 Protocol $\pi^C$: Computing the output (stand-alone setting) 16
      3.4.1 Using [GMW87] as $\pi^C$ 16
      3.4.2 Recap of the original protocol 16
      3.4.3 The modifications 17
      3.4.4 Proof of Lemma 3.2 18
   3.5 Protocol $\pi^C$: Computing the output (UC setting) 20
      3.5.1 Oblivious transfer 21
      3.5.2 Commitments 21
      3.5.3 Zero-knowledge proofs 22
      3.5.4 Commit-and-prove 22
      3.5.5 The general construction for passive adversaries 23
      3.5.6 Two-party computation with an active adversary 23
      3.5.7 Multi-party computation with active adversaries 23
   3.6 The complete protocol $\Pi_f$ 24
   3.7 Security of $\Pi_f$ 25

4 The proof of security in the UC setting 26
   4.1 The goal 27
   4.2 Proof for the information-theoretic case in the UC setting 27
      4.2.1 Using the simulator for protocol $\pi^N$ 28
      4.2.2 Replacing the broadcast channel 29
1 Introduction

1.1 Secure multi-party computation

In [Yao82], Yao introduced the idea of Multi-Party Computation (MPC): Given any arbitrary but fixed function, and a set of $n$ mutually distrusting parties, an MPC protocol enables those parties to compute this function on their input, while providing certain security guarantees. Even if some of the parties are corrupted by an active adversary, the private input of the honest parties should remain secret and the output should be correct.

In [GMW87], Goldreich, Micali, and Wigderson provide a first general solution for the MPC problem, which is based on intractability assumptions and uses a synchronous broadcast channel as underlying communication resource. They achieve security against $t < n/2$ corrupted parties. Ben-Or, Goldwasser, Wigderson [BGW88], and independently Chaum, Crépeau, Damgård [CCD88] present protocols which are information-theoretically (IT) secure and are only based on a complete network of secure and synchronous channels, where every pair of parties is connected. However, they proof that security can only be achieved as long as $t < n/3$ parties are corrupted. When additionally provided with a broadcast channel, this bound can be extended to $t < n/2$ [RB89, Bea91, CDD+99].

These bounds are tight for full security including privacy of the input, correctness of the output, fairness and robustness. For weaker security requirements, a higher number of corrupted parties can be tolerated. [Gol01] proposes a variant of the protocol in [GMW87] that provides computational privacy and correctness against any number of corrupted parties, losing robustness and fairness. This protocol is also based on a synchronous broadcast channel.

1.2 Hybrid security

Most known multi-party protocols are designed to be secure either against information-theoretic or against computational adversaries. Information-theoretic protocols have the disadvantage that only a corrupted minority can be tolerated. Otherwise, the security cannot be guaranteed anymore. On the other hand, computational protocols are based on intractability assumptions. As soon as the underlying problem can be solved efficiently, even a single party might break the security. However, as long as the computational assumption holds, privacy and correctness can be achieved for any number of corrupted parties.

Now, the goal of MPC protocols with hybrid security is to provide different levels of security, depending on the number of corrupted parties. In the first part of this work, the levels are basically information-theoretic and computational security. This distinction will be refined in later sections. However, as discussed in the next section, other interpretations on the theme of “hybrid security” are possible.
1.3 Previous work

The goal of the construction described in [Cha89], on which our work is based, is to provide computational privacy and correctness for any number of actively corrupted parties, and simultaneously information-theoretic privacy and correctness, given that only a minority of the parties is corrupted.

[FHW04] uses another approach to combine IT and computational security. The protocol there provides three thresholds: Up to the first threshold $t_p$, the security is unconditional. Between $t_p$ and the second threshold $t_{\sigma}$ security can only be provided as long as the underlying public-key infrastructure (PKI) is consistent. Finally, between $t_{\sigma}$ and $T$ the protocol is as secure as the signature scheme in use. In [FHW04] they focus on computational signature schemes. However, information-theoretic schemes could be used as well. They show that their notion of hybrid MPC is achievable if and only if $(2T + t_p < n) \land (T + 2t_{\sigma} < n)$, which is tight.

Yet another notion of hybrid security is discussed in [FHHW03]. This protocol provides more flexibility with two thresholds $t_v$ and $t_c$ in the IT setting, when no broadcast channel is available [BGW88, CCD88]. For $t \leq t_v \leq n/3$ corrupted parties, fully secure MPC is achieved. For $t_v < t \leq t_c$ corrupted parties only non-robust (but fair, private, and correct) MPC is possible. Beyond $t_c$ no security guarantees are given. Hence, the tradeoff here is not between computational and IT bounds, but between robust and non-robust MPC. This so called Detectable Multi-Party Computation is possible if and only if $t_v = 0$ or $t_v + 2t_c < n$.

1.4 Contributions

First of all, this work formalizes the protocol sketch presented in [Cha89] and provides a rigorous proof of security. The original protocol idea was based in a stand-alone security model, which does not guarantee security in the case where the protocol is only a component in a larger application. [Can01] proposes a new framework for universally composable (UC) protocols. We adapt protocols providing UC security to fit our needs and show how the protocol idea from [Cha89] can be realized in a universally-composable way.

Furthermore, we extend the achieved security properties. The original protocol is only concerned with privacy of the input and correctness of the output. In the second part of this work, we investigate the possibility to add fairness (i.e. the adversary only learns as much about the output as the honest parties) and robustness (i.e. the adversary cannot abort the protocol anymore). Whereas fairness is obtained “for free”, robustness requires a trade-off.

We begin with the assumption that a broadcast channel is provided. With the original protocol as sketched in [Cha89] we achieve security with abort (i.e. privacy and correctness, but no fairness or robustness). For the computational setting this holds for $t < n$ actively corrupted parties, which is obviously optimal. In the IT setting, we achieve the desired security for $t < \frac{n}{2}$ actively corrupted parties. This is optimal according to [Kil00].
In a next step we extend the protocol to achieve fairness. We obtain IT security with fairness for \( t < \frac{n}{2} \) and computational security with abort for \( t < n \) actively corrupted parties. According to [Cle86] fairness cannot be achieved beyond \( \frac{n}{2} \). Hence, this result is tight.

Considering robustness, for every \( \rho < \frac{n}{2} \), we achieve full IT security up to \( t \leq \rho \), IT security with fairness up to \( t < \frac{n}{2} \), and computational security with abort up to \( t < n - \rho \) actively corrupted parties. The bounds for robustness and for security with abort are tight according to [IKLP06, Kat07]. Again, the bound for fairness is tight according to [Cle86].

In Section 8 we investigate the implications of dropping the assumption that a broadcast channel is available. In this setting, for \( \rho > 0 \), we achieve full IT security (with robustness) for \( t \leq \rho < \frac{n}{2} \), and IT security with fairness for \( t \leq \frac{n - \rho}{2} \) actively corrupted parties. According to [FHHW03], no IT security can be achieved beyond that bound, nor do we achieve any computational security. This result was already presented in [FHHW03].

However, for the special case that \( \rho = 0 \), we achieve information-theoretic fairness for \( t < \frac{n}{2} \), and computational security with abort for \( t < n \) corrupted parties, which goes beyond [FHHW03] (highlighted in Figure 1). The computational security beyond \( \frac{n}{2} \) is due to the particular construction in [Cha89].
2 Definitions and notations

2.1 Definition of security

2.1.1 Ideal vs. real model

In simulation based security models, the security of a protocol (the real model) is defined with respect to an ideal model, where the computation is performed by a **Trusted Third Party** or **Ideal Functionality** $I$. In reality, however, such an ideal model $I$ may not be available and the task of a protocol $\pi$ is to implement it from a given set of resources (e.g. authentic or secure communication channels, broadcast channels, or a public-key infrastructure) using local protocol machines $\pi_i$ for each party $P_i$.

![Diagram of ideal vs real model](image)

(a) The ideal model. (b) The real model.

Figure 2: The ideal vs the real model.

Informally, a protocol achieves security according to the simulation paradigm if whatever the adversary can do in the real model, he could also do in the ideal model. With $I_f$ we denote an ideal functionality that computes a function $f$, and $\pi_f$ is a protocol that implements this ideal functionality. In the real model, an adversary $A$ may corrupt a subset $A \subseteq [n]$ (where $[n] = \{1, 2, ..., n\}$) containing $t$ parties. We assume that the adversary $A$ does not take any external input. However, it may provide input for the corrupted parties. Let $H \subseteq [n]$ denote the subset of indices belonging to the honest parties (hence $H = [n] \setminus A$). For the sake of notational simplicity and without loss of generality, we usually assume the corrupted parties to be $P_{1:t} = P_1, ..., P_t$. The labeling of parties is irrelevant since labels are only names and renaming is always possible.

2.1.2 Security in the stand-alone setting

To formalize the above notion we have to define what it means to be able to achieve “essentially the same” in both models. This is usually done using an ideal adversary (or simulator) $\sigma$ for the ideal system which usually has black-box access to the adversary $A$. This is called **black-box security** and is stronger than what we actually need. For stand-alone security it is enough if for every adversary there is a simulator and this simulator is generally not restricted to black-box access. However, nearly all proofs in the literature
(and all proofs in this work) are black-box proofs. Now, a protocol is secure in the stand-alone setting if we are able to construct a simulator $\sigma$ such that for every adversary $A$ the outputs $(y_A, y_{t+1}, ..., y_n)$ and $(y_\sigma, y_{t+1}, ..., y_n)$ are indistinguishable for every input (see Figure 3).

![Diagram](image_url)

(a) The real model with adversary.  
(b) The ideal model $\mathcal{I}_f^{\text{sec}}$ with simulator.

Figure 3: The ideal vs the real model including an adversary.

With this notation we can formally define the above notion:

**Definition 2.1. Stand-alone Security.** Given an ideal model $\mathcal{I}_f$ computing a function $f$, a class of distinguishers $\mathcal{D}$, a class of adversaries $\mathcal{A}$ and a class of simulators $\Sigma$. We say that a protocol $\pi_f$ securely implements $\mathcal{I}_f$ in the stand-alone setting if for any adversary $A \in \mathcal{A}$ in the real model there is a simulator $\sigma \in \Sigma$ for the ideal model, such that for every distinguisher $D \in \mathcal{D}$, providing input $(x_1, ..., x_n)$,

$$adv_D((y_A, y_{t+1}, ..., y_n), (y_\sigma, y_{t+1}, ..., y_n)) \leq \epsilon(\kappa)$$

holds, where $\epsilon(\kappa)$ is a negligible function in the security parameter $\kappa$.

In the information-theoretic case we let $\mathcal{A} = \mathcal{A}^{\text{all}}$ be the class of all adversaries, $\Sigma = \Sigma^{\text{eff}}$ be the class of efficient simulators, and $\mathcal{D} = \mathcal{D}^{\text{all}}$ be the class of all distinguishers. On the other hand, in the computational case, we restrict the classes to be $\mathcal{A} = \mathcal{A}^{\text{eff}}$ the class of efficient adversaries, $\Sigma = \Sigma^{\text{eff}}$ the class of efficient simulators, and $\mathcal{D} = \mathcal{D}^{\text{eff}}$ the class of efficient distinguishers.

In this definition we use the notion of an advantage, that a distinguisher has given one system or the other. This advantage for a single distinguisher and a class of distinguishers respectively, is defined as follows (see [Mau06, Def. 2.2]):

**Definition 2.2. Advantage of a distinguisher.** For systems $S$ and $S'$ the advantage of a distinguisher $D$ in distinguishing $S$ and $S'$, denoted $adv^D(S, S')$, is defined as

$$adv^D(S, S') := |P(D(S) = 1) - P(D(S') = 1)|.$$ 

The advantage of a class $\mathcal{D}$ of distinguishers in distinguishing $S$ and $S'$, denoted $adv^D(S, S')$, is

$$adv^D(S, S') := \max_{D \in \mathcal{D}} adv^D(S, S').$$
2.1 Definition of security

In the stand-alone setting, there is no interaction between the distinguisher and the system other than providing inputs and receiving outputs. To illustrate this fact in the definition, we directly use the string of outputs on which the distinction is based.

2.1.3 Security in the universally composable setting

One problem with the above definition is that protocols that securely implement an ideal functionality cannot be composed freely. They are “stand-alone”. This is due to the fact that the simulator has access to the adversary, i.e. the simulator knows about the environment. The distinguisher cannot interact freely with the system and base the distinction on the complete transcript of such an interaction. After providing the input, the distinguisher only receives the output.

In a more general setting, these restrictions should not apply. For general composition we want to be able to interchange ideal functionalities with real protocols without losing security in the real scenario. This is done using a universal simulator for the ideal model that does not depend on the environment in any way. The idea is that every adversary could theoretically connect this simulator to its interface of the ideal model and the resulting system should be indistinguishable from the real protocol execution.

Since this is the case for all adversaries, we could replace the real protocol execution with the ideal model and the simulator, no matter what the environment is. The result would always be indistinguishable, even if the distinguisher may now interact freely with the system and is not restricted anymore to the output.

This idea was first presented in [PW94, PW00, PW01, BPW04]. Universal composability was introduced in [Can01]. An equivalent and, in our opinion more intuitive definition of security was given in [Mau06]:

**Definition 2.3. Universally Composable (UC) Security.** Given an ideal model $I_f$ computing a function $f$, a class of distinguishers $D$ and a class of simulators $\Sigma$. We say that a protocol $\pi_f$ securely implements $I_f$ in the universally composable setting if there is a simulator $\sigma \in \Sigma$ such that

$$\text{adv}^D(\sigma_{\mathcal{A}}(I_f), \pi_f^I(R)) \leq \epsilon(\kappa)$$

holds, where $\epsilon(\kappa)$ is a negligible function in the security parameter $\kappa$. $\sigma_{\mathcal{A}}$ is a universal simulator, connected to the interfaces belonging to the set of corrupted parties $\mathcal{A}$. $\pi_f^I$ denotes the protocol machines of the protocol $\pi_f$ run by the set of honest parties $\mathcal{H}$.

In other words, wherever a protocol $\pi_f$ is used, we can replace this protocol by the ideal functionality it implements together with a simulator, and the result is indistinguishable. The main difference as opposed to the stand-alone case is that now the simulator does not have access to the adversary. As a consequence, rewinding techniques as used in many proofs of stand-alone security cannot be applied.
2.2 Notions of security

The term ‘security’ can be broken down into (at least) five different notions. The following informal definitions are taken partially from [HM06]:

**Privacy**: A protocol achieves privacy if the adversary cannot learn more about the honest players’ input than given by the inputs and outputs of the corrupted players.

**Correctness**: A protocol achieves correctness if the output equals the function value $f(x_1, ..., x_n)$ or there is no output.

**Robustness**: A protocol achieves robustness if an adversary cannot abort the protocol and prevent the honest players from obtaining an output, once the input is provided.

**Fairness**: A protocol achieves fairness if the adversary cannot abort the protocol with an advantage in knowledge about the output. That is, the honest parties will learn whatever the adversary learns.

**Agreement on abort**: A protocol achieves agreement on abort if the adversary can only abort either all honest parties or none. Hence, in the end the honest parties have an agreement on whether the protocol execution was aborted or not.

Demanding security with respect to the ideal functionality $I_{f^{sec}}$ is equivalent to demanding all five notions (actually only the first three, as fairness and agreement on abort are implied by robustness). Of course we can relax our definition of security to encompass only a subset of the notions above. In this work we mainly focus on privacy and correctness with agreement on abort. In the following we provide a formal specification of this security goal by describing a suitable ideal functionality.

### 2.2.1 Security with abort

When the only concerns are privacy and correctness with agreement on abort, there is no guarantee for the honest parties that they will learn anything about the output (no robustness), not even in the case where the adversary learns everything about it (no fairness). Of course, such protocols do not implement $I_{f^{sec}}$ as defined above. Hence, we have to adapt the ideal model that a protocol has to implement to fit in this new scenario (see Figure 4).

![Figure 4: The ideal model with abort $I_{f^{abort}}$.](image-url)
2.2 Notions of security

We use ⊥ to represent the output of honest parties resulting from the adversary aborting the protocol. Now, first, all parties send their inputs to $I_{f}^{\text{abort}}$. Then, $I_{f}^{\text{abort}}$ computes the output according to $f$ and sends the result to the simulator. Given these values, the simulator decides whether the other parties also receive the output (flag 1) or not (flag 0). Finally, $I_{f}^{\text{abort}}$ sends either the output $y$ or the empty value $\bot$ to the honest parties, depending on the flag received by the simulator. It is important to note that the adversary can only abort either all honest parties, or none. Hence, the honest parties have an agreement on abort. We could relax the definition further by enabling the adversary to send one flag for each party. However, this relaxation is not necessary here as our protocols always achieve agreement on abort.

Demanding privacy and correctness only for the computation of $f$ is captured by using the ideal functionality $I_{f}^{\text{abort}}$ in the definition of security (for stand-alone, Def. 2.1, and UC, Def. 2.3, likewise).

2.2.2 Security with a designated party

We define another ideal model $I_{f}^{\text{des}}$ that specifies security relying on a designated party $P_{N}$ being uncorrupted. As long as this holds, privacy of $P_{N}$’s input and correctness (for all honest parties) with agreement on abort are guaranteed.

![Figure 5: The ideal model $I_{f}^{\text{des}}$ with a designated party.](image)

So we assume that the designated party $P_{N}$ is not corrupted (in the case where $P_{N}$ is corrupted we demand no security at all, formalized by the ideal functionality $I_{f}^{\text{NoSec}}$ in Section 2.2.3). As in the previous case, first all parties send their inputs to the ideal model $I_{f}^{\text{des}}$. The inputs of all honest parties but $P_{N}$ are directly forwarded to the simulator. Then, $I_{f}^{\text{des}}$ computes the output and sends the result to the simulator. Given these values, the simulator decides whether the other parties also receive the output (flag 1) or not (flag 0). Finally, $I_{f}^{\text{des}}$ sends either the output $y$ or the empty value $\bot$ to the honest parties, depending on the flag received by the simulator. Again, note that the honest parties have agreement on abort.

Demanding these special security properties for the computation of $f$ is captured by using the ideal functionality $I_{f}^{\text{des}}$ in the corresponding definition of security (for stand-alone, Def. 2.1, and UC, Def. 2.3, likewise).
2.2.3 Ideal model without security

For the sake of completeness we present an ideal model that demands no security whatsoever, together with a generic UC simulator that only depends on a black-box of the protocol, and is independent from the adversary. All that the ideal functionality \( I_{f^{\text{NoSec}}} \) does is to forward the honest parties' input directly to the simulator, take whatever is the simulator’s output \( \{y_i\}_{i \in \mathcal{N}} \) and send it back to the honest parties. The simulator \( \sigma_{f^{\text{NoSec}}} \) uses this input to simulate the honest protocol machines. Note that in this case, the output may be asymmetric.

This already proves the following (rather trivial) theorem:

**Theorem 2.1.** Any protocol \( \pi_f \) securely implements the ideal model \( I_{f^{\text{NoSec}}} \) without security in the universally composable setting.

2.3 Symmetric vs. asymmetric functionalities

A symmetric functionality is a functionality where each party receives the same output: \( y = f(x_1, ..., x_n) \). In contrast to that, an asymmetric functionality provides a separate output for each party: \( (y_1, ..., y_n) = f(x_1, ..., x_n) \).

On first glance an asymmetric functionality seems to be more powerful than a symmetric one. However, a simple argument shows that asymmetric MPC can be reduced to symmetric MPC under our notion of security: In addition to its input, each party provides \( m_i \) random bits \( r_i \) to a slightly modified functionality (where \( m_i \) is the length of the output \( y_i \)). Then, instead of \( (y_1, ..., y_n) \), this modified functionality computes \( y = (y_1 \oplus r_1, ..., y_n \oplus r_n) \) and provides the same output \( y \) to all parties. Finally, each party recovers its own output.

It follows from the security of the One-Time-Pad (OTP) encryption that no party is able to obtain any information about the other parties’ output, even with unbounded computational resources, given that the input remains private. This idea was presented in [GMW87] for computational security with asymmetric encryption. The generalization to the IT case with an OTP encryption was discussed e.g. briefly in [CDG88] (there called private output).

As a consequence, it is sufficient to consider symmetric functionalities, which will simplify the discussion. In some cases throughout this work we use indexed outputs \( y_i \). However,
this is not an indication for asymmetric output, but denotes the output received at interface $i$. Naturally, all these outputs are supposed to be equal, but an adversary might be able to change some party’s output arbitrarily. Therefore we need to distinguish the output received by different parties.

3 The protocol

In this section we present a detailed instantiation of the protocol sketched in [Cha89]. The complete protocol $\Pi_f$ is based on two subprotocols, which we denote with $\pi^C$ and $\pi^N$. Now, we first describe the idea behind the protocol. Then we discuss the two subprotocols $\pi^C$ and $\pi^N$ separately, also investigating in more detail the use of broadcast channels. Finally we combine the components in the complete protocol and give a formal description thereof together with a formal statement of the security it achieves.

3.1 An intuition

We first present Chaum’s idea. The goal of any MPC is to compute a possibly randomized mapping $y = f(x_1, ..., x_n)$. The protocol sketch in [Cha89] demands that each party $P_i$ locally shares its input into $x_i = x^C_i \oplus x^N_i$. Then the protocol $\Pi$ calls a subprotocol $\pi^C$ which computes $y = f'(x^C_1, ..., x^C_n, (x^N_1 \parallel ... \parallel x^N_n))$, where the last input will be provided by a simulated party, which Chaum calls the “Neutral Zone” (or short just $P_N$).

The simulation of $P_N$ is done with an information-theoretically secure protocol $\pi^N$ among all $n$ parties, where each party $P_i$ inputs its $x^N_i$. By contrast, the computation of the output is done with a computationally secure protocol $\pi^C$ among the $n$ parties $P_i$ and the simulated party $P_N$ (hence, this is an $n+1$ party MPC). In this protocol $P_i$ inputs $x^C_i$ and $P_N$ inputs $(x^N_1 \parallel ... \parallel x^N_n)$. So, the computation of $f'$ by the $n+1$ party protocol $\pi^C$ first reconstructs the input $x_i = x^C_i \oplus x^N_i$, and then computes $f$ on the $x_i$. Thus each party $P_i$ runs two protocol machines $\pi^C_i$ and $\pi^N_i$.

3.2 Broadcast channels

Since the protocol $\Pi$ is built on two subprotocols $\pi^C$ and $\pi^N$, we assume the existence of two broadcast channels: a normal one for $\pi^N$, and a modified broadcast channel $BC'$ defined as follows:

**Definition 3.1. Modified broadcast $BC'$.** On one hand, $BC'$ has a standard broadcast connection to each protocol machine $\pi^C_i, i \neq P_N$. On the other hand, there are another $n$ connections all of which are connected to the designated protocol machine $\pi^C_N$. This makes a total of $2n$ connections. Now, when protocol machine $\pi^C_i, i \neq N$ broadcasts a message, $BC'$ sends a copy over all $2n-1$ connections. In contrast, when the designated protocol machine $\pi^C_N$ broadcasts a message, $BC'$ receives $n$ messages over the $n$ connections from $\pi^C_N$, concatenates them and sends them as a single message to the protocol machines $\pi^C_i, i \neq N$. 
3.3 Protocol $\pi^N$: Simulating the “Neutral Zone”

The modifications are necessary since the simulated party $P_N$ does not have an own broadcast interface. In Section A.2 we show how to multiplex a single (normal) broadcast channel in order to provide the necessary broadcast channels for the two subprotocols $\pi^C$ and $\pi^N$.

There is one more issue concerning the modified broadcast channel. According to the definition above, when the designated protocol machine $\pi^C_N$ broadcasts a message, the protocol machines $\pi^C_i, i \neq N$ receive $n$ concatenated messages. This is not what they expect according to their specification (see Section 3.4). In order to solve this inconsistency we define a Majority Vote, implemented by a local machine $\pi^v$, which each party uses (locally) to reduce the number of messages to a single one (see Section 3.3.4).

**Definition 3.2. Majority Vote.** Let $\pi^v$ denote a local machine implementing a majority vote with the following properties: All messages not coming from the designated party $P_N$ are forwarded without modification. Broadcast messages from the designated party $P_N$, however, consist of $n$ parts $b_{ci}$. If $\exists m : |\{b_{ci} | b_{ci} = m\}| \geq n/2$, $\pi^v$ forwards $m$ to the protocol machine $\pi^C_i$. Otherwise it sends $\perp$ to $\pi^C_i$.

3.3 Protocol $\pi^N$: Simulating the “Neutral Zone”

We will now focus on the first component in the complete protocol, the subprotocol $\pi^N$. This subprotocol is used for the simulation of the designated party $P_N$. The first step is to formally state the security, which the subprotocol $\pi^N$ has to provide. In a second step we further investigate how this simulated party interacts with the other protocol machines during the execution of the second subprotocol $\pi^C$.

For the simulation of the designated party we need an IT secure protocol $\pi^N$ as stated in the following lemma:

**Lemma 3.1.** [RB89] There is a protocol $\pi^N$ that implements the ideal model $I_{sec}^r$ (without abort) with IT security (i.e. $A = A^{all}$, $\Sigma = \Sigma^{eff}$, and $D = D^{all}$), from a complete and synchronous network of secure channels, and a broadcast channel, in the UC setting, given that $t < \frac{n}{2}$ parties are corrupted.

For the actual simulation, party $P_N$ has to send and receive messages, and perform some calculations on the input and the received messages. We now show how this interaction with other protocol machines is achieved.

3.3.1 Initialization

The initialization happens according to the protocol $\pi^N$. No modifications are needed.

3.3.2 Calculation

During the simulation, the party $P_N$ has to perform calculations on its own input and the information received in messages (see below). All these pieces of information are shared
3.3 Protocol $\pi^N$: Simulating the “Neutral Zone”

among the protocol machines $\pi_i^N$. Hence, all computations can be done according to the protocol $\pi^N$.

3.3.3 Sending and receiving private messages

During the protocol execution of $\pi^C$, the simulated party $P_N$ has to be able to send and receive private messages. However, there is no protocol machine $\pi_C^N$ for this party. The protocol machine $\pi_C^N$ is simulated by $n$ protocol machines $\pi_i^N$. Still, each protocol machine $\pi_i^C$ in protocol $\pi^C$ has an interface for a private channel to the protocol machine $\pi_C^N$, which has to be connected to some protocol machine. Here we exploit the fact that always corresponding parties are corrupted. Hence, when the protocol machine $\pi_i^C$ is corrupted, so is the protocol machine $\pi_i^N$. We thus connect each pair $\pi_i^C$ and $\pi_i^N$ of protocol machines directly.

Now, in the case where the protocol machine $\pi_i^C$ has to send a message $m$ to party $P_N$, the protocol machine $\pi_i^C$ sends $m$ to the protocol machine $\pi_i^N$. Then, $\pi_i^N$ provides the message as input to protocol $\pi^N$ as prescribed by the protocol (normally this is done with some kind of sharing among all $\pi_j^N$).

When party $P_N$ has to send a message to protocol machine $\pi_i^C$, first an MPC is performed where the content of the message is computed using protocol $\pi^N$. Then, the protocol machine $\pi_i^N$ receives as output the message, whereas $\forall j \neq i : \pi_j^C$ receive $\perp$. Finally, the protocol machine $\pi_i^N$ forwards the message to protocol machine $\pi_i^C$.

3.3.4 Sending and receiving broadcast messages

Since we allow the use of a broadcast channel $BC'$ (see Def. 3.1) for the computational protocol $\pi^C$, we connect each protocol machine $\pi_i^N$ to one of the $n$ connections of $BC'$ corresponding to party $P_N$. Furthermore, for each party $P_i$, we interconnect a local machine $\pi_i^v$ implementing the majority vote (see Def. 3.2) between $BC'$ and the protocol machines $\pi_i^C, i \neq N$.

When protocol machine $\pi_i^C, i \neq N$ is required to broadcast a message $m$, nothing changes: It sends $m$ to $\pi_i^v$ which forwards the message to $BC'$, and every party receives a copy of it. On the other hand, when party $P_N$ has to broadcast a message $m$, an MPC is performed where the content of the message is computed using protocol $\pi^N$. Then, every protocol machine $\pi_i^N$ receives $m$ as output and sends it to $BC'$. The local machines implementing the majority vote will take care that each protocol machine $\pi_i^C$ receives a single message and the correct one as long as $t < \frac{n}{2}$ parties are corrupted.

This concludes our discussion on the simulation of the designated party $P_N$. We have stated the security requirements for the corresponding subprotocol $\pi^N$, and discussed the necessary construction for the interaction between the simulated protocol machine $\pi_C^N$ and the other protocol machines $\pi_i^C$ in protocol $\pi^C$. 
3.4 Protocol $\pi^C$: Computing the output (stand-alone setting)

In this section we focus on the second component of the complete protocol. For the computation of the actual output we perform a multiparty-computation among the $n$ parties $P_{1:n}$ and the simulated party $P_N$. The protocol will be denoted as $\pi^C$. In the previous section we have seen how the protocol machine $\pi^C_N$ for the designated party $P_N$ can be simulated with an IT secure protocol $\pi^N$. Now, again we first state the kind of security the protocol $\pi^C$ has to provide. After that, we show how to construct a protocol that fulfills these requirements, taking into account a designated party. In this section we provide a protocol with stand-alone security. Section 3.5 discusses the UC setting.

As in the previous case, the protocol has to provide certain security guarantees as stated in the following lemma:

**Lemma 3.2.** There is a protocol $\pi^C$ that implements

1. the ideal model $I^1_{\text{abort}}$ (with abort), with computational security (i.e. $\mathcal{A} = \mathcal{A}^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $\mathcal{D} = \mathcal{D}^{\text{eff}}$),
2. the ideal model $I^1_{\text{des}}$ (with designated party $P_N$), with IT security (i.e. $\mathcal{A} = \mathcal{A}^{\text{all}}$, $\Sigma = \Sigma^{\text{eff}}$, and $\mathcal{D} = \mathcal{D}^{\text{all}}$),

from a complete and synchronous network of secure channels, and a broadcast channel, in the stand-alone setting, given that $t < n$ parties are corrupted.

We now show that a modified version of [GMW87] satisfies these requirements.

3.4.1 Using [GMW87] as $\pi^C$

[Cha89] proposes to use [CDG88] as the computational protocol $\pi^C$. However, this protocol is rather complicated and does not generalize readily as needed in later sections of this work. Instead, we propose to use a modified version of [GMW87]. The modification is done in two steps:

1. First, [Gol01] provides a variant of [GMW87] that is secure with abort (private and correct) for up to $t < n$ active adversaries thus satisfying property 1. of Lemma 3.2.
2. Second, we describe further modifications to obtain the desired designated party property (property 2. in Lemma 3.2).

3.4.2 Recap of the original protocol

We will give a short recapitulation of the protocol (for a full description refer to [Gol01]). The protocol is constructed in two steps. In a first step only passive adversaries are considered. The computation is done as follows:

**Situation:** Each party $P_i$ has input $x_i$. The goal is to compute $y = f(x_1, ..., x_n)$ where $f$ is given as a circuit over $GF(2)$ only containing addition and multiplication gates.

**Initialization:** Party $P_i$ XOR-shares $x_i$ among all parties.
3.4 Protocol $\pi^C$: Computing the output (stand-alone setting)

**Addition Gates:** Given a sharing of $a$ and $b$, calculate a sharing of $c = a + b$. To do this, each party sets the share of the sum to be the sum $c$ of the shares for $a$ and $b$: $c_i = a_i + b_i$.

**Multiplication Gates:** Given a sharing of $a$ and $b$, calculate a sharing of $c = a \cdot b$ such that $\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} a_i \cdot \sum_{i=1}^{n} b_i$. The key observation is that $\sum_{i=1}^{n} a_i \cdot \sum_{i=1}^{n} b_i = n \sum_{i=1}^{n} a_i b_i + \sum_{1 \leq i < j \leq n} (a_i + a_j) \cdot (b_i + b_j)$. This last equation can be evaluated in several 2-party computations, each using an oblivious transfer primitive.

**Output:** Each party sends its shares of the output to all other parties.

This protocol is secure against a passive adversary corrupting $t < n$ parties (proof in [Gol01]). To make it secure against active adversaries, one has to guarantee that the parties cannot deviate from the protocol. This is equivalent to asserting that all messages are computed correctly. For simplicity we assume that a broadcast channel is provided. Using this broadcast channel, a public-key infrastructure (PKI) can easily be generated as follows: Before the protocol is started, first each party privately computes a private-key and the corresponding public-key. Then it sends the public-key over the broadcast channel. This PKI is used to encrypt messages sent over the broadcast channel. Now the protocol proceeds as follows:

**Commitments:** Before sending a message, each party $P_i$ commits to its input $x_i$ using a commitment scheme $C_i$. For our purpose, it is crucial to note that commitment schemes for different parties do not have to be the same.

**Randomness:** General computations might include randomness. In order to make sure that all parties use truly random bits, this randomness has to be generated in an MPC. As long as this MPC is not aborted and there is at least one honest player, each party ends up with a random string and is committed to this random string. Basically, this is done using Blum Cointosses.

**Correct Messages:** The key observation to guarantee correct messages is that the next message a party has to send to another party is a deterministic function of the input, the random string (to both of which the player is committed), and the previously received messages (as could be seen on the broadcast channel). Thus when sending a message, the sender can give a zero-knowledge proof that the content of the message was computed correctly.

We claimed that the next message is always a deterministic function of the input, the random string and the previously received messages as could be seen on the broadcast channel. However, all messages are encrypted, the previous ones as well as the new one. Hence, the private and public keys have to be provided as input to the function as well. The public keys are known to everybody and can easily be included. In case of the private key we can assume without loss of generality that it is uniquely determined by the public key.

### 3.4.3 The modifications

The protocol in [Gol01], as described in the previous section, provides only computational security guarantees. In the IT scenario, the protocol is completely insecure. Now, we
modify this protocol in such a way that the requirements stated in Lemma 3.2 are fulfilled.

The above protocol is based on an oblivious transfer primitive, commitments and zero-knowledge proofs. For the original protocol it is not relevant which particular protocols are used to implement these primitives as only computational security is considered. For our purposes we have to use certain instantiations of these primitives, which provide IT privacy for the designated party $P_N$ and IT correctness, while leaving the computational privacy for all other honest parties unchanged.

For the oblivious transfer (OT) we can use the protocol from [Gol01] which IT protects the receiver. The original protocol makes no restriction as to who is the sender and who the receiver. Whenever $P_N$ is involved in an OT, we set the other party to be the sender. Thus $P_N$ is always on the receiver side and information-theoretically protected.

For the commitments we use the fact that each party can use a different commitment scheme. It is impossible to have a commitment scheme which is information-theoretically hiding and simultaneously information-theoretically binding without resorting to non-standard resources. Hence, we can provide information-theoretic security only for one of the two parties, and computational security for the other. The parties $P_{1:n}$ use a binding commitment scheme, whereas $P_N$ uses a hiding commitment scheme.

We use a similar approach for the zero-knowledge proofs. The statements we have to prove are all NP statements involving commitments. From [BCC88] we have two different notions of zero-knowledge proofs for such statements, depending on the type of the commitment scheme. The parties $P_{1:n}$ use a binding commitment scheme. Hence, according to [BCC88], they can give computationally zero-knowledge proofs which are information-theoretically correct. On the other hand, party $P_N$ uses a hiding commitment scheme and gives “proofs” which are perfectly zero-knowledge. However, in this case the prover could cheat when granted unlimited computational resources. Hence, the result of the protocol execution is not called an interactive proof, but only an interactive argument.

All the above primitives can be implemented given that enhanced trapdoor permutations exist. In the next section, given these modifications of [GMW87], we can prove Lemma 3.2, which claims the existence of a protocol $\pi^C$ with special security properties.

### 3.4.4 Proof of Lemma 3.2

We will now show that the modified version of [GMW87] described in the previous section can be used for protocol $\pi^C$ as it fulfills the requirements stated in Lemma 3.2 above.

**Security with abort in the computational setting.** This first property of Lemma 3.2 is already implied by the original protocol. Our modifications do not change the protocol, but only apply some restrictions as to what kind of primitives can be used in the different situations. Hence, the proof in [Gol01] remains valid. Especially note that agreement on abort is achieved using the broadcast channel: The only way to make a party abort is to send an incorrect message (one for which the zero-knowledge proof does not hold). However, since the message together with the proof is sent over the broadcast channel, this will be noted by all honest parties and they will all abort.
### 3.4 Protocol \( \pi^C \): Computing the output (stand-alone setting)

**Information theoretic privacy of the input of the designated party \( P_N \).** This constitutes the first part of the second requirement in Lemma 3.2. To show that the designated party’s privacy is information-theoretic, we have to go through the three primitives used in the protocol and show, that no information is distributed to other parties.

The first step is the commitment to the input. We let the designated party \( P_N \) use a commitment scheme that is information-theoretically hiding. Hence, even breaking the computational assumption does not reveal the committed value. Thus, during this step, no information is revealed by the designated party.

The next step in the protocol is sharing the input with the other parties, which is done using an XOR-sharing. Hence, as long as one party (i.e. \( P_N \)) is honest, the other shares give absolutely no information about the value. We modify this part later to provide a certain level of robustness. Then it still has to be guaranteed that no coalition of parties not containing \( P_N \) can learn anything about \( P_N \)’s input. Thus, the sharing does not violate the IT privacy of the designated party’s input.

Addition is done locally without \( P_N \) sending any messages. Consequently, no information can be leaked. For the multiplication we use the OT-primitive described in [Gol01, pp. 640–643], which works as follows: Let \( b_1, \ldots, b_k \) be the input of the sender and \( j \) the index of the bit the receiver selects. First, the sender selects an enhanced trapdoor permutation \( f \) together with a trapdoor \( f^{-1} \) and sends \( f \) to the receiver. Then, the receiver selects \( y_i, (i \neq j) \) and \( x_j \) at random, and computes \( y_j = f(x_j) \). Eventually, he sends \( y_1, \ldots, y_k \) to the sender who computes \( z_i = f^{-1}(y_i) \) and returns \( [b_1 \oplus B(z_1), \ldots, b_k \oplus B(z_k)] \) (where \( B \) is a hard core bit of \( f \)) to the receiver. Finally the receiver extracts \( b_j \) using \( x_j = z_j \).

With the designated party \( P_N \) always playing the role of the receiver and \( P_i, 1 \leq i \leq n \) the role of the sender, the only message \( P_N \) sends to \( P_i \) contains solely independent and identically distributed random values that are independent of the internal values of \( P_N \). Hence, this primitive is information-theoretically private for the receiver \( P_N \).

Finally we have to analyze the zero-knowledge proofs. As described above, we have the designated party \( P_N \) give perfectly zero-knowledge interactive arguments, again using hiding commitments. It is important to note that the hiding commitments used in the run of the interactive argument are different from the hiding commitments distributed during the input phase. The latter formulate the NP statement, whereas the former are used to prove this statement. So while being IT protected, the prover \( P_N \) could cheat when given unlimited computational resources. However, this does not affect our considerations as we want to provide information-theoretic security only for one party, whereas the other parties have to rely on the computational assumption not being broken. Hence, the zero-knowledge proofs leak no information about the designated party’s input.

The fact that messages are encrypted using a scheme that only provides computational security does not matter for the IT privacy of \( P_N \). We could always assume that all parties but the designated party \( P_N \) are corrupted. In that case the adversary can read all messages anyway and one could just as well send them in plain text. It follows from the discussion above, that even in that case, no information about the input of the designated party \( P_N \) is leaked to the adversary.
Information theoretic correctness of the output given that the designated party $P_N$ is honest. This constitutes the second part of the second requirement in Lemma 3.2. The correctness of the protocol relies on the fact that parties cannot send incorrect messages once they are IT binding committed to their input and random tape. This is guaranteed by zero-knowledge proofs of correct messages. The parties $P_{1:n}$ have to give proofs which are information-theoretically secure against cheating, since 1) they are based on the binding commitments distributed during the input phase (formulating the corresponding NP statement), and 2) also use binding commitments during the run of the zero-knowledge proof. Hence, even breaking the computational assumptions does not help any collusion of players $P_{1:t}$ to change the output because incorrect messages will always be detected.

Since these proofs are sent over the broadcast channel, inconsistencies are detected by all parties. This also implies agreement on abort: If one honest party receives a message with an incorrect proof, all honest parties receive the same message with the same incorrect proof. Consequently, all honest parties will abort.

This concludes our proof of Lemma 3.2. Now we have a protocol $\pi^C$ that provides computational security with abort and IT security with a designated party, for any number of corrupted parties, in the stand-alone setting. Together with the protocol $\pi^N$ used to simulate the designated party $P_N$, we can now formally specify the complete protocol and its security in the stand-alone setting. This is done in Section 3.6. Before, we discuss how the protocol $\pi^C$ can be realized in the UC setting.

### 3.5 Protocol $\pi^C$: Computing the output (UC setting)

In this section we revisit the second component of the complete protocol. So far we only considered protocols $\pi^C$ in the stand-alone setting. In this section we explore the possibility to use an universally composable protocol for the computation of the output. Our discussion is based on the protocol presented in [CLOS02], which basically implements the protocol from [GMW87] using universally composable primitives instead of the standard ones, based on the same resources (essentially a broadcast channel and an additional setup). In turn, this paper is based on [CF01], which discusses universally composable commitments.

However, for our purposes we cannot use an unmodified [CLOS02] protocol as it does not provide special protection for a designated party. Instead we have to apply the modifications of Section 3.4.3 and verify that the UC property is not affected. We do this component-wise. At this point the power of UC is very convenient: in case we need a different component with different properties we can just “plug it in”. The framework takes care that the further construction still works.

In contrast to the stand-alone setting, it is provably impossible to construct a UC commitment scheme in the plain model without a special setup [CF01]. Since all known MPC protocols in the UC setting without honest majority are based on commitments, some kind of setup has to be provided. In this work, all commitment schemes (and hence all corresponding MPC protocols) are proven to be secure in the common reference string (CRS) model. In the CRS-model all parties are given a common, public reference string
that is ideally chosen from a given distribution. This model was originally proposed in the context of non-interactive zero-knowledge proofs [BFM88] and since then has been proved useful in other cases as well.

We adapt Lemma 3.2 to this setting and describe the modifications to [CLOS02] that implicitly proof the following lemma:

**Lemma 3.3.** There is a protocol $\pi^C$ that implements

1. the ideal model $\mathcal{I}_{\text{abort}}^I$ (with abort), with computational security (i.e. $\mathcal{A} = \mathcal{A}^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $\mathcal{D} = \mathcal{D}^{\text{eff}}$),
2. the ideal model $\mathcal{I}_{\text{des}}^I$ (with designated party $P_N$), with IT security (i.e. $\mathcal{A} = \mathcal{A}^{\text{all}}$, $\Sigma = \Sigma^{\text{eff}}$, and $\mathcal{D} = \mathcal{D}^{\text{all}}$),

from a complete and synchronous network of secure channels, a broadcast channel, and a CRS-setup, in the UC setting, given that $t < n$ parties are corrupted.

In order to prove this lemma, we have to show that each component used in the construction fulfills the requirements in Lemma 3.3. We do this along the lines of [CLOS02] and start with the very basic primitives, working our way up to the complete MPC protocol.

### 3.5.1 Oblivious transfer

First we need a protocol realizing the oblivious transfer primitive $\mathcal{I}_{\text{OT}}$, which works as follows: For a fix $l$, the sender sends $l$ values $x_1, \ldots, x_l$ to $\mathcal{I}_{\text{OT}}$. Then, the receiver sends an index $i \in [l]$ to $\mathcal{I}_{\text{OT}}$ and obtains in return the value $x_i$. The sender does not learn the selected index $i$, and the receiver does not learn the other values $x_j, j \neq i$.

The protocol used in [CLOS02] is the same as used in [GMW87] since it is already UC secure (proof in [CLOS02, Section 4.1.1]). The same arguments as in Section 3.4.4 hold as to why this primitive fulfills the requirements of Lemma 3.3: Allowing the designated party $P_N$ to be always on the receiver side, protects its privacy with IT security. Thus, with the same modification we can use this protocol for our purpose.

### 3.5.2 Commitments

A basic primitive needed to provide security against active adversaries is the commitment functionality $\mathcal{I}_{\text{Com}}$, which proceeds as follows: During the commit phase, the sender sends a bit $b \in \{0, 1\}$ to $\mathcal{I}_{\text{Com}}$, which confirms the receipt to the set of receivers (in the two-party case this set contains only one party, in the multi-party case $n - 1$ parties). In the reveal phase, the sender causes $\mathcal{I}_{\text{Com}}$ to send $b$ to the set of receivers.

As [GMW87], [CLOS02] only aims for computational security. Therefore, commitments only have to be computationally binding and hiding. However, our modifications require IT binding commitments to be used by the parties $P_{1:n}$, and IT hiding commitments to be used by the designated party $P_N$. 
The commitment scheme described in [CF01] is IT binding and UC secure. However, Canetti and Fischlin do not provide an UC secure IT hiding commitment scheme. On the other hand, [DN02] describes a construction to obtain both UC secure IT hiding and UC secure IT binding commitment schemes in the CRS-model. We can use these commitments without further modifications.

### 3.5.3 Zero-knowledge proofs

Another primitive introduced to provide security against active adversaries is the functionality for zero-knowledge proofs $\mathcal{I}_{ZK}$, which is parameterized with a relation $R$. Here, the prover sends a pair $(x, w)$ to $\mathcal{I}_{ZK}$. If $R(x, w) = 1$, then $\mathcal{I}_{ZK}$ confirms to the verifier, that the prover knows a witness $w$ with respect to $x$ and $R$, without revealing $w$.

The protocol in [CF01, Section 5] uses the well-known protocol based on Hamiltonian Cycles. From their paper: “In the $\mathcal{I}_{Com}$-hybrid model the protocol securely realizes $\mathcal{I}_{ZK}$ without any computational assumptions.” This is a very important observation for our goal. It implies that when using an IT binding commitment scheme, we obtain zero-knowledge proofs which are IT correct. On the other hand, when using an IT hiding commitment scheme, we obtain perfect zero-knowledge interactive arguments. This exactly meets the requirements in Lemma 3.3. We can use this protocol, requiring the participating parties to use their corresponding commitment schemes (IT hiding for the designated party $P_N$, IT binding for the other parties).

### 3.5.4 Commit-and-prove

Instead of directly replacing the commitment and zero-knowledge functionalities, [CLOS02] introduces a new primitive called commit-and-prove $\mathcal{I}_{CP}$, which again is parameterized with a relation $R$. During the commit phase, the committer sends a witness $w$ to $\mathcal{I}_{CP}$, which is appended to a list $\bar{w}$. $\mathcal{I}_{CP}$ confirms the receipt to the set of verifiers. This phase can be repeated several times. During the prove phase, the committer sends a statement $x$ to $\mathcal{I}_{CP}$. If $R(x, \bar{w}) = 1$, then $\mathcal{I}_{CP}$ confirms to the verifiers, that the committer knows witnesses $\bar{w}$ with respect to $x$ and $R$, without revealing $\bar{w}$.

This extension is due to the fact that commitments in [GMW87] are never actually opened but only used in zero-knowledge proofs. For the construction of the protocol realizing $\mathcal{I}_{CP}$, [CLOS02] only uses the primitives for zero-knowledge proofs $\mathcal{I}_{ZK}$, and a standard commitment scheme. As was the case with zero-knowledge proofs and commitments, we need two different realization for this primitive, depending on who is the sender.

When the designated party $P_N$ is the sender, we need to IT protect the senders privacy. This can be achieved by using IT UC hiding commitments when realizing $\mathcal{I}_{ZK}$, and plain IT hiding commitments, e.g. Pederson-Commitments. On the other hand, when a party $P_i, i \in [n]$ is the sender, we have to IT guarantee the correctness. This is done using IT UC binding commitments for the realization of $\mathcal{I}_{ZK}$, and plain IT binding commitments, e.g. El-Gamal-Commitments.

Applying the protocol in this way, we can implement the commit-and-prove functionality $\mathcal{I}_{CP}$ with the required security guarantees as stated in Lemma 3.3.
3.5 Protocol $\pi^C$: Computing the output (UC setting)

3.5.5 The general construction for passive adversaries

Given the UC primitives above, we now have to provide a protocol that combines these primitives and enables the participating parties to perform a MPC. In [CLOS02] this is done along the lines of [GMW87]: A first protocol only provides security against passive adversaries. This protocol is later compiled into a protocol secure against active adversaries. Here, we focus on the first protocol.

The general construction of the protocol in [CLOS02, Section 4] is identical to the one in [GMW87]. Given the OT primitive $I_{OT}$ the protocol makes no further computational assumptions ([CLOS02, Section 4.2]). Hence, when applying the same modifications as in the stand-alone setting in Section 3.4.3 of this work, we obtain the required security guarantees (especially IT privacy for the designated party $P_N$).

Another consequence of this fact is that our modifications in Sections 6 (fairness) and 7 (robustness) work in the UC model just as well as they do in a stand-alone model.

3.5.6 Two-party computation with an active adversary

Given the protocol which is secure against passive adversaries from the previous section, we now transform it into a two-party protocol, secure against active adversaries. Like [GMW87], [CLOS02] uses a compiler for this purpose ([CLOS02, Section 8.1, p. 61]). This compiler makes no additional cryptographic assumptions and is only based on the commit-and-prove functionality $I_{CP}$. Furthermore, it is possible to use separate functionalities $I_{CP}$ depending on who is the sender. From these observations and the discussion above concerning the UC primitives, it follows directly that our modifications from Section 3.4.3 apply. Thus, we obtain a two-party protocol fulfilling the security requirements of Lemma 3.3.

3.5.7 Multi-party computation with active adversaries

The generalization to the multi-party case is relatively straightforward. However, there are some subtleties we have to look at. First, we note that the construction in the semi-honest case ([CLOS02, Section 9.1]) is again identical to [GMW87] and no further computational assumptions are made. Hence, using the OT protocol in the appropriate direction results in a construction with the desired security properties.

As a next step we have to generalize the functionalities described above to the multi-party scenario ([CLOS02, Section 9.3]). In contrast to [CLOS02] we only consider static adversaries. This simplifies the following discussion: Non-interactive two-party protocols can easily be extended to the multi-party case by having the sender broadcast the messages to all parties, instead of sending them to a single party. As no interaction takes place, the messages are the same ones as in the two-party case. Hence, the corresponding proofs are almost identical.

The multi-party variant of the commitment functionality, $I_{Com,1:M}$, can be realized in the same way as the two-party functionality. Both protocols for IT binding and IT hiding
commitments are non-interactive and following the argument above can be used without modification.

Next we look at the one-to-many zero-knowledge functionality $I_{ZK,1:M}$. From [CLOS02]: “The protocol for realizing $I_{ZK,1:M}$ works by having the prover separately prove the statement in question to all parties. The protocol used in each of these pairwise proofs is exactly the two-party protocol of [CF01], with the exception that the messages of each proof are broadcast. [...] Note that other than the use of $I_{Com,1:M}$, no cryptographic primitives are used.” This is exactly what we need and implies the required security properties of Lemma 3.3.

Now we describe how to realize the one-to-many commit-and-prove functionality $I_{CP,1:M}$. [CLOS02, Section 9.3] specifies a protocol which is secure against adaptive adversaries what we do not need. Instead, we can adapt the static two-party variant described in [CLOS02, Section 7.1]. It is non-interactive, thus following the argument above it can be used straight-away.

Finally, the compiler transforming the protocol into one that is secure against active adversaries is solely based on the one-to-many commit-and-prove functionality $I_{CP,1:M}$ and makes no additional computational assumptions. As in the two-party case, we can use separate functionalities $I_{CP,1:M}$ depending on who is the sender, in order to achieve the desired security properties of Lemma 3.3.

This concludes our proof of Lemma 3.3. The modifications discussed in the previous sections result in a protocol $\pi^C$ that provides computational security with abort and IT security with a designated party, for any number of corrupted parties, in the UC setting. Together with the UC protocol $\pi^N$ used to simulate the designated party $P_N$, we can now formally specify the complete protocol and its security in the UC setting, as well.

### 3.6 The complete protocol $\Pi_f$

In the previous sections we discussed how the subprotocols $\pi^C$ and $\pi^N$ can be realized ($\pi^N$ only in the UC setting, $\pi^C$ in both the stand-alone and the UC setting). We will now show how these two subprotocols can be combined into the complete protocol $\Pi_f$, which combines computational and IT security. Let $\pi^C_i$, $(i = 1, \ldots, n)$ denote a standard protocol machine for protocol $\pi^C$, $\pi^C_N$ denote the protocol machine for the designated party in protocol $\pi^C$, and $\pi^N_i$ a protocol machine for protocol $\pi^N$.

**Initial Situation:** Each party $P_i$, $1 \leq i \leq n$ has an input $x_i$.

**Goal:** Each party $P_i$ holds the output $y = f(x_1, \ldots, x_n)$ or the empty value $\perp$.

1. $\forall P_i : x_i^N \leftarrow_R \{0, 1\}^m, \quad x_i^C := x_i \oplus x_i^N$

2. Using a hiding commitment scheme $C(\cdot)$, each party $P_i$ computes $[C(x_i^N), o_i] = \text{COMMIT}(x_i^N)$

3. $\forall i : P_i$ starts a protocol machine $\pi^N_i$ on input $(x_i^N, o_i)$ (protocol $\pi^N$ emulates $\pi^C_N$ with input $\{x_i^N, o_i\}_{i \in [n]}$ IT securely as stated in Lemma 3.1)
3.7 Security of $\Pi_f$

With the protocol description $\Pi_f$ in the previous section, we now have to formalize what kind of security exactly can be achieved. [Cha89] claims that a protocol can be constructed along the lines of Section 3.6 that achieves security with abort for $t < \frac{n}{2}$ corrupted parties in the information-theoretic scenario, and for $t < n$ corrupted parties in the computational scenario.

We state this claim in two theorems for UC and stand-alone security. In a slight abuse of notation, we use the same identifier $\Pi_f$ for the protocol in both cases, even though there are differences, mainly concerning the subprotocol $\pi^C$. In the stand-alone setting

These checks have to be computed by $\pi^C$: The goal of the extra commitments is to guarantee that the correct values $x^N_i$ are used for the computation even if $\pi^N$ is adversarially controlled (for $t > n/2$ corrupted parties).

Figure 7: An illustration of the full protocol.

4. $\forall i : P_i$ starts a protocol machine $\pi^C_i$ on input $(x^C_i, C(x^N_i))$

The broadcast interface of each protocol machine $\pi^C_i$ is connected to a local machine $\pi^N_i$ implementing a majority vote (see Def. 3.2). The interface for the private channel with the designated party $P_N$ is connected locally to the protocol machine $\pi^N$. Protocol $\pi^C$ is designed to implement the ideal functionality ($I_{\text{abort}}$ with computational security, and $I_{\text{des}}$ with IT security and designated party $P_N$) that performs the following steps (existence in the stand alone setting by Lemma 3.2, in the UC setting by Lemma 3.3):

(a) Input check: $\forall i : \text{OPEN}(c_i, o_i) = x^N_i$ (abort on fail: $y = \bot$)\(^1\)

(b) Recover input: $\forall i : x_i = x^N_i \oplus x^C_i$

(c) Computation and output (unless aborted): $y = f(x_1, \ldots, x_n)$

The protocol is illustrated in Figure 7.
4 The proof of security in the UC setting

we use the subprotocol that exists according to Lemma 3.2. The realization of this protocol is based on [GMW87]. On the other hand, in the UC setting we use the subprotocol that exists according to Lemma 3.3, which is based on [CLOS02]. However, this is only a minor adaptation, and the overall protocol implements corresponding security guarantees.

Lemma 3.4. **Stand-alone setting.**

\( \Pi_f \) implements

1. the ideal model \( \mathcal{I}_f^{abort} \) (with agreement on abort), with computational security (i.e. \( A = A^{eff}, \Sigma = \Sigma^{eff}, \) and \( D = D^{eff} \)), given that \( t < n \) parties are corrupted, and

2. the ideal model \( \mathcal{I}_f^{abort} \) (with agreement on abort), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( t < n/2 \) parties are corrupted,

from a complete and synchronous network of secure channels, and a broadcast channel, in the stand-alone setting.

And analogously for the UC setting:

Lemma 3.5. **UC Setting.**

\( \Pi_f \) implements

1. the ideal model \( \mathcal{I}_f^{abort} \) (with agreement on abort), with computational security (i.e. \( A = A^{eff}, \Sigma = \Sigma^{eff}, \) and \( D = D^{eff} \)), given that \( t < n \) parties are corrupted, and

2. the ideal model \( \mathcal{I}_f^{abort} \) (with agreement on abort), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( t < n/2 \) parties are corrupted,

from a complete and synchronous network of secure channels, a broadcast channel, and a \textit{CRS}-setup in the UC setting.

Now we have a formal protocol description and a precise claim of security. In Section 4 we first give a proof for the UC Theorem 3.5, before we extend it to prove the stand-alone Theorem 3.4 in Section 5.

4 The proof of security in the UC setting

In the previous sections we have seen how each subprotocol \( \pi^N \) and \( \pi^C \) can be realized in the UC setting with the required security properties, and how the complete protocol is constructed based on these subprotocols. In this section we give a proof for Theorem 3.5, which states the security in the UC setting. That means, we use the UC protocol \( \pi^C \) which exists according to Lemma 3.3, and the UC protocol \( \pi^N \) which exists according to Lemma 3.1. For this model we can use the notation as introduced in [Mau06], and Section 2.1.3 in this work, respectively. In addition to the notation used so far we use \( R \) to denote the underlying resources, i.e. in this case a complete and synchronous network of secure channels, a broadcast channel, and a \textit{CRS}-setup.

Now, we first give an overview of the goal, i.e. what we have to do in the proof. Then we provide the actual proofs for the IT and the computational scenario separately.
4.1 The goal

To proof the security of the protocol \( \Pi_f \) we have to provide a simulator \( \sigma^{\Pi_f} \) that makes the ideal system with abort \( I_f^{\text{abort}} \) indistinguishable from the real protocol execution, i.e.

\[
\exists \sigma^{\Pi_f} \in \Sigma : \text{adv}^D(\sigma^{\Pi_f}(I_f^{\text{abort}}), \Pi_{\mathcal{H}}(R)) \leq \epsilon(\kappa)
\]

where \( \Sigma \) is the class of simulators, \( D \) the class of distinguishers, \( \mathcal{H} \) the set of honest parties and \( \mathcal{A} \) the set of corrupted parties. \( \Pi_{\mathcal{H}}(R) \) denotes the real system where the interfaces for the honest parties are connected to correct protocol machines \( \Pi \), and the interfaces for the corrupted parties are open for adversarial interaction. We separately prove the computational and information-theoretic case. However, the basic task is the same and illustrated in Figure 8.

![Figure 8: The Goal: Prove \( \Pi_f \) indistinguishable from the ideal model \( I_f^{\text{abort}} \) with a suitable simulator \( \sigma^{\Pi_f} \). (The broadcast channel belonging to \( \pi^N \) is not shown for simplicity.)](image)

Usually, proofs in the UC setting exploit the modularity of the components, using a composition theorem and hybrid systems: In a sequence of systems \( S_1, \ldots, S_k \), it is enough to show that each system \( S_i \) is indistinguishable from \( S_{i+1} \), using a simulator \( \sigma^{i,i+1} \). Then it follows from the composition theorem that there is a simulator \( \sigma^{1,k} \) such that the systems \( S_1 \) and \( S_k \) are indistinguishable. There is no need to actually construct this simulator \( \sigma^{1-k} \). However, in the following proof we do construct this complete simulator, since it allows us to easily derive a simulator for the stand-alone setting in Section 5.

4.2 Proof for the information-theoretic case in the UC setting

In this section we prove the second property of Theorem 3.5, claiming that the protocol \( \Pi_f \) implements the ideal model \( I_f^{\text{abort}} \) with abort and agreement on abort (without obtaining
fairness or robustness) with information-theoretic security, given that only a minority of the parties is corrupted. In contrast to Theorem 3.4, Theorem 3.5 treats the universally composable setting as introduced in Section 2.1.3. Since we are in the IT setting, we have $\mathcal{A} = \mathcal{A}^{all}$, $\Sigma = \Sigma^{all}$, and $\mathcal{D} = \mathcal{D}^{all}$. However, as the simulators $\sigma^C$ and $\sigma^N$ for the subprotocols $\pi^C$ and $\pi^N$ are all in $\Sigma^{eff}$ as demanded in Lemmas 3.3 and 3.1, the resulting simulator $\sigma^\Pi_f$ is as well.

The protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Additionally, since we are in the UC setting, we need a setup as demanded by the subprotocol $\pi^C$ in Section 3.5.

The first step is to replace the IT protocol $\pi^N$ with the ideal system $I^{sec}_N$ and the corresponding simulator $\sigma^N$. In a second step, we replace the modified broadcast channel $BC'$ together with the ideal system $I^{sec}_N$ with a normal broadcast $BC$ and the ideal system $I^{sec}_N$. Here we introduce the simulator $\sigma^{BC}$. Then, we replace the computational protocol $\pi^C$ with the ideal system $I^{des}_C$ and the corresponding simulator $\sigma^C$. Finally, based on the replacements, we construct the complete simulator $\sigma^\Pi_f$ as required for the proof.

4.2.1 Using the simulator for protocol $\pi^N$

First, we consider the protocol $\pi^N$, simulating the designated party $P_N$. From the security of protocol $\pi^N$ as stated in Lemma 3.1 and the definition of universally composable, we have a UC simulator $\sigma^N$ for which

$$\text{adv}^D(\sigma^N(I^{sec}_N), \pi^N(R)) \leq \epsilon(\kappa)$$

holds. It follows that we can indistinguishably replace the protocol $\pi^N$ with the ideal model $I^{sec}_N$ and the simulator $\sigma^N$. The resulting systems are IT-indistinguishable. This is illustrated in Figure 9.

![Figure 9: Replacing the protocol $\pi^N$ with the ideal model and simulator in the UC setting.](image)

This property can be applied to the complete protocol $\Pi_f$, which is illustrated in Figure 10. To make this substitution work we have to declare everything outside the system (i.e. outside the protocol machines $\pi^N_i$) to be a new distinguisher $D' = D \circ D^{aux}$. This is due
4.2 Proof for the information-theoretic case in the UC setting

Figure 10: Replacing the protocol $\pi^N$ in the full protocol $\Pi_f$ (UC setting, information-theoretic scenario).

to the fact that the definition only provides for the case where everything but the system under consideration is regarded as environment, which is completely controlled by the distinguisher. However, since the resulting systems must be indistinguishable for every distinguisher, they are also indistinguishable for $D'$.

In the resulting system, $I_{sec}^{N'}$ securely implements the protocol machine $\pi^C_N$ with one modification: It takes an input, consisting of $n$ components $(x_i^N, o_i), i \in [n]$, and participates correctly in protocol $\pi^C$ as the designated party. However, whenever it needs to send a broadcast message, it sends $n$ identical copies of the message over the channels leading to $BC'$ (compare with Section 3.3.4 about the simulated party $P_N$ and broadcast messages). When other protocol machines $\pi_i^C$ send broadcast messages, the ideal system $I_{sec}^{N'}$ receives $n$ identical copies of the message from the modified broadcast channel $BC'$ (this follows from the definition of $BC'$ in Def. 3.1). The ideal system $I_{sec}^{N'}$ selects the first one and discards the rest.

As result of this replacement, instead of a protocol, we now have an ideal functionality $I_{sec}^{N'}$ behaving similar (but not equal) to the protocol machine $\pi^C_N$. Usually, the simulator $\sigma^N$ could be discarded in the following discussion. However, it is part of the complete simulator $\sigma^{\Pi_f}$ (which we construct with an eye on the stand-alone setting), so we leave it in place.

4.2.2 Replacing the broadcast channel

After the replacement in the previous section, we have an ideal functionality $I_{sec}^{N'}$, adapted to the modified broadcast channel $BC'$, and local machines $\pi_i^r$ implementing a majority
vote. All these components are different from the definition of subprotocol $\pi^C$. Hence, in the next step, we have to replace the broadcast channel $BC'$ together with the local machines $\pi^v_i$ and $I_{NC}'$ with a system that allows us to replace the protocol $\pi^C$ (as we will do in the following step).

The new system consists of three components: A standard broadcast channel $BC$, the ideal system $I_{NC}'$ implementing the unmodified protocol machine $\pi^C_N$, and a simulator $\sigma^{BC}$ which is connected to the interfaces that belong to the corrupted parties. This is shown in Figure 11.

On the left hand side in Figure 11, the channels running from $I_{NC}'$ to $BC'$ represent the channels from the honest protocol machines $\pi^C_N$ ($i \in \mathcal{F}$) to $BC'$; the channels running from $\sigma^N$ to $BC'$ the ones belonging to the corrupted protocol machines $\pi^C_N$ ($i \in \mathcal{A}$). According to the procedure described in Section 3.3.4 all the messages on the “honest channels” are identical (let this message be $m$). This is due to the fact that in the current setting only a minority of the parties is corrupted and that, as a consequence, the protocol $\pi^N$ provides full security, especially correctness. Thus, each honest protocol machine $\pi^C_N$ obtains the same output to be broadcasted. On the other hand, the messages over the “corrupted channels” are arbitrary. As mentioned above, we have an honest majority. As a consequence, the local machine $\pi^v_i$ implementing the majority vote receives a set of messages in which a majority is equal to $m$. Hence, the majority vote will always output this message $m$, independently of what the corrupted parties might send.

On the right hand side in Figure 11, $I_{NC}'$ behaves like the honest protocol machine $\pi^C_N$ for designated party $P_N$ and will thus send the correct message $m$ (according to protocol $\pi^N$) to the broadcast channel $BC'$ (which forwards this message to the other parties participating in protocol $\pi^C$). The consequence is that the broadcasts received by the honest parties in $\pi^C$ do not change.

There are 3 different message flows, the simulator $\sigma^{BC}$ has to handle (see Section 3.3). It does so in the following way:

$P_N \rightarrow \pi^C_i$. $I_{NC}'$ behaves like the honest protocol machine $\pi^C_N$ and broadcasts the correct message $m$ according to protocol $\pi^C$, which the simulator receives over its connection to $BC$. $t$ copies of this message are forwarded over channel (2) (see Figure 11) to $\sigma^N$. Over channel (1) (same figure) from $\sigma^N$, $\sigma^{BC}$ receives the broadcasts $\{bc_i\}_{i \in \mathcal{A}}$ belonging to the corrupted parties. Now, $\sigma^{BC}$ sets $\forall i \in \mathcal{F}$ : $bc_i := m$ and sends $\{bc_i\}_{i \in \mathcal{F}}$ over the channel to the distinguisher.

$\pi^C_i, i \in \mathcal{F} \rightarrow P_N$. The simulator receives the broadcasted message from $BC$ and forwards it to both the distinguisher and simulator $\sigma^N$ over channel (1) (Figure 11).

$\pi^C_i, i \in \mathcal{A} \rightarrow P_N$. The simulator receives the message to be broadcasted from the distinguisher and forwards it to both $BC$ and simulator $\sigma^N$ over channel (1) (Figure 11).

It is straightforward to see that the two systems are perfectly indistinguishable, under the assumption of an honest majority $n - t > n/2$.

After this replacement, we have a normal broadcast channel $BC$ and an unmodified protocol machine $\pi^C_N$. Furthermore, there are no more local machines $\pi^v$. Now it is
4.2 Proof for the information-theoretic case in the UC setting

possible to consider the protocol $\pi^C$ and perform a similar replacement as for protocol $\pi^N$, since all components comply with the definition of protocol $\pi^C$. Again, we leave the simulator $\sigma^{BC}$ in place since it is part of the complete simulator $\sigma^{\Pi_f}$.

4.2.3 Using the simulator for protocol $\pi^C$

The replacements in the previous sections result in a system that applies the protocol $\pi^C$ according to its definition, i.e. there is a normal broadcast channel $BC$ and correct protocol machines for all honest parties, including the designated party $P_N$.

In the next step we take the new system and replace the protocol $\pi^C$ with the ideal model $I_{C_{des}}$ with designated party $P_N$. From the security of $\pi^C$ as stated in Lemma 3.3 we have an UC simulator $\sigma^C$ for which

$$\text{adv}^D(\sigma^C_{N}(I_{C_{des}}), \pi^C(R)) \leq \epsilon(\kappa)$$

holds, given that the designated party $P_N$ is not corrupted ($P_N \notin \mathcal{A}$), and considering the class of all distinguishers ($D = D^{all}$). The corresponding illustration is similar to the one in Figure 9, just replace $N$ with $C$. The protocol $\pi^C$ requires a CRS-setup that could be illustrated with a box that is connected to all entities. For the sake of simplicity, we omit the setup in following discussion, since it would only detract from the main focus of the proof.

Again we have to redefine what the distinguisher exactly comprises in order to make the simulation work. As in the previous case, we consider everything but the protocol machines $\pi^C$ and the broadcast channel $BC$ to be the environment. Then, we substitute the remaining system $\pi^C_S(R)$ with $\sigma^C_S(I_{C_{des}})$. This process is illustrated in Figure 12.

Figure 11: Replacing $BC$ in the full protocol $\Pi_f$ (UC setting, information-theoretic scenario).
4.2 Proof for the information-theoretic case in the UC setting

As explained above, \( I_N^{\sec} \) acts as the designated protocol machine \( \pi_N^C \) and hence disappears just like the other honest protocol machines. In the resulting system, \( I_C^{\des} \) takes two different kinds of input. On one hand, it takes \( n \) inputs of the form \((x^C_i, c_i)\). On the other hand, it takes one input (consisting of \( n \) parts) of the form \((x^N_i, o_i)\) \(i \in [n]\), which belongs to the designated party \( P_N \). Then, \( I_C^{\des} \) computes step 4 of the complete protocol \( \Pi_f \) as described in Section 3.6, i.e. performs a consistency check, recovers the original input and computes the output of \( f \).

With this last replacement we are already very close to the ideal model that implements the complete protocol \( \Pi_f \). However, the input is still split into two parts, and additionally contains a commitment. It is the task of the complete simulator \( \sigma^f \), which we introduce in the next section, to take care of these message flows.

4.2.4 Assembling the complete simulator \( \sigma^{\Pi_f} \) for protocol \( \Pi_f \)

This is how far the results of Lemma 3.1 (security of \( \pi^N \) in the UC setting) and Lemma 3.3 (security of \( \pi^C \) in the UC setting) brought us. So far we only substituted subsystems to facilitate the task of constructing the complete UC simulator \( \sigma^{\Pi_f} \) for protocol \( \Pi_f \). In this last step we cannot use any postulated simulators but have to construct the simulator \( \sigma^f \) by ourselves and argue why it makes the ideal functionality \( \mathcal{I}_f^{\text{abort}} \) indistinguishable from \( \pi_f(\mathcal{T}_C^{\des}) \).

After the last replacement, the ideal functionality \( \mathcal{I}_C^{\des} \) still takes two different kinds of input \((x^N_i, o_i)\) and \((x^C_i, c_i)\), which additionally are correlated with a commitment. However, in the end, the ideal functionality \( \mathcal{I}_f^{\text{abort}} \) only takes one kind of input \( x_i \), and neither

---

**Figure 12:** Replacing \( \pi^C \) in the full protocol \( \Pi_f \) (UC setting, information-theoretic scenario).

As explained above, \( I_N^{\sec} \) acts as the designated protocol machine \( \pi_N^C \) and hence disappears just like the other honest protocol machines. In the resulting system, \( I_C^{\des} \) takes two different kinds of input. On one hand, it takes \( n \) inputs of the form \((x^C_i, c_i)\). On the other hand, it takes one input (consisting of \( n \) parts) of the form \((x^N_i, o_i)\) \(i \in [n]\), which belongs to the designated party \( P_N \). Then, \( I_C^{\des} \) computes step 4 of the complete protocol \( \Pi_f \) as described in Section 3.6, i.e. performs a consistency check, recovers the original input and computes the output of \( f \).

With this last replacement we are already very close to the ideal model that implements the complete protocol \( \Pi_f \). However, the input is still split into two parts, and additionally contains a commitment. It is the task of the complete simulator \( \sigma^f \), which we introduce in the next section, to take care of these message flows.

4.2.4 Assembling the complete simulator \( \sigma^{\Pi_f} \) for protocol \( \Pi_f \)

This is how far the results of Lemma 3.1 (security of \( \pi^N \) in the UC setting) and Lemma 3.3 (security of \( \pi^C \) in the UC setting) brought us. So far we only substituted subsystems to facilitate the task of constructing the complete UC simulator \( \sigma^{\Pi_f} \) for protocol \( \Pi_f \). In this last step we cannot use any postulated simulators but have to construct the simulator \( \sigma^f \) by ourselves and argue why it makes the ideal functionality \( \mathcal{I}_f^{\text{abort}} \) indistinguishable from \( \pi_f(\mathcal{T}_C^{\des}) \).

After the last replacement, the ideal functionality \( \mathcal{I}_C^{\des} \) still takes two different kinds of input \((x^N_i, o_i)\) and \((x^C_i, c_i)\), which additionally are correlated with a commitment. However, in the end, the ideal functionality \( \mathcal{I}_f^{\text{abort}} \) only takes one kind of input \( x_i \), and neither
needs to perform any consistency checks, nor does it have to recover the input \(x_i\). The simulator \(\sigma^f\) has to take care of this. This is illustrated in Figure 13.

The simulator \(\sigma^f\) will only be connected to the interfaces belonging to the corrupted parties. The inputs belonging to the honest parties are given directly to the ideal functionality \(I_{\text{abort}}^f\), and we do not have to worry about that. So, the simulator \(\sigma^f\) has two direct connections to the environment. For a better understanding, we denote the interface in Figure 13 to simulator \(\sigma^N\) by C1, and the interface to simulator \(\sigma^C\) by C2. The simulator has to produce the correct distributions on these two interfaces and the interface to the ideal functionality \(I_{\text{abort}}^f\), in order to render \(\sigma^f(I_{\text{abort}}^f)\) indistinguishable from \(\pi_H(I_{\text{des}}^C)\).

First, over interface C1, the simulator \(\sigma^f\) only receives the input of the corrupted parties intended for \(I^{\text{sec}}_{\text{N}}\), which is of the form \(\{x^N_i, o_i\}_{i \in A}\). Thus, perfect indistinguishability is maintained with respect to this step. However, over interface C2, the environment \(D' = D \circ D^{\text{aux}}\) expects the input of the honest parties intended for the ideal functionality \(I^{\text{des}}_C\), which is of the form \(\{x^C_i, C(x^N_i)\}_{i \in B}\). This information is not available and has to be simulated by \(\sigma^f\) correctly.

The simulation is depicted in Figure 14. The simulator \(\sigma^f\) performs the following steps:

1. Receive output \(\{x^N_i, o_i\}_{i \in A}\) over interface C1, intended for \(I^{\text{sec}}_{\text{N}}\).
2. For \(i \in B\), chose \(\tilde{x}^C_i\) and \(\tilde{x}^N_i\) at random.
3. For these \(\tilde{x}^N_i\), compute the commitments \(C(\tilde{x}^N_i)\).
4. Give \(\{\tilde{x}^C_i, C(\tilde{x}^N_i)\}_{i \in B}\) as output over C2.

Figure 13: Replacing \(I^{\text{des}}_C\) and the protocol machines \(\pi_i\) in the full protocol \(\Pi_f\) (UC setting, information-theoretic scenario).
5. Receive input \( \{x_i^C, c_i\}_{i \in \mathcal{A}} \) over C2, intended for \( \mathcal{I}_{C}^{\text{des}} \).

6. Calculate the XOR \( x_i = x_i^C \oplus x_i^N \) for \( i \in \mathcal{A} \) and provide the result as input for the corrupted parties to the ideal functionality.

7. Upon receiving the result \( y \), perform the check: \( \forall i \in \mathcal{A} : \text{OPEN}(c_i, o_i) \equiv x_i^N \) (Correct opening information)
   - [holds] \( y \) is forwarded to C2 and the abort flag is forwarded from C2 to the ideal functionality \( \mathcal{I}_f^{\text{abort}} \).
   - [fails] \( \bot \) is forwarded to C2 and the ideal functionality \( \mathcal{I}_f^{\text{abort}} \) is aborted.

Since we are currently treating the information-theoretic scenario, the statement in the check does not imply that \( c_i \) actually “contains” \( x_i^N \). The commitments are only computationally binding. Hence, the adversary could have “changed” the content. However, this does not affect the protocol and can be seen as a change in the input \( x_i \) provided by the adversary.

In the execution of \( \pi_{\mathcal{H}}(\mathcal{I}_{C}^{\text{des}}) \), party \( P_i \) first splits its input into \( x_i = x_i^C \oplus x_i^N \) and computes the commitment \( c_i = C(x_i^N) \) together with the corresponding opening information \( o_i \). This is done locally without interaction. Then, party \( P_i \) provides \( \{x_i^N, o_i\} \) as input to protocol \( \pi^N \). The simulation in \( \sigma^f \) imitates this input phase in step 1: The protocol \( \pi^N \)
provides full security. Hence, the environment has to provide the input belonging to the corrupted parties over C1, without obtaining any information about the input of honest parties.

After that, party $P_i$ provides $x_i^C$ together with the commitment $c_i$ as input to the ideal functionality $I_C^{des}$. Again, the simulation in $\sigma^f$ imitates this input phase in steps 2-5: The ideal functionality $I_C^{des}$ does not guarantee privacy for any honest party but for the designated party $P_N$. Hence, the adversary could obtain the corresponding input (in other words, the environment expects these values over C2). The simulator $\sigma^f$ has to fake these values. Due to the argument below this can be achieved with information-theoretic indistinguishability. Still, the ideal functionality $I_C^{des}$ guarantees correctness, and the environment has to provide the input belonging to the corrupted parties over C2 (step 5).

Now, the ideal functionality $I_C^{des}$ is started: It first checks the input (simulated in step 7), and then computes the output (simulated in step 6). The change in the order of the last two steps does not affect the result. All that matters is that the adversary receives the output if and only if the check succeeds. This holds in the real as well as in the ideal model.

It is easy to see that the simulation in Figure 14 works, i.e. it has the right distribution on the interfaces to the environment:

- Instead of the correct values $\{x_i^C, C(x_i^N)\}_{i \in \mathcal{H}}$, the simulator $\sigma^f$ provides random values $\{\tilde{x}_i^C, C(\tilde{x}_i^N)\}_{i \in \mathcal{H}}$ to the environment over C2. However, since the $\{x_i^N\}_{i \in \mathcal{H}}$ are independent (mutually and from the $x_i$’s) and uniformly distributed (see step 1 in the protocol description in Section 3.6), the values $x_i^C = x_i \oplus x_i^N, i \in \mathcal{H}$ are as well. And so are the $\{\tilde{x}_i^C\}_{i \in \mathcal{H}}$.

- Furthermore, the commitments $\{C(\tilde{x}_i^N)\}_{i \in \mathcal{H}}$ are hiding commitments. Hence, they are statistically independent of their content (and thereby also of the $\tilde{x}_i^C$’s and real inputs $x_i$). As a consequence, there is no difference between the real system and the simulation.

This shows that the distributions on all interfaces of simulator $\sigma^f$ connected to the ideal functionality $I_f^{abort}$ are information-theoretically indistinguishable from the distributions on the interfaces of the ideal functionality $I_C^{des}$ connected to protocol machines $\pi_i$. In other words, for all distinguishers, the left hand side and the right hand side in Figure 13 are information-theoretically indistinguishable.

Hence, the construction $\sigma^{\Pi_f}$ incorporating all simulators $\sigma^N, \sigma^C, \sigma^{BC}$, and $\sigma^f$ as shown in Figure 14 acts as a simulator for the ideal functionality $I_f^{abort}$ against the complete protocol $\Pi_f$. Thus it fulfills the requirement

$$\text{adv}^D(\sigma^{\Pi_f}(I_f^{abort}), \Pi_{\mathcal{H}}(R)) \leq \epsilon(\kappa)$$

for the class of all distinguishers $D = D^{all}$, thereby proving the second part of Theorem 3.5.

Footnote: Here we use the fact that the input of the designated party $P_N$ remains private due to the specification of the ideal model $I_C^{des}$. Otherwise the values would not be mutually independent.
4.3 Proof for the computational case in the UC setting

In this section we prove the first property of Theorem 3.5, claiming that the protocol $\Pi_f$ implements the ideal model $I_{\text{abort}}$ with abort and agreement on abort (without obtaining fairness or robustness) with computational security, for an arbitrary number of corrupted parties. In contrast to Theorem 3.4, Theorem 3.5 treats the universally composable setting as introduced in Section 2.1.3. Since we are in the computational setting, we have $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$.

The protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Additionally, since we are in the UC setting, we need a setup as demanded by the subprotocol $\pi_C$ in Section 3.5.

The procedure will be analogous to the procedure in Section 4.2: The first step is to replace the IT protocol $\pi^N$ with the ideal system $I^{\text{NoSec}}_N$ and the corresponding simulator $\sigma^N$. In a second step, we replace the modified broadcast channel $BC'$ together with the local machines $\pi^v$ (implementing a majority vote) with a normal broadcast $BC$ and the corresponding simulator $\sigma^{BC}$. Then, we replace the computational protocol $\pi^C$ with the ideal system $I_{\text{abort}}^C$ and the corresponding simulator $\sigma^C$. Finally, based on the replacements, we construct the complete simulator $\sigma^{\Pi_f}$ as required for the proof.

4.3.1 Using the simulator for $\pi^N$

First, we consider the protocol $\pi^N$, simulating the designated party $P_N$. From the security of protocol $\pi^N$ as stated in Lemma 3.1 and the definition of universal compositability, we have an UC-simulator $\sigma^N$ for which

$$\text{adv}^D(\sigma^N_0(I^{\text{NoSec}}_N), \pi^N_0(R)) \leq \epsilon(\kappa)$$

holds. It follows that we can indistinguishably replace the protocol $\pi^N$ with the simulator $\sigma^N$ and the ideal model $I^{\text{NoSec}}_N$. The resulting systems are IT-indistinguishable. This is illustrated above in Figure 9. Again, using an auxiliary distinguisher $D^{\text{aux}}$ as in Section 4.2.1, we can apply this property to the complete protocol $\Pi_f$, which is illustrated in Figure 15.

In the resulting system, $I^{\text{NoSec}}_N$ cannot be considered as an honest protocol machine. It is completely controlled by the adversary and can send arbitrary messages. Furthermore, any input provided by honest parties to $I^{\text{NoSec}}_N$ is forwarded to the adversary. Again, the simulator $\sigma^N$ could be discarded in the following discussion. However, it is part of the complete simulator $\sigma^{\Pi_f}$, so we leave it in place.

4.3.2 Replacing the broadcast channel

After the replacement in the previous section, we have a modified broadcast channel $BC'$ and local machines $\pi^v$, implementing a majority vote. These components are different from the definition of subprotocol $\pi^C$. Hence, in the next step, we replace the broadcast channel $BC'$ together with the local machines $\pi^v$ (implementing a majority vote) with a
system that allows us to replace the protocol $\pi^C$ (as we will do in the following step). Note that in this case, the ideal functionality $I_{NoSec}^N$ is not a concern. It is completely controlled by the adversary, and thus is not replaced together with the honest protocol machines $\pi_i^C$.

The new system consists of two components: A standard broadcast channel $BC$, and a simulator $\sigma^{BC}$ which is connected to the interfaces that belong to the corrupted parties. This time, $P_N$ also belongs to this group. This is shown in Figure 16.

There are 3 different message flows, the simulator $\sigma^{BC}$ has to handle (see Section 3.3). It does so in the following way:

$P_N \rightarrow \pi_i^C$. The channels running from $I_{NoSec}^N$ to the simulator represent the channels from the honest protocol machines $\pi_i^N$ to $BC'$, the channels from simulator $\sigma^N$ to the simulator $\sigma^{BC}$ the ones belonging to the corrupted parties. However, the adversary controls all these channels and can send any messages he wants. In the simulation, $\sigma^{BC}$ first sends a copy of all the messages to the distinguisher. Then, it applies the majority vote algorithm $\pi^v$ to the messages and forwards the result to $BC$. The simulator $\sigma^{BC}$ receives this same message from $BC$, but ignores it.

$\pi_i^C, i \in \mathcal{I} \rightarrow P_N$. The simulator receives the broadcasted message from $BC$ and forwards it to the distinguisher, the simulator $\sigma^N$, and $I_{NoSec}^N$.

$\pi_i^C, i \in \mathcal{A} \rightarrow P_N$. The simulator receives the message to be broadcasted from the distinguisher and forwards it to $BC$, the simulator $\sigma^N$, and $I_{NoSec}^N$.

It is straightforward to see that the two systems are perfectly indistinguishable. In the final system we replace $I_{NoSec}^N$ (which only forwards messages) with corresponding channels. In contrast to the previous case, $I_{NoSec}^N$ appears as a corrupted party in protocol $\pi^C$ and would not be absorbed into the ideal functionality. In this process, a couple of “self-loops” appear (channels with both ends connected to the same entity), which look curious and
are only needed for formal correctness: The simulator sees all communication taking place during the protocol execution of $\pi^N$, including its own messages.

After this replacement, we have a normal broadcast channel $BC$ and no more local machines $\pi^v$. Now it is possible to consider the protocol $\pi^C$ and perform a similar replacement as for protocol $\pi^N$, since all components comply with the definition of protocol $\pi^C$. Again, we leave the simulator $\sigma^{BC}$ in place since it is part of the complete simulator $\sigma^{\Pi_f}$.

4.3.3 Using the simulator for protocol $\pi^C$

The replacements in the previous sections result in a system that applies the protocol $\pi^C$ according to its definition, i.e. there is a normal broadcast channel $BC$ and correct protocol machines for all honest parties, this time not including the designated party $P_N$.

In the next step we take the new system and replace the protocol $\pi^C$ in the same way. From the security of $\pi^C$ as stated in Lemma 3.3 we have an UC simulator $\sigma^C$ for which

$$\text{adv}^D(\sigma^C_{\Delta}(I_{C}^{abort}), \pi^C_{\Delta}(R))) \leq \epsilon(\kappa)$$

holds. The corresponding illustration is similar to the one in Figure 9, again just replace $N$ with $C$. The protocol $\pi^C$ requires a CRS-setup that could be illustrated with a box that is connected to all entities. As in the previous case, for the sake of simplicity, we omit the setup in following discussion, since it would only detract from the main points of the proof.

Again we have to redefine what the distinguisher exactly comprises in order to make the simulation work. As in the previous case, we consider everything but the honest protocol machines $\pi^C$ and the broadcast channel $BC$ to be the environment. This time, in contrast
4.3 Proof for the computational case in the UC setting

Figure 17: Replacing $\pi^C$ in the full protocol $\Pi_f$ (UC setting, computational scenario).

to the information-theoretic scenario, the ideal functionality $I_{\text{NoSec}}^N(R)$ (which was replaced with corresponding channels) introduced in the previous step cannot be considered an honest party but is part of the adversary set $\mathcal{A}$. Then, we substitute the remaining system $\pi_C^C(R)$ with $\sigma_{\mathcal{S}}^C(I_{\text{abort}}^C)$. This process is illustrated in Figure 17.

In the resulting system, $I_{\text{abort}}^C$ takes two different kinds of input. On one hand, it takes $n$ inputs of the form $(x_i^C, c_i)$. On the other hand, it takes one input (consisting of $n$ parts) of the form $(x_i^N, o_i)_{i \in [n]}$. Then, $I_{\text{abort}}^C$ computes the steps as described in step 4 of the complete protocol $\Pi_f$ as in Section 3.6, i.e. performs a consistency check, recovers the original input and computes the output of $f$.

With this last replacement we are already very close to the ideal model that the complete protocol $\Pi_f$ implements. However, the input is still split into two parts, and additionally contains a commitment. It is the task of the simulator $\sigma^f$, which we introduce in the next section, to take care of these message flows.

### 4.3.4 Assembling the complete simulator $\sigma^{{\Pi}_f}$ for protocol $\Pi_f$

As in Section 4.2.4, the last step before assembling the complete simulator $\sigma^{{\Pi}_f}$ for protocol $\Pi_f$ is to construct a simulator $\sigma^f$ that makes the ideal functionality $I_{\text{abort}}^C$ indistinguishable from $\pi_C^C(I_{\text{abort}}^C)$. This is illustrated in Figure 18. After the last replacement, the ideal functionality $I_{\text{abort}}^C$ still takes two different kinds of input $(x_i^N, o_i)$ and $(x_i^C, c_i)$, which additionally are correlated with a commitment. However, in the end, the ideal functionality $I_{\text{abort}}^C$ only takes one kind of input $x_i$, and neither needs to perform any consistency checks, nor does it have to recover the input $x_i$. The simulator $\sigma^f$ has to take care of this.
4.3 Proof for the computational case in the UC setting

The simulator $\sigma^f$ will only be connected to the interfaces belonging to the corrupted parties. The inputs belonging to the honest parties are given directly to the ideal functionality $I_{abort}^f$, and we do not have to worry about that. So, the simulator $\sigma^f$ has two direct connections to the environment. For a better understanding, we denote the interface in Figure 18 to simulator $\sigma^N$ by C1, and the interface to simulator $\sigma^C$ by C2.

The simulator has to produce the correct distributions on these two interfaces and the interface to the ideal functionality $I_{abort}^f$, in order to render $\sigma^f(I_{abort}^f)$ indistinguishable from $\pi_B(I_{abort}^f)$.

First, over interface C1, the environment $D' = D \circ D_{aux}$ expects the input of the honest parties intended for the ideal functionality $I_{NoSec}^N$, which is of the form $\{x_i^N, o_i\}_{i \in \mathcal{H}}$. This information is not available and has to be simulated by $\sigma^f$ correctly. On the other hand, over interface C2, the simulator $\sigma^f$ only receives the input of the corrupted parties intended for $I_{abort}^C$, which consists of two parts: $\{x_i^N, o_i\}_{i \in \mathcal{H}}$ representing the input of party $P_N$ to protocol $\pi_C$, and $\{x_i^C, c_i\}_{i \in \mathcal{A}}$ representing the input of the corrupted parties $\pi_C^i$, $i \in \mathcal{A}$ to protocol $\pi_C$.

The simulation is depicted in Figure 19. The simulator $\sigma^f$ performs the following steps:

1. For $i \in \mathcal{H}$, chose $\tilde{x}_i^N$ at random
2. For these $\tilde{x}_i^N$, compute the commitments $C(\tilde{x}_i^N)$ and the corresponding opening information $\tilde{o}_i$.
3. Give $\{\tilde{x}_i^N, \tilde{o}_i\}_{i \in \mathcal{H}}$ as output over C1.
4. Receive two kinds of input over C2, intended for $I_{abort}^C$.
4.3 Proof for the computational case in the UC setting

Figure 19: The simulator for the computational scenario in the UC setting. The dashed line separates the algorithm implemented by the simulator $\sigma^f$ on the right hand side, from the way it interacts with the given simulators $\sigma^C$, $\sigma^N$, and $\sigma^{BC}$ on the left hand side.

- $\{x_i^N, o_i\}_{i \in [n]}$ representing the input of party $P_N$ to protocol $\pi^C$.
- $\{x_i^C, c_i\}_{i \in \mathcal{A}}$ representing the input of the corrupted parties $\pi^C_i, i \in \mathcal{A}$ to protocol $\pi^C$.

5. Calculate the XOR $x_i = x_i^C \oplus x_i^N$ for $i \in \mathcal{A}$ and provide the result as input for the corrupted parties to the ideal functionality $I_{f^{abort}}$.

6. Upon receiving the result $y$, check whether $\forall i \in \mathcal{H} : (\tilde{x_i^N, \tilde{o_i}}) \equiv (x_i^N, o_i)$ (Honest parties’ input to $\pi^C$ unchanged) and $\forall i \in \mathcal{A} : \text{open}(c_i, o_i) \equiv x_i^N$ (Consistent opening information for the corrupted parties) hold:

- [holds] $y$ is forwarded to C2 and the abort flag is forwarded from C2 to the ideal functionality.
- [fails] $\perp$ is forwarded to C2 and the ideal functionality is aborted.

In the execution of $\pi_{\mathcal{H}}(I_{C^{abort}})$, party $P_1$ first splits its input into $x_i = x_i^C \oplus x_i^N$ and computes the commitment $c_i = C(x_i^N)$ together with the corresponding opening information $o_i$. This is done locally without interaction. Then, party $P_1$ provides $\{x_i^N, o_i\}$ as input to protocol $\pi^N$. The simulation in $\sigma^f$ imitates this input phase in steps 1-3: The protocol $\pi^N$ is insecure and completely controlled by the adversary, hence the input of the honest parties to $\pi^N$ is directly forwarded to the adversary (in other words, the environment expects these values over C1). The simulator $\sigma^f$ has to fake these values. Due to the argument below this can be achieved with computational indistinguishability.
After that, party $P_i$ provides $x_i^C$ together with the commitment $c_i$ as input to the ideal functionality $I_C^{abort}$. Again, the simulation in $\sigma^f$ imitates this input phase in step 5: The ideal functionality $I_C^{abort}$ remains private and correct. Hence, the corrupted parties, including the simulated party $P_N$, have to provide their input without obtaining any information about the input of the honest parties. The environment has to provide this data over $C2$.

Now, the ideal functionality $I_C^{abort}$ is started: It first checks the input (simulated in step 7), and then computes the output (simulated in step 6). The change in the order of the last two steps does not affect the result. All that matters is that the adversary receives the output if and only if the check succeeds. This holds in the real as well as in the ideal model.

It is easy to see that the simulation in Figure 19 works:

- Instead of the correct values $\{x_i^N\}_{i \in \mathcal{H}}$, the simulator $\sigma^f$ provides random values $\{\tilde{x}_i^N\}_{i \in \mathcal{H}}$ to the environment over $C1$. However, both the $\{x_i^N\}_{i \in \mathcal{H}}$ and the $\{\tilde{x}_i^N\}_{i \in \mathcal{H}}$ are independent and uniformly distributed. Furthermore, in both cases the opening information is computed according to the protocol. Hence, there is no difference between the real system and the simulation.

- The commitments to the values $x_i^N$ in the protocol have been introduced to guarantee that the computations in protocol $\pi^C$ are carried out with correct values $x_i^N$. That is, the input $x_i^C$ provided by the (honest) parties and the inputs $x_i^N$ provided by party $P_N$ have the relation $x_i^C \oplus x_i^N = x_i$. Otherwise, if the adversary controls the protocol $\pi^N$ (as is the case here), he could change the values $x_i^N$, and at the same time pretend that everything is alright. This would lead to a computation with wrong inputs $x_i$ and hence to an incorrect result.

Indeed, in the simulation $\sigma^f$, when the adversary tries to change the value $x_i^N$, the protocol is always aborted and no output is provided neither to the adversary nor to the honest parties. The same thing happens in the execution of $\pi^R(I_C^{abort})$, unless the adversary can open a commitment for a different value (i.e. provide $o_i$ such that $\text{open}(C(\tilde{x}_i^N), o_i) = x_i^N$ holds with $\tilde{x}_i^N \neq x_i^N$ for $i \in \mathcal{H}$). Then the execution would proceed and the adversary would receive a result.

To exploit this difference, the adversary has to be able to open one of the commitments differently. However, these (hiding) commitments are computationally binding, which means that in the real system the probability that the (computationally bounded) adversary achieves this is negligible. As a consequence, the difference between the real system and the simulation is negligible.

This shows that the distributions on all interfaces of simulator $\sigma^f$ connected to the ideal functionality $I_C^{abort}$ are computationally indistinguishable from the distributions on the interfaces of the ideal functionality $I_C^{abort}$ connected to protocol machines $\pi_i$. In other words, for all distinguishers, the left hand side and the right hand side in Figure 18 are computationally indistinguishable.
Hence, the construction $\sigma^{\Pi_f}$ incorporating all simulators $\sigma^N$, $\sigma^C$, $\sigma^{BC}$, and $\sigma^f$ as shown in Figure 19 acts as a simulator for the ideal functionality $I_f^{\text{abort}}$ against the complete protocol $\Pi_f$. Thus it fulfills the requirement

$$\text{adv}^D(\sigma^{\Pi_f}(I_f^{\text{abort}}), \Pi_\Delta(R)) \leq \epsilon(\kappa)$$

for the class of efficient distinguishers $D = D^{\text{eff}}$, thereby proving the first part of Theorem 3.5.

5 The proof of security in the stand-alone setting

Now we finally prove Chaum’s claim as stated in Theorem 3.4. For this purpose, we show that for any adversary $A$ there is a simulator $\sigma^{\Pi_f}$ that makes the ideal system with abort $I_f^{\text{abort}}$ indistinguishable from the real protocol execution (according to Section 3.6), i.e. security of $\Pi_f$ in the stand-alone setting. The indistinguishability is either information-theoretic for $t < n/2$ corrupted parties, or computational for $t < n$ corrupted parties.

In Section 3.3 we have seen how the subprotocol $\pi^N$ can be realized in the UC setting, and in Section 3.4 how the subprotocol $\pi^C$ can be realized in the stand-alone setting. In this scenario the protocol is based on a complete and synchronous network of secure channels and a broadcast channel. In contrast to the UC setting we do not require any setups. However, the security is weaker because this time the distinguisher may not interact with the system but only receives the output. Thus, protocols that only provide stand-alone security cannot be composed freely.

Again, we prove the computational and the information-theoretic case separately, first showing how to use the simulators $\sigma^C$ and $\sigma^N$ for the subprotocols $\pi^C$ and $\pi^N$, respectively. Then we construct the simulator $\sigma^{\Pi_f}$ for the complete protocol $\Pi_f$.

5.1 Proof for the information-theoretic case

We start again with proving the first property, this time of Theorem 3.4, claiming that the protocol $\Pi_f$ implements the ideal model $I_f^{\text{abort}}$ with abort and agreement on abort (without obtaining fairness or robustness) with information-theoretic security, given that only a minority of the parties is corrupted, in the stand-alone setting. The protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the IT scenario, we have $A = A^{\text{all}}$, $\Sigma = \Sigma^{\text{all}}$, and $D = D^{\text{all}}$. However, as the simulators $\sigma^C$ and $\sigma^N$ for the subprotocols $\pi^C$ and $\pi^N$ are all in $\Sigma^{\text{eff}}$ as demanded in Lemmata 3.2 and 3.1, the resulting simulator $\sigma^{\Pi_f}$ is as well.

From Lemma 3.2 we have a stand-alone simulator $\sigma^C$ for $I_C^{\text{des}}$ for which $\sigma^C_A(A) \circ I^{\text{des}}_C \approx A \circ \sigma^C_\Delta(R)$ holds. From Lemma 3.1 we have a UC simulator $\sigma^N$ for $I^{\text{sec}}_C$ to replace protocol $\pi^N$. We can also reuse the simulator $\sigma^{BC}$ introduced for the information-theoretic UC setting in Section 4.2.2. We use all three of them to construct the complete simulator $\sigma^{\Pi_f}$.

So, we still have UC simulators $\sigma^N$ and $\sigma^{BC}$ which we can use here in the same way we did above in the UC setting (see Section 4.2). Hence, we can directly start off with the system,
where the protocol $\pi^N$ and the modified broadcast channel $BC'$ are already replaced with the ideal model $I^N_N$ and a standard broadcast channel $BC$, and concentrate on the task of applying the stand-alone simulator $\sigma^C$ and constructing the complete simulator $\sigma^\Pi_f$.

### 5.1 Proof for the information-theoretic case

In this section we replace the protocol $\pi^C$ with the ideal functionality $I^C_N$ and the simulator $\sigma^C$. The ideal functionality $I^N_N$ that was introduced when replacing protocol $\pi^N$ is completely secure guaranteeing privacy, correctness, and robustness. Hence it correctly executes the protocol machine $\pi^C$ implementing a correct designated party in protocol $\pi^C$. As a consequence, the simulator $\sigma^C$ does not control its interface.

This confronts us with a problem: As of now, the adversary determines a part of the input that belongs to a honest party. The definition of stand-alone security (see Def. 2.1) does not cover such a scenario. However, this problem is solved by the strict ordering of the steps in the full protocol $\Pi_f$ (see Section 3.6): The protocol machines $\pi^C_i$ in step 4 are started only after the protocol machine $\pi^C_N$ has committed to its input. Thus we make sure that every honest party first provides its input to the protocol $\pi^N$, and only then exchanges messages for protocol $\pi^C$.

Based on that fact we can do the following: We first consider $\sigma^{BC}(\sigma^N(A))$ together with the “splitting protocols”\(^3\) (denoted by $\pi_a$ and $\pi_b$ in Figure 20) as being part of the environment. Upon initialization, the simulator $\sigma^N$ outputs messages of the form $(x^N_{i}, o_i)$ that represent a part of the input to the protocol machine $\pi^C_N$ (the part belonging to the corrupted parties). Second, after this step, the construct $\sigma^{BC}(\sigma^N(A))$ acts like an adversary for protocol $\pi^C$. Hence, we can start the simulator $\sigma^C$, providing this new adversary as input. This step is illustrated in Figure 20. Finally, we have to provide the new adversary $\sigma^{BC}(\sigma^N(A))$ with the output of the protocol machine $\pi^C_N$. However, since we only consider symmetric functionalities (see Section 2.3), the output of $\pi^C_N$ is equal to the output of all parties participating in the protocol $\pi^C$. Thus, the simulator $\sigma^C$ already received this output and forwarded it to the new adversary $\sigma^{BC}(\sigma^N(A))$.

In the resulting system, $I^C_N$ takes $n$ inputs of the form $(x^C_i, c_i)$, $t$ of which belong to the corrupted parties, and one input (consisting of $n$ parts) of the form $(x^N_i, o_i)_{i \in [n]}$. The honest parties’ input (with the exception of the input from the designated party $P_N$) is forwarded to $\sigma^C$. Then, $I^C_N$ computes the steps as described in step 4 of the complete protocol $\Pi_f$ (see Section 3.6), i.e. performs a consistency check, recovers the original input and computes the output of $f$.

With this last replacement we are already very close to the ideal model that the complete protocol $\Pi_f$ implements. However, the input is still split into two parts, and additionally

---

\(^3\)These protocols split the input $x_i$ into $x^N_i$ and $x^C_i$ such that $x_i = x^N_i \oplus x^C_i$. compute the commitment $C(x^N_i)$ together with its opening information $o_i$, and provide $(x^N_i, o_i)$ and $(x^C_i, C(x^N_i))$ as input for the protocol machines $\pi^C_i$ and $\pi^N_i$, respectively.
5.1 Proof for the information-theoretic case

contains a commitment. It is the task of the complete simulator $\sigma_{\Pi_f}$ to take care of these message flows.

5.1.2 Assembling the complete simulator $\sigma_{\Pi_f}$ for protocol $\Pi_f$

The previous step provided us with a system that uses an ideal functionality $I_{\text{des}}^C$, which takes two different kinds of input and first checks the consistency and recovers the original input, before performing the computation of the output. However, in the complete protocol, we have to give a simulator for the ideal functionality with abort $I_{\text{abort}}^f$, which directly expects the original input and only computes the output. The task of the simulator $\sigma_{\Pi_f}$ is to “transform” the simulator $\sigma^C$ above into one that is suited for $I_{\text{abort}}^f$. That means that on one hand it has to take care of the additional input provided by the simulator $\sigma^C$.

On the other hand, it has to simulate message flows belonging to honest parties, which the simulator $\sigma^C$ expects to see.

Let $C(x)$ denote a hiding commitment to the value $x$, computed in the way as in step 2 of the full protocol $\Pi_f$ as in Section 3.6. Let $S_f = \{t+1, \ldots, n\}$. The simulation is depicted in Figure 21. The simulator $\sigma_{\Pi_f}$ performs the following steps:

1. Receive output $\{x_i^N, o_i\}_{i \in S}$ from simulator $\sigma^N$, intended for $I_{\text{sec}}^N$.
2. For $i \in S_f$, chose $\tilde{x}_i^C$ and $\tilde{x}_i^N$ at random.
3. For these $\tilde{x}_i^N$, compute the commitments $C(\tilde{x}_i^N)$.
4. Give $\{\tilde{x}_i^C, C(\tilde{x}_i^N)\}_{i \in S_f}$ as input to simulator $\sigma^C$.
5. Receive output $\{x_i^C, c_i\}_{i \in S}$ from simulator $\sigma^C$, intended for $I_{\text{des}}^C$.
6. Calculate the XOR \( x_i = x_i^C \oplus x_i^N \) for \( i \in A \) and provide the result as input for the corrupted parties to the ideal functionality.

7. Upon receiving the result \( y \), perform the check: \( \forall i \in A : \text{OPEN}(c_i, o_i) \equiv x_i^N \) (Correct opening information)
   - [holds] \( y \) is forwarded to the simulator \( \sigma^C \) and the abort flag is forwarded from the simulator \( \sigma^C \) to the ideal functionality.
   - [fails] \( \bot \) is forwarded to the simulator \( \sigma^C \) and the ideal functionality is aborted.

8. The output of the simulator \( \sigma^{\Pi_f} \) is the output of the simulator \( \sigma^C \).

Since we are currently treating the information-theoretic scenario, the statement in the check does not imply that the commitment \( c_i \) actually “contains” \( x_i^N \). The commitments are only computationally binding. Hence, the adversary could have “changed” the content. However, this does not affect the protocol and can be seen as a change in the input \( x_i \) provided by the adversary.

The simulation is very similar to the one in the UC setting and the same arguments as in Section 4.2.4, discussing why the simulation works, hold. Hence, the simulator \( \sigma^{\Pi_f} \) fulfills the requirement
\[
\text{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y_{\sigma}, y_{t+1}, \ldots, y_n)) \leq \epsilon(\kappa)
\]
for the class of all distinguishers \( D = \mathcal{D}^{\text{all}} \), thereby proving the second part of Theorem 3.4.
5.2 Proof for the computational case

In this section we prove the second property of Theorem 3.4, claiming that the protocol $\Pi_f$ implements the ideal model $I_{\text{abort}}^f$ with abort and agreement on abort (without obtaining fairness or robustness) with computational security, for any number of corrupted parties, in the stand-alone setting. The protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the computational scenario, we have $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$. We allow an arbitrary number of corrupted parties, hence we no longer have any security guarantees for the protocol $\pi_N$ and have to assume that the adversary completely controls it. This can be formalized using $I_{\text{NoSec}}^N$, the ideal model without security.

From Lemma 3.2 we have a simulator $\sigma_C$ for $I_{\text{abort}}^C$ for which $\sigma_C^C(A) \circ I_{\text{abort}}^C \approx A \circ \pi_C^C(R)$ holds. Hence this simulator does not expect any input (except for the output produced by $I_{\text{abort}}^C$). From Theorem 2.1 we have a UC simulator $\sigma^N$ for $I_{\text{NoSec}}^N$ to replace the protocol $\pi_N$. This simulator expects the inputs belonging to the honest parties which is of the form $\{x_i^N, o_i\}_{i \in H}$. This information is not available and has to be simulated by our simulator $\sigma_{\Pi_f}$ correctly. We can also reuse the simulator $\sigma_{BC}$ introduced for the computational UC setting in Section 4.3.2. Again we use all three of them to construct the complete simulator $\sigma_{\Pi_f}$.

Again, we use the UC simulators $\sigma^N$ and $\sigma_{BC}$ in the same way we did above in the UC setting (see Section 4.3). Hence, we can directly start off with the system, where the protocol $\pi_N$ and the broadcast channel $BC$ are already replaced, and concentrate on the task of replacing the protocol $\pi_C$ (for which we have a stand-alone simulator) and constructing the complete simulator $\sigma_{\Pi_f}$, just like we did in the information-theoretic case.

5.2.1 Using the simulator for the protocol $\pi_C$

The replacements of subprotocol $\pi_N$ and the broadcast channel $BC'$ result in a system that applies the protocol $\pi_C$ according to its definition, i.e. there is a normal broadcast channel $BC$ and correct protocol machines for all honest parties, this time not including the designated party $P_N$.

Again we first explore the effect of introducing the simulator $\sigma_C$ before constructing the complete simulator $\sigma_{\Pi_f}$. As explained in Section 4.3.2, $I_{\text{NoSec}}^N$ can be replaced by corresponding channels. The adversary would control the complete communication to and from this entity, which becomes a corrupted party in protocol $\pi_C$. As a consequence, the simulator $\sigma_C$ controls the corresponding interface.

As in the previous case we need to have a closer look at the construct $\sigma_{BC}(\sigma^N(A))$. First, this construct together with the “splitting protocols” (denoted by $\pi_a$ and $\pi_b$ in Figure 22) is considered as part of the environment that expects the honest parties’ inputs to the protocol $\pi_N$ (this could also be regarded as hardcoding the corresponding input). Second, after this input has been provided, the rest of the communication consists of protocol messages for protocol $\pi_C$. So, when the first message exchange is completed, the
5.2 Proof for the computational case

\[ \forall i \in H: \exists x_i \in \mathbb{N} \subseteq R \{0,1\}^m \]
\[ C(x_i), \tilde{o}_i \rightrightarrows \text{commit}(\tilde{x}_i) \]
\[ \{\tilde{x}_i, \tilde{o}_i\}_{i \in \mathbb{N}} \]

Figure 22: Replacing the protocol \( \pi^C \) in the full protocol \( \Pi_f \) for the computational scenario.

construct \( \sigma^{BC}(\sigma^N(A)) \) acts like an adversary for the protocol \( \pi^C \). Hence, we can start the simulator \( \sigma^C \), providing this new adversary as input. This step is illustrated in Figure 22.

In the resulting system, \( \mathcal{I}_{ab} \) takes two different kinds of input from the simulator. On one hand, it takes \( t \) inputs of the form \((x^C, c_i)\) which belong to the corrupted parties. On the other hand, it takes one input (consisting of \( n \) parts) of the form \((x^N, o_i)_{i \in [n]}\). Then, \( \mathcal{I}_{ab} \) computes the steps as described in step 4 of the complete protocol \( \Pi_f \) (see Section 3.6), i.e. performs a consistency check, recovers the original input and computes the output of \( f \).

With this last replacement we are already very close to the ideal model that implements the complete protocol \( \Pi_f \). However, the input is still split into two parts, and additionally contains a commitment. It is the task of the complete simulator \( \sigma^{\Pi_f} \) to take care of these message flows.

5.2.2 Assembling the complete simulator \( \sigma^{\Pi_f} \) for protocol \( \Pi_f \)

The previous step provided us with a system that uses an ideal functionality \( \mathcal{I}_{ab} \), which takes two different kinds of input and first checks the consistency and recovers the original input, before performing the computation of the output. However, in the complete protocol, we have to give a simulator for the ideal functionality with \( \text{abort} \) \( \mathcal{I}_{ab} \), which directly expects the original input and only computes the output. The task of the simulator \( \sigma^{\Pi_f} \) is to “transform” the simulator \( \sigma^C \) above into one that is suited for \( \mathcal{I}_{ab} \). That means that on one hand it has to take care of the additional input provided by the simulator \( \sigma^C \). On the other hand, it has to simulate message flows belonging to honest parties, which the simulator \( \sigma^C \) expects to see.

Let \( C(x) \) denote the commitment to the value \( x \) using the hiding scheme applied in step 2
of the full protocol (see Section 3.6). Let $\mathcal{F} = \{t + 1, \ldots, n\}$. The simulation is depicted in Figure 23. The simulator $\sigma^N$ performs the following steps:

1. For $i \in \mathcal{F}$, chose $\tilde{x}_i^N$ at random.
2. For these $\tilde{x}_i^N$, compute the commitments $C(\tilde{x}_i^N)$ and the corresponding opening information $\tilde{o}_i$.
3. Give $\{\tilde{x}_i^N, \tilde{o}_i\}_{i \in \mathcal{F}}$ as input to the simulator $\sigma^N$.
4. Then, $\sigma^{BC}(\sigma^N(A))$ is given to the simulator $\sigma^C$ which is then started.
5. The simulator $\sigma^C$ provides two kinds of outputs intended for $I_{\text{abort}}$:
   - Output $\{x_i^N, o_i\}_{i \in [n]}$ representing the input of the designated party $P_N$ to protocol $\pi_C$.
   - Output $\{x_i^C, c_i\}_{i \in A}$ representing the input of the corrupted parties to protocol $\pi_C$.
6. Calculate the XOR $x_i = x_i^C \oplus x_i^N$ for $i \in \mathfrak{A}$ and provide the result as input for the corrupted parties to the ideal functionality.
7. Upon receiving the result $y$, check whether $\forall i \in \mathcal{F} : (\tilde{x}_i^N, \tilde{o}_i) \equiv (x_i^N, o_i)$ (Honest parties’ input to protocol $\pi_C$ unchanged) and $\forall i \in \mathfrak{A} : \text{OPEN}(c_i, o_i) \equiv x_i^N$ (Consistent opening information for the corrupted parties) hold:
   - [holds] $y$ is forwarded to the simulator $\sigma^C$ and the abort flag is forwarded from the simulator $\sigma^C$ to the ideal functionality.
   - [fails] $\perp$ is forwarded to the simulator $\sigma^C$ and the ideal functionality is aborted.

Instead of checking that the honest parties’ input remains unchanged, we could also use the commitments computed above. In that case we would have to check whether $\text{OPEN}(C(\tilde{x}_i^N), o_i) \equiv x_i^N$ holds for $i \in H$. This simulation would be equivalent and the (negligible) difference in the distribution would appear in the output $y$ which in the real system would correspond to wrong inputs for honest parties, whereas in the simulation the honest parties’ input would be unchanged.

The simulation is very similar to the one in the UC setting and the same arguments as to why the simulation works hold (see Section 4.3.4). Hence, the simulator $\sigma^N$ fulfills the requirement

$$\text{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y^N_A, y_{t+1}, \ldots, y_n)) \leq \epsilon(\kappa)$$

for the class of efficient distinguishers $D = \mathcal{D}^{\text{eff}}$, thereby proving the first part of Theorem 3.4.

6 Fairness

In the previous sections we have seen how IT security with abort (excluding fairness and robustness) can be combined with computational security with abort. This was the main concern in [Cha89]. In this section we investigate the possibility to add another security
6.1 Definition of fairness

For a formal description of fairness, we introduce two ideal models $I_{\text{fair}}^f$ and $I_{\text{des,fair}}^f$. They are essentially identical to the ones described in Section 2.2.1 with the only difference that the simulator has to send the abort flag before receiving any output.

6.1.1 Security with abort including fairness

In addition to privacy and correctness, as demanded by security with abort, here the ideal functionality $I_{\text{fair}}^f$ should also specify fairness. This means that the honest parties learn as much about the output as the adversary. However, the adversary may still abort the protocol before the complete output is obtained. Thus, we have no robustness.

We use $\bot$ to represent the output to honest parties resulting from the adversary aborting the protocol. First, all parties send their inputs to the ideal functionality $I_{\text{fair}}^f$. Additionally, the simulator sends a flag indicating whether the ideal model is aborted or not. If $I_{\text{fair}}^f$ is not aborted (flag 0), it computes the output according to $f$ and sends the result.
6.2 Claiming fairness

Our goal in this section is to achieve fairness, given that $t < \frac{n}{2}$ parties are corrupted, and that a broadcast channel and a network of secure channels are available. We state the corresponding theorem for both the stand-alone and the UC setting (the latter one
6.3 New requirements for the protocol $\Pi^C$

requiring an additional CRS-setup). However, in the following discussion, we only consider
the stand-alone setting. The modifications and proofs for the UC setting follow readily.

Whereas the first property of Theorem 3.4 (for the computational scenario in the stand-
alone setting) remains unchanged, we have to adapt the second property for the IT sce-
nario, applying the definition of an ideal functionality $I^\text{fair}_f$ with fairness in Section 6.1.1.

**Theorem 6.1.** *Fairness in the stand-alone setting.*

$\Pi^\text{fair}$ implements

1. the ideal model $I^{\text{abort}}_f$ (with agreement on abort), with computational security (i.e. $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$), given that $t < n$ parties are corrupted, and

2. the ideal model $I^\text{fair}_f$ (with fairness), with IT security (i.e. $A = A^\text{all}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^\text{all}$), given that $t < n/2$ parties are corrupted,

from a complete and synchronous network of secure channels, and a broadcast channel,
in the stand-alone setting.

And analogously for the UC setting:

**Theorem 6.2.** *Fairness in the UC setting.*

$\Pi^\text{fair}$ implements

1. the ideal model $I^{\text{abort}}_f$ (with agreement on abort), with computational security (i.e. $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$), given that $t < n$ parties are corrupted, and

2. the ideal model $I^\text{fair}_f$ (with fairness), with IT security (i.e. $A = A^\text{all}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^\text{all}$), given that $t < n/2$ parties are corrupted,

from a complete and synchronous network of secure channels, and a broadcast channel,
in the stand-alone setting.

6.3 New requirements for the protocol $\Pi^C$

For the whole protocol $\Pi^{\text{fair}}$ to achieve fairness as claimed in the previous section, we only
have to demand a certain kind of fairness from the protocol $\Pi^C$. For this purpose, we mod-
ify Lemma 3.2 (stating the security requirements for $\Pi^C$ as needed for the protocol $\Pi_f$) to
include the new requirements, applying the definition of an ideal functionality $I^\text{des.fair}_f$
with a designated party including fairness in Section 6.1.2.

**Lemma 6.3.** *There is a protocol $\Pi^C$ that implements*

1. the ideal model $I^{\text{abort}}_f$ (with abort), with computational security (i.e. $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$),

2. the ideal model $I^\text{des.fair}_f$ (with designated party $P_N$), with IT security (i.e. $A = A^\text{all}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^\text{all}$),
6.3 New requirements for the protocol $\pi^C$

from a complete and synchronous network of secure channels, and a broadcast channel, in the stand-alone setting, given that $t < n$ parties are corrupted.

The only difference to Lemma 3.2 is that in the information-theoretic case the protocol has to securely implement $I^{\text{des, fair}}_f$ instead of $I^{\text{des}}_f$. Before we can give a proof for Theorem 6.1 for the protocol $\Pi^{\text{fair}}_f$, we have to show that such a protocol $\pi^C$ exists. This is done in the next section.

6.3.1 Further modifications of [GMW87]

In this section we discuss how to obtain a protocol $\pi^C$ that fulfills the requirements in Lemma 6.3. Obviously, we would like to use the modified version of [GMW87] as described in Section 3.4.1. However, this protocol does not achieve the designated fairness property in the IT scenario. Fortunately, in order to achieve fairness, we only need to consider the output reconstruction phase in [GMW87], when each party has to send its shares of the output to the other parties. So far, this is done in an arbitrary order. All we do now is to fix a more specific ordering. The original security properties are not affected by this.

First, each party $P_i, i \in [n]$ performs its part of the output reconstruction phase (i.e. it sends its shares of the output to the other parties, together with a proof of correctness). All these messages are sent over the broadcast channel. Hence, if one party (or more) sends an incorrect message, or no message at all, then the other parties (including the designated party $P_N$) notice it and take according actions (i.e. abort the protocol$^4$).

Finally, if and only if all messages have been sent correctly, the designated party $P_N$ performs its part of the output reconstruction phase. Otherwise it stops.

In the following section we show that this modification achieves the required security guarantees.

6.3.2 Proof sketch of Lemma 6.3: The modified [GMW87] achieves fairness

In this section we prove that the latest modifications of [GMW87] result in a protocol $\pi^C$ that fulfills all requirements stated in Lemma 6.3.

First, the modifications in the previous section do not affect any of the proven security results, because all we did was to fix an ordering that was arbitrary before. Thus, we can directly concentrate on the fairness property.

Due to the chosen sharing scheme, no party can reconstruct its output without the share coming from the designated party $P_N$. However, $P_N$ distributes his output shares only after all possibly corrupted parties provided their information to all other parties. Once started, $P_N$ does not stop sending its output shares. Hence, the rest of the protocol run is independent of the participation of corrupted parties and they cannot prevent any party from reconstructing its output.

As a consequence, either all parties obtain the output, or none. Thus the fairness claim is fulfilled. We refrain from providing the corresponding simulators as they follow readily.

$^4$In Section 7 we consider robustness. In that case it is possible to reconstruct the missing message.
6.4 Adapting the proof

Now we want to prove Theorem 6.1. The first claim in this theorem is that the protocol $\Pi^f_{fair}$ implements the ideal model $I^f_{abort}$ with abort and agreement on abort (without obtaining fairness or robustness) with computational security, for any number of corrupted parties, in the stand-alone setting. The proof for this part is identical to the proof in Section 5.2.

The second claim in Theorem 6.1 is that the protocol $\Pi^f_{fair}$ implements the ideal model $I^f_{fair}$ with fairness (without obtaining robustness) with information-theoretic security, given that only a minority of the parties is corrupted, in the stand-alone setting. To prove this claim, we have to adapt the proof from Section 5.1. Luckily, all steps until the introduction of the simulator and ideal model for protocol $\pi^C$ are done in the same way. The only difference in the proof is that when replacing the protocol $\pi^C$, we introduce the ideal functionality $I^f_{des, fair}$ instead of the ideal functionality $I^f_{des}$. What remains to do is to modify the complete simulator $\sigma^\Pi$ to obtain $\sigma^{\Pi^f_{fair}}$.

The protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the IT scenario, we have $A = A^{all}$, $\Sigma = \Sigma^{all}$, and $D = D^{all}$. However, as the postulated simulators (in Theorem 6.3 and 3.1) are all in $\Sigma^{eff}$, the resulting simulator $\sigma^{\Pi^f_{fair}}$ is as well.

The result is shown in Figure 26. The algorithmic description follows readily.

1. Receive output $\{x^N_i, o_i\}_{i \in A}$ from the simulator $\sigma^N$, intended for the ideal functionality $I^f_{sec}$.
2. For \( i \in H \), chose \( \tilde{x}_i^C \) and \( \tilde{x}_i^N \) at random

3. For these \( \tilde{x}_i^N \), compute the commitments \( C(\tilde{x}_i^N) \).

4. Give \( \{\tilde{x}_i^C, C(\tilde{x}_i^N)\}_{i \in H} \) as input to the simulator \( \sigma^C \).

5. Receive output \( \{x_i^C, c_i\}_{i \in A} \) from the simulator \( \sigma^C \), intended for the ideal functionality \( I_{des,fair} \).

6. Calculate the XOR \( x_i = x_i^C \oplus x_i^N \) for \( i \in A \) and provide the result as input for the corrupted parties to the ideal functionality \( I_{fair} \).

7. Perform the CHECK: \( \forall i \in A : \text{OPEN}(c_i, o_i) \equiv x_i^N \) (Correct opening information)
   - [holds] The abort flag is forwarded from the simulator \( \sigma^C \) to the ideal functionality \( I_{fair} \).
   - [fails] The ideal functionality \( I_{fair} \) is aborted.

8. Upon receiving the result \( y \), forward it directly to the simulator \( \sigma^C \).

9. The output of the simulator \( \sigma^{\Pi_f} \) is the output of the simulator \( \sigma^C \).

Up to step 6, the simulation is identical to the simulation in Section 5.1.2. The only difference lies in the order of the last two messages. Now, the simulator provides the abort flag before expecting to see the output.

In the real protocol execution, the simulator \( \sigma^C \) receives \( y = \perp \) either when it sent the abort flag 1, or when it sent incorrect opening information such that \( \text{OPEN}(c_i, o_i) \equiv x_i^N \) does not hold. In both cases the protocol execution is aborted and the output \( y = \perp \) is provided to all parties. The simulation above guarantees the exact same behavior and is hence perfectly indistinguishable from a real protocol execution. Hence, the simulator \( \sigma^{\Pi_f}_{fair} \) fulfills the requirement

\[
\text{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y_{\sigma^{\Pi_f}_{fair}, y_{t+1}, \ldots, y_n})) \leq \epsilon(\kappa)
\]

for the class of all distinguishers \( D = D^{all} \), thereby proving the second part of Theorem 6.1.

## 7 Robustness

In Section 3 we started with a protocol \( \Pi_f \) that provides IT security with abort (i.e. correctness and privacy, without fairness or robustness) for an honest majority, and computational security with abort for any number of corrupted parties. In the previous section, we extended the protocol obtaining a new protocol \( \Pi_{f,\text{fair}} \), which additionally provides IT security with fairness for an honest majority. Now, in this section, we investigate the possibility to include yet another security guarantee: Robustness. In general, robustness can only be achieved when \( t < \frac{n}{2} \) parties are actively corrupted, given a broadcast channel (see [Cle86]). Furthermore, according to [IKLP06, Kat07], there is a tradeoff between the bound for robustness \( \rho \) and the bound \( n - \rho \), where privacy is lost.
In contrast to the fairness case, it is necessary, but not sufficient to adapt the subprotocol \( \pi^C \). In order to include robustness, we also have to adapt the complete protocol, obtaining the new protocol \( \Pi_f^p \). This is discussed in Section 7.1. In this section, the need of a \textit{callback} arises, which we formalize, together with robustness, in Section 7.2. Given these new definitions of security, we formally state the security guarantees achieved by the new protocol \( \Pi_f^p \) in Section 7.3. Before we give a proof of this statement in Section 7.5, we discuss necessary modifications of the subprotocol \( \pi^C \), and how they can be realized, in Section 7.4.

### 7.1 Adapting \( \Pi_f \) for robustness

In this section, we adapt the complete protocol \( \Pi_f \) from Section 3.6 in such a way that robustness can be achieved. In protocol \( \Pi_f \) the subprotocol \( \pi^C \) implements the ideal functionality that performs three steps: Check the input for consistency, recover the actual input, and compute the output. If the consistency check fails, the execution is aborted and the output is empty. This gives rise to a problem when trying to achieve robustness. It is quite easy for an adversary to abort the protocol by simply sending a wrong commitment instead of \( C(x_i^N) \).

Therefore, to obtain robustness it is necessary to adapt the original protocol \( \Pi_f \) and introduce a callback option: If there is an inconsistent triple \((c_i, o_i, x_i^N)\), the functionality for protocol \( \pi^C \) asks the corresponding party \( P_i \) to provide the desired input \( x_i^N \). If \( P_i \) does not react on this callback, we set \( x_i^N = 0 \). Eventually, the protocol proceeds with the computation.

Hence, the new protocol \( \Pi_f^p \) performs the following steps:

**Initial Situation:** Each party \( P_i, 1 \leq i \leq n \) has an input \( x_i \).

**Goal:** Each party \( P_i \) holds the output \( y = f(x_1, ..., x_n) \).

1. \( \forall P_i : x_i^N \leftarrow_R \{0, 1\}^m, \quad x_i^C := x_i \oplus x_i^N \)

2. Using a hiding commitment scheme \( C(\cdot) \), each party \( P_i \) computes \([C(x_i^N), o_i] = \text{commit}(x_i^N)\)

3. \( \forall i : P_i \) starts a protocol machine \( \pi_i^N \) on input \((x_i^N, o_i)\)
   (protocol \( \pi^N \) emulates \( \pi^C \) with input \( \{x_i^N, o_i\}_{i \in [n]} \) IT securely as stated in Lemma 3.1)

4. \( \forall i : P_i \) starts a protocol machine \( \pi_i^C \) on input \((x_i^C, C(x_i^N))\)
   The broadcast interfaces of each protocol machine \( \pi_i^C \) is connected to a local machine \( \pi_i^r \) implementing the majority vote (see Def. 3.2). The interface for the private channel with the designated party \( P_N \) is connected locally to protocol machine \( \pi_i^N \). Protocol \( \pi^C \) is designed to implement the ideal functionality \( I_{\text{abst, rob}, cb} \) with computational security, and \( I_{\text{des, rob, rob}, cb} \) and \( I_{\text{des, fair, rob}, cb} \) with IT security and designated party \( P_N \), see Lemma 7.3) that performs the following steps:

   (a) Input check: \( \forall i : \text{open}(c_i, o_i) = x_i^N \).
   For every \( i \) where this check fails, ask party \( P_i \) to send \( x_i^N \) (use \( x_i^N = 0 \) in case the party does not react). For a detailed explanation on how this callback works, see Section 7.2.
7.2 New definitions of security with a callback

(b) Recover input: \( \forall i : x_i = x_i^N \oplus x_i^C \)

(c) Computation and output (unless abort): \( y = f(x_1, ..., x_n) \)

With this new protocol \( \Pi^f_\rho \), robustness can be achieved. In order to formalize this statement, we first have to specify the corresponding ideal functionalities, which is the goal of the next section. Furthermore, the requirements for subprotocol \( \pi^C \) have changed as well. Hence, we have to formalize these requirements and show that they can be fulfilled. This is topic of Section 7.4.

7.2 New definitions of security with a callback

The previous section describes the new protocol \( \Pi^f_\rho \), which achieves robustness. In this section, we introduce the necessary ideal functionalities that are essential to formalize this statement. The ideal functionality including privacy, correctness, and robustness is the ideal functionality with full security \( I^{sec}_f \) as described in Section 2.1. We prove in Section 7.5 that the protocol \( \Pi^f_\rho \) implements this functionality under certain conditions.

As already mentioned, it is also necessary to adapt the requirements for subprotocol \( \pi^C \). For that purpose, we have to modify the ideal functionalities \( I^{sec}_f \), \( I^{abort}_f \), and \( I^{des, fair}_f \) to obtain a designated party property and a callback, respectively. In general, the callbacks work as one might assume: If the ideal functionality requires additional input, it performs a callback with the corresponding party, which provides the information (with a fixed default value in case the party does not provide the information).

On first glance, these callbacks seem to require some sort of interaction. This would contradict the model of stand-alone security, since in the corresponding definition no interaction beyond input-output is allowed. However, this predicament can be solved in the given situation as follows: We define the input of a party \( P_i \) to consist of two parts \( x'_i = x_i \parallel x_{cb} \). The first input \( x_i \) forms the normal input, which will always be distributed during the input phase. The second input \( x_{cb} \) is the callback input, which will only be distributed when a callback to party \( P_i \) occurs. This modification reduces the callback to a mere “ping”: Now, a callback may not ask anymore for any kind of information, but can only “ping” for a specific value that was fixed before the computation started. Still, for our needs, the callback is powerful enough. For easier understanding and notational convenience, we refrain from applying this strict view and use the callback as introduced above.

Now, there is an important issue concerning the privacy of these callback values. The only reasonable solution is to make the callback values of a party \( P_i \) as private as the input of this party \( P_i \). This is due to the fact that the only practical way to provide this callback information in a real protocol execution is to use the same mechanism as for the normal input. Hence, the ideal functionalities \( I^{des, rob, cb}_f \) and \( I^{des, fair, cb}_f \) forward the callback values belonging to non-designated honest parties to the simulator. On the other hand, \( I^{abort, cb}_f \) protects the privacy of all honest parties. Thus it does not reveal any callback values to the simulator.
7.2 New definitions of security with a callback

7.2.1 Security with a designated party including robustness and callback

Here, we extend the notion of robustness to the ideal model with a designated party and include a callback channel, obtaining the ideal functionality $I_{f_{\text{des, rob, cb}}}$. Now, as long as the designated party $P_N$ is not corrupted, privacy of $P_N$’s input, correctness and robustness (for all honest parties) are guaranteed.

Here we assume that the designated party $P_N$ is not corrupted (in the case where $P_N$ is corrupted we demand no security at all). Again, first all parties send their inputs to the ideal functionality $I_{f_{\text{des, rob, cb}}}$. The inputs of all honest parties but $P_N$ are directly forwarded to the simulator. Depending on the functionality implemented by $I_{f_{\text{des, rob, cb}}}$, the ideal model might ask a subset of the parties to provide more information (over the callback channel, illustrated by the double arrow). If one party (or more) does not provide the additional information, a default value has to be defined and used instead. All callback values but the ones from the designated party $P_N$ are given to the simulator $\sigma$. Finally, the output $y$ is provided to all parties. This is illustrated in Figure 27.

7.2.2 Security with a designated party including fairness and callback

Furthermore, we have to include the callback channel in the ideal model $I_{f_{\text{des, fair, cb}}}$, obtaining $I_{f_{\text{des, fair, cb}}}$.

Figure 28: The ideal model $I_{f_{\text{des, fair, cb}}}$ with a designated party including fairness and a callback.
Here we assume that the designated party $P_N$ is not corrupted (in the case where $P_N$ is corrupted we demand no security at all). Again, first all parties send their inputs to $I_f^{\text{des, fair, cb}}$. The inputs of all honest parties but $P_N$ are directly forwarded to the simulator. Depending on the functionality implemented by $I_f^{\text{des, fair, cb}}$, the ideal model might ask a subset of the parties to provide more information. If one party (or more) does not provide the additional information, a default value has to be defined and used instead. All callback values but the ones from the designated party $P_N$ are given to the simulator $\sigma$. In addition to the input, the simulator sends a flag indicating whether the ideal model is aborted or not. If $I_f^{\text{des, fair, cb}}$ is not aborted (flag 0), it computes the output and sends the result to all parties. Otherwise (flag 1), $I_f^{\text{des, fair, cb}}$ sends $\bot$ to all parties. Like before we demand agreement on abort. This is illustrated in Figure 28.

### 7.2.3 Security with abort including a callback

Finally, we have to include the callback channel in the ideal model with abort, obtaining $I_f^{\text{abort, cb}}$. See Figure 29 for an illustration.

![Figure 29: The ideal model with abort $I_f^{\text{abort, cb}}$ including a callback.](image)

In this case, first, all parties send their inputs to $I_f^{\text{abort, cb}}$. The callback works as in the previous cases. However, this time the callback values are not forwarded to the simulator $\sigma$ since the privacy of all honest parties is protected. Then, $I_f^{\text{abort, cb}}$ computes the output and sends the result to the simulator. Given these values, the simulator decides whether the other parties also receive the output (flag 1) or not (flag 0). Finally, $I_f^{\text{abort, cb}}$ sends either the output $y$ or the empty value $\bot$ to the honest parties, depending on the flag received by the simulator. Again, we demand agreement on abort.

### 7.3 New security claims

Given the new protocol $\Pi_f^\rho$ in Section 7.1 and the definitions of security in the previous section, we are now ready to state the new security claim including robustness. In contrast to the original protocol $\Pi_f$ and the fair protocol $\Pi_f^{\text{fair}}$ with a single threshold, this theorem contains three thresholds and implements four different ideal functionalities: For up to $t \leq \rho$ corrupted parties, the protocol $\Pi_f^\rho$ achieves IT security with robustness, for up to $t < n/2$ corrupted parties it achieves IT security with fairness, and for up to $t < n - \rho$
corrupted parties it achieves computational security with abort. Beyond that bound, no security guarantees can be given [IKLP06, Kat07].

As in Section 6, where fairness was introduced, we state the theorem for both the stand-alone and the UC setting. Again, in the following discussion, we only consider the stand-alone setting. The modifications and proofs for the UC setting follow readily.

Theorem 7.1. Robustness in the stand-alone setting.

\( \Pi^\rho \) implements

1. the ideal model \( I^\text{sec} \) (with robustness), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( 0 \leq |A| \leq \rho \) parties are corrupted,
2. the ideal model \( I^\text{fair} \) (with fairness), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( \rho < |A| < \frac{n}{2} \) parties are corrupted, and
3. the ideal model \( I^\text{abort} \) (with agreement on abort), with computational security (i.e. \( A = A^{eff}, \Sigma = \Sigma^{eff}, \) and \( D = D^{eff} \)), given that \( \frac{n}{2} < |A| < n - \rho \) parties are corrupted,

from a complete and synchronous network of secure channels, and a broadcast channel, in the stand-alone setting, where \( \rho < \frac{n}{2} \).

And analogously for the UC setting:

Theorem 7.2. Robustness in the UC setting.

\( \Pi^\rho \) implements

1. the ideal model \( I^\text{sec} \) (with robustness), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( 0 \leq |A| \leq \rho \) parties are corrupted,
2. the ideal model \( I^\text{fair} \) (with fairness), with IT security (i.e. \( A = A^{all}, \Sigma = \Sigma^{eff}, \) and \( D = D^{all} \)), given that \( \rho < |A| < \frac{n}{2} \) parties are corrupted, and
3. the ideal model \( I^\text{abort} \) (with agreement on abort), with computational security (i.e. \( A = A^{eff}, \Sigma = \Sigma^{eff}, \) and \( D = D^{eff} \)), given that \( \frac{n}{2} < |A| < n - \rho \) parties are corrupted,

from a complete and synchronous network of secure channels, a broadcast channel, and a CRS-setup, in the UC setting, where \( \rho < \frac{n}{2} \).

When more than \( n - \rho \) parties are corrupted, the protocol does not provide any security. However, before we can prove Theorem 7.1, we have to adapt the requirements for subprotocol \( \pi^C \), and show that there is such a protocol.

7.4 New requirements for subprotocol \( \pi^C \)

In Section 6.3 we adapted the subprotocol \( \pi^C \) such that the protocol \( \Pi^\text{fair}_f \) achieved fairness. In this section, again the bounds and security guarantees from the complete protocol \( \Pi^\rho_f \)
are reflected in the subprotocol $\pi^C$. Now, we first state the new requirements for sub-protocol $\pi^C$ by adapting Lemma 6.3, which stated the corresponding requirements for fairness. Then we show that a modified version of [GMW87] fulfills these requirements.

For subprotocol $\pi^C$ we have the same three thresholds as for the complete protocol $\Pi^\rho_f$.

For robustness in the complete protocol $\Pi^\rho_f$, the subprotocol $\pi^C$ has to guarantee robustness, given a designated party. For overall fairness, $\pi^C$ has to guarantee fairness, given a designated party. And for overall security with abort, $\pi^C$ also has to achieve security with abort (though without a designated party). In all three cases, the subprotocol $\pi^C$ may still use callbacks. Of course, in the complete protocol $\Pi^\rho_f$ these callbacks do not appear anymore.

**Lemma 7.3.** There is a protocol $\pi^C$ that implements

1. the ideal model $I_{\text{des},\text{rob},\text{cb}}^f$ (with designated party $P_N$), with IT security (i.e. $A = A^{\text{all}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{all}}$), given that $0 \leq |A| \leq \rho$, and

2. the ideal model $I_{\text{des},\text{fair},\text{cb}}^f$ (with designated party $P_N$), with IT security (i.e. $A = A^{\text{all}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{all}}$), given that $\rho < |A| < \frac{n}{2}$, and

3. the ideal model $I_{\text{abort},\text{cb}}^f$ (with agreement on abort), with computational security (i.e. $A = A^{\text{eff}}$, $\Sigma = \Sigma^{\text{eff}}$, and $D = D^{\text{eff}}$), given that $0 \leq |A| \leq n - \rho$,

from a complete and synchronous network of secure channels, and a broadcast channel, in the stand-alone setting, given that $\rho < \frac{n}{2}$.

It is important to note that $\pi^C$ is a protocol with $n + 1$ parties $P_{1:n}$ and $P_N$, each of which can be corrupted or not. By contrast, the full protocol $\Pi^\rho_f$ is executed only among $n$ parties. Therefore, here we can tolerate $\leq n - \rho$ corrupted parties (last property in Lemma 7.3), and in the full protocol only $< n - \rho$ (last property in Theorem 7.1).

### 7.4.1 Even more modifications of [GMW87]

Given the security requirements for subprotocol $\pi^C$ in Lemma 7.3, now we have to show that they can be fulfilled. To obtain such a protocol, we continue to modify [GMW87], i.e. all modifications described in Sections 3.4.3 and 6.3.1 still apply.

**Robustness.** A protocol is said to be robust when the adversary cannot prevent the honest parties from computing the result, once the input is given. With the n-out-of-n XOR-sharing that we currently use this cannot be achieved. A single party that just stops sending messages would violate robustness. Hence we need a sharing that enables any set of $n - \rho + 1$ parties to reconstruct the input of up to $\rho$ corrupted parties.

On the other hand, we still have to guarantee information-theoretic privacy for the designated party $P_N$ and fairness for all (honest) parties, even in the case where all other shares are compromised. This implies that a reconstruction without $P_N$’s share has to be impossible.
7.4 New requirements for subprotocol πC

This can be summarized in an access structure $\Gamma$ for a sharing scheme with the following properties:

- $\forall M \in \Gamma : |M| \geq n - \rho + 1$ given that $0 \leq \rho < \frac{n}{2}$
- $\forall M \in \Gamma : P_N \in M$

Furthermore the calculations during the MPC should be possible with the same primitives. This can be achieved with a $(2n - \rho)$-out-of-$2n$ polynomial sharing where $P_N$ obtains $n$ shares and each other party $P_i$ obtains a single share. It is easy to see that without the shares belonging to $P_N$ a reconstruction is not possible. On the other hand, any subset of $n - \rho$ parties together with $P_N$ can reconstruct the information, as they possess $n + n - \rho = 2n - \rho$ shares.

**Callback.** A callback consists of two parts: First, a party $P_i$ has to be notified that it is required to provide additional information. This can be achieved with intermediate outputs. That is, at certain points during the computation where additional information might be necessary, the corresponding parties receive a corresponding output.

And second, this party $P_i$ has to provide the information. For that purpose, we can easily apply the same mechanism used to distribute the original input. That means, party $P_i$ commits to this new input, and then distributes a sharing for the committed value.

7.4.2 Proof sketch of Lemma 7.3

In this section we prove that the latest modifications of [GMW87] in the previous section result in a protocol that confirms Lemma 7.3. We refrain from constructing simulators, as the precise arguments (and simulators) easily follow from our high level discussion and the proofs as laid down in [Gol01].

Before we consider the three requirements one by one, note the fact that the correctness of the protocol remains unchanged and that the proof in Section 3.4.4 still holds: All parties are committed to their internal values (with binding or hiding security) and have to provide zero-knowledge proofs (given binding commitments) or arguments (given hiding commitments). This also holds for messages containing shares: Even though not all shares of some value are distributed, every single share is completely determined by the input value and the randomness, to both of which the sender is committed. Hence, all “wrong” messages (i.e. messages without a correct zero-knowledge proof or argument) will be detected by all honest parties and corresponding measures can be taken. During the input phase, an incorrect message either leads to an abort of the protocol (this does not violate robustness, since robustness is only guaranteed after the input phase), or a default value can be used for the offending participant. During the computation and output phases, incorrect messages can be reconstructed for up to $\rho$ offending parties, and the protocol proceeds.

**Requirement 1: Robustness with $0 \leq |A| \leq \rho$ in the information-theoretic scenario with designated party $P_N$.** To prove the first requirement, we basically
7.4 New requirements for subprotocol $\pi^C$

have to show that the new sharing scheme does not violate the privacy of the input of the designated party $P_N$, while achieving robustness for all honest parties.

As explained in Section 7.2 the callback value of a party $P_i$ can only be as private as the normal input of that party $P_i$, because the same mechanism is used to distribute the data. Since here the protocol $\pi^C$ implements the ideal functionality $I_{f}^{des,rob,cb}$, the adversary might obtain callback values of non-designated parties.

It is straightforward to see that the input coming from the designated party $P_N$ remains private even in the information-theoretic scenario. The sharing scheme ensures that $\forall \mathcal{M} \in \Gamma : P_N \in \mathcal{M}$. Hence, the shares of all other parties $P_i \neq P_N$ taken together do not contain any information.

It remains to show that the protocol cannot be aborted by up to $\rho$ corrupted parties. Given that $P_N$ is honest and $t \leq \rho$ parties are corrupted, we have a set $\mathcal{H}$ of honest parties with $|\mathcal{H}| \geq n - \rho + 1$ and $P_N \in \mathcal{H}$. Hence, as soon as a corrupted party stops sending correct messages (this is easily noted due to the use of a broadcast channel), the honest parties can reconstruct its input and proceed with the computation.

**Requirement 2:** Fairness with $\rho < |A| < \frac{n}{2}$ in the information-theoretic scenario with designated party $P_N$. To prove this requirement, we have to show again that the new sharing scheme does not violate the designated party’s $P_N$ privacy, this time while achieving fairness for all honest parties. Again, the privacy of the callback values of non-designated parties is not an issue here.

The argument why $P_N$’s input remains information-theoretically private is the same as for the first requirement. Furthermore, fairness is achieved by the same argument as in Section 6.3.2, where we discussed the corresponding modifications of [GMW87] for fairness.

**Requirement 3:** Security with abort in the computational case. In addition to correctness, here we have to show that the input of all honest parties remains private, given that $0 \leq |A| \leq n - \rho$ and that the designated party $P_N$ may also be corrupted. This follows directly from the chosen sharing scheme. We have that $\forall \mathcal{M} \in \Gamma : |\mathcal{M}| \geq n - \rho + 1$, hence any set of $n - \rho$ or less parties do not possess any information.

In contrast to the first two requirements, here we additionally have to show that the privacy of the callback values is guaranteed. The callback values are an input to the protocol $\pi^C$ and are provided in the same way as the normal input to the protocol, using commitments and sharings. Currently, we are in the computational scenario. Hence, these commitments reveal no information, and all messages are encrypted. It follows that the privacy of the callback values is guaranteed.

This concludes our proof sketch. Now we have a subprotocol $\pi^C$ that fulfills all security requirements as laid down in Lemma 7.3. In the last section we show that when applying this subprotocol $\pi^C$ in the complete protocol $\Pi_{f}^{\rho}$ as described in Section 7.1, we can prove Theorem 7.1, claiming robustness of the complete protocol $\Pi_{f}^{\rho}$. 
7.5 Adapting the proof for protocol $\Pi_f^\rho$

In this section we prove Theorem 7.1, claiming the security of protocol $\Pi_f^\rho$ as described in Section 7.1. Depending on the number of corrupted parties, the security guarantees encompass robustness, fairness, or security with abort, in the stand-alone setting. Apart from the subprotocol $\pi^N$, which remains unchanged, this protocol is based on the special security guarantees provided by the new subprotocol $\pi^C$ as laid down in Lemma 7.3.

Since we changed the original protocol $\Pi_f$ to the new protocol $\Pi_f^\rho$, we have to prove all three properties of Theorem 7.1. However, the modifications are small and the proofs resemble their counterparts in Section 5, where we prove security with abort, and Section 6.4, where we prove fairness. We only have to adapt the complete simulator $\sigma_{\Pi_f}$ to obtain the simulator $\sigma_{\Pi_f^\rho}$.

7.5.1 Case 1: $0 \leq |A| \leq \rho$

In this section we prove the first property of Theorem 7.1, claiming that the protocol $\Pi_f^\rho$ implements the ideal model $I_{\Pi_f^\sec}$ with full information-theoretic security, given that $t \leq \rho$ parties are corrupted, in the stand-alone setting. Again, the protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the IT scenario, we have $A = A^{alt}$, $\Sigma = \Sigma^{alt}$, and $D = D^{alt}$. However, as the simulators $\sigma^C$ and $\sigma^N$ for the subprotocols $\pi^C$ and $\pi^N$ are all in $\Sigma^{alt}$ as demanded in Lemmata 7.3 and 3.1, the resulting simulator $\sigma_{\Pi_f^\rho}$ is as well.

From Lemma 7.3 we have a protocol $\pi^C$ with a simulator $\sigma^C$ for the ideal functionality $I_{\Pi_f^{\des,\rob,\cb}^C}$, such that $\sigma^C_A(A) \circ I_{\Pi_f^{\des,\rob,\cb}^C} \approx A \circ \pi^C_A(R)$ holds. The other simulators $\sigma^N$ and $\sigma^{BC}$ remain the same as in Section 5.1, where we prove IT security with abort in the stand-alone setting.

The complete simulator $\sigma_{\Pi_f^{\sec}}$ is shown in Figure 30. It performs the following steps:

1. Receive output $\{x_i^N, o_i\}_{i \in A}$ from simulator $\sigma^N$, intended for $I_{\Pi_f^{\sec}}^N$
2. For $i \in H$, chose $\tilde{x}_i^C$ and $\tilde{x}_i^N$ at random
3. For these $\tilde{x}_i^N$, compute the commitments $C(\tilde{x}_i^N)$.
4. Give $\{\tilde{x}_i^C, C(\tilde{x}_i^N)\}_{i \in H}$ as input to simulator $\sigma^C$.
5. Receive output $\{x_i^C, c_i\}_{i \in A}$ from simulator $\sigma^C$, intended for $I_{\Pi_f^{\des,\rob,\cb}}^C$
6. Perform the check: $\forall i \in A : \text{OPEN}(c_i, o_i) \equiv x_i^N$ (Correct opening information)
   - [holds] For every triple, where the check holds, use the existing values for $x_i^N$.
   - [fails] For every triple, where the check fails, do a callback with simulator $\sigma^C$ and use the new values for $x_i^N$ instead of the previous ones. Let $x_i^N = 0$ in case the callback is not answered.
7. Calculate the XOR $x_i = x_i^C \oplus x_i^N$ for $i \in \mathfrak{A}$ and provide the result as input for the corrupted parties to the ideal functionality.
8. Forward the output $y$ to simulator $\sigma^C$.

9. The output of simulator $\sigma^{\Pi_f,\text{sec}}$ is the output of simulator $\sigma^C$.

Up to step 5, the simulation presented above is identical to the simulation in Section 5.1.2, and so are the corresponding steps in the protocol execution. Hence, we can focus on the last part and argue why the simulation is indistinguishable from a real execution.

In the real protocol execution, a callback for party $P_i$ takes place whenever the opening information to the commitment $c_i$ is not correct, i.e. when $\text{OPEN}(c_i, o_i) \equiv x_i^N$ does not hold. In this case, party $P_i$ is given the opportunity to provide the correct input for $x_i^N$. If party $P_i$ does not answer, the protocol $\Pi_f$ sets $x_i^N = 0$. The exact same behavior is simulated by step 6 and 7 above: When a corrupted party $P_i$ sends inconsistent information, it is asked to provide a value for $x_i^N$, which is used from that point onwards.

There is another issue concerning the callbacks: In the definition of $\mathcal{I}_{f,\text{des,rob,cb}}$ as provided in Section 7.2.1 the adversary sees all callback values $x_i^N$ provided from honest parties (excluding the designated party $P_N$). In the simulation we cannot provide such values: the simulator $\sigma^{\Pi_f}$ already fixed values $\bar{x}_i^C$. In order to provide indistinguishable values $\bar{x}_i^N$, it would have to know the corresponding correct input value $x_i$, which it does not. However, in the real protocol execution, such callbacks cannot take place either: We assume that only a minority of the parties is corrupted. Thus, the designated party $P_N$ is honest and provides correct input $(x_i^N, o_i)_{i \in [n]}$ which is consistent with the values provided by the honest parties. As a consequence, callbacks can only occur for corrupted parties.

The rest of the simulation is straightforward. As a consequence, the simulation is perfectly indistinguishable from a real protocol execution. Hence, the simulator $\sigma^{\Pi_f,\text{sec}}$ fulfills the
7.5 Adapting the proof for protocol $\Pi_\rho^f$

Figure 31: The simulator for the information-theoretic scenario including fairness and callback.

requirement

$$\text{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y_{\sigma^C}, y_{t+1}, \ldots, y_n)) \leq \epsilon(\kappa)$$

for the class of all distinguishers $D = D_{\text{all}}$, thereby proving the first part of Theorem 7.1.

7.5.2 Case 2: $\rho < |A| < \frac{n}{2}$

Now we prove the second property of Theorem 7.1, claiming that the protocol $\Pi_\rho^f$ implements the ideal model $I_{\text{fair}}^f$ with fairness, with information-theoretic security, given that $\rho < t < \frac{n}{2}$ parties are corrupted, in the stand-alone setting. Again, the protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the IT scenario, we have $A = A_{\text{all}}$, $\Sigma = \Sigma_{\text{all}}$, and $D = D_{\text{all}}$. However, as the simulators $\sigma^C$ and $\sigma^N$ for the subprotocols $\pi^C$ and $\pi^N$ are all in $\Sigma_{\text{eff}}^{\rho}$ as demanded in Lemmata 7.3 and 3.1, the resulting simulator $\sigma^{\Pi_\rho^f}$ is as well.

From Lemma 7.3 we have a protocol $\pi^C$ with a simulator $\sigma^C$ for the ideal functionality $\mathcal{I}^{\text{des,fair,cb}}_C$, such that $\sigma^C(A) \circ \mathcal{I}^{\text{des,fair,cb}}_C \approx A \circ \pi^C(R)$ holds. The other simulators $\sigma^N$ and $\sigma^{BC}$ remain the same as in Section 6.4, where we prove IT security with fairness in the stand-alone setting.

The complete simulator $\sigma^{\Pi_\rho^f,fair}$ is shown in Figure 31. It performs the following steps:

1. Receive output $\{x_i^N, o_i\}_{i \in A}$ from simulator $\sigma^N$, intended for $\mathcal{I}^{\text{sec}}_N$

2. For $i \in S$, chose $\tilde{x}_i^C$ and $\tilde{x}_i^N$ at random
3. For these $\tilde{x}_i^N$, compute the commitments $C(\tilde{x}_i^N)$.
4. Give $\{\tilde{x}_i^C, C(\tilde{x}_i^N)\}_{i \in \mathbb{N}}$ as input to simulator $\sigma^C$.
5. Receive output $\{x_i^C, c_i\}_{i \in \mathbb{N}}$ from simulator $\sigma^C$, intended for $\mathcal{I}_C^{des, fair, cb}$.
6. Perform the CHECK: $\forall i \in \mathcal{A}: OPEN(c_i, o_i) \equiv x_i^N$ (Correct opening information)
   - [holds] For every triple, where the check holds, use the existing values for $x_i^N$.
   - [fails] For every triple, where the check fails, do a callback with $\sigma^C$ and use the new values for $x_i^N$ instead of the previous ones. Let $x_i^N = 0$ in case the callback is not answered.
7. Calculate the XOR $x_i = x_i^C \oplus x_i^N$ for $i \in \mathcal{A}$ and provide the result as input for the corrupted parties to the ideal functionality.
8. Forward the abort flag from simulator $\sigma^C$ to $\mathcal{I}_f^{fair}$.
9. Forward the output $y$ to simulator $\sigma^C$.
10. The output of simulator $\sigma^{\Pi_f^{\rho}, fair}$ is the output of simulator $\sigma^C$.

The indistinguishability of the first 7 steps follows directly from the previous case. Also the arguments concerning the callbacks to honest and corrupt parties hold. However, now the simulator $\sigma^C$ sends an abort flag intended for the ideal functionality $\mathcal{I}_C^{des, fair, cb}$, which has to be handled in the correct way.

In the real protocol execution, the output $y = \perp$ is provided if and only if the subprotocol $\pi^C$ is explicitly aborted by the adversary (i.e. when the simulator $\sigma^C$ sends an abort flag in the simulation). The protocol execution does not stop automatically when input is inconsistent, since now a callback takes place in this case. This is in contrast to the scenario in Section 6.4, where the protocol was aborted as soon as inconsistent input was detected. Otherwise, when the adversary does not abort the subprotocol $\pi^C$, there will always be an output $y \neq \perp$.

The simulation above guarantees the exact same behavior: The consequence of a negative result in the check is a callback, not an abort. On the other hand, the abort flag from the simulator $\sigma^C$ is directly forwarded to the ideal functionality $\mathcal{I}_f^{fair}$.

Hence, the simulator $\sigma^{\Pi_f^{\rho}, fair}$ fulfills the requirement

$$\text{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y_{\sigma^{\Pi_f^{\rho}, fair}, t+1}, \ldots, y_n)) \leq \epsilon(\kappa)$$

for the class of all distinguishers $D = D^{all}$, thereby proving the second part of Theorem 7.1.

7.5.3 Case 3: $\frac{n}{2} \leq \lvert \mathcal{A} \rvert < n - \rho$

Finally we prove the third property of Theorem 7.1, claiming that the protocol $\Pi_f^{\rho}$ implements the ideal model $\mathcal{I}_f^{abort}$ with computational security with agreement on abort, given that $\frac{n}{2} \leq t < n - \rho$ parties are corrupted, in the stand-alone setting. Again, the protocol is based on a complete and synchronous network of secure channels and a broadcast channel. Since we are in the computational scenario, we have $\mathcal{A} = \mathcal{A}^{eff}$, $\Sigma = \Sigma^{eff}$, and $D = D^{eff}$.
From Lemma 7.3 we have a protocol $\pi^C$ with a simulator $\sigma^C$ for the ideal functionality $I_{C}^{\text{abort}, cb}$, such that $\sigma_N^C(A) \circ I_{C}^{\text{abort}, cb} \approx A \circ \pi^C_N(R)$ holds. The other simulators $\sigma^N$ and $\sigma^{BC}$ remain the same as in Section 5.2, where we prove computational security with agreement on abort in the stand-alone setting.

The complete simulator $\sigma^{\Pi_f, \text{abort}}$ is shown in Figure 32. It performs the following steps:

1. For $i \in S$, chose $\hat{x}_i^N$ at random.
2. For these $\hat{x}_i^N$, compute the commitments $C(\hat{x}_i^N)$ and the corresponding opening information $\hat{o}_i$.
3. Give $\{\hat{x}_i^N, \hat{o}_i\}_{i \in S}$ as input to simulator $\sigma^N$.
4. Then, $\sigma^{BC}(\sigma^N(A))$ is given to simulator $\sigma^C$ which is then started.
5. The simulator $\sigma^C$ provides two kinds of outputs intended for $I_{C}^{\text{abort}, cb}$:
   - Output $\{x_i^N, o_i\}_{i \in [n]}$ representing the input of the designated party $P_N$ to protocol $\pi^C$.
   - Output $\{x_i^C, c_i\}_{i \in A}$ representing the input of the corrupted parties to protocol $\pi^C$.
6. Perform the check: $\forall i \in A : \text{OPEN}(c_i, o_i) \equiv x_i^N$ (Correct opening information)
   - [holds] For every triple, where the check holds, use the existing values for $x_i^N$.
   - [fails] For every triple, where the check fails, do a callback with simulator $\sigma^C$ and use the new values for $x_i^N$ instead of the previous ones. Let $x_i^N = 0$ in case the callback is not answered.
7. Calculate the XOR $x_i = x_i^C \oplus x_i^N$ for $i \in A$ and provide the result as input for the corrupted parties to the ideal functionality.
8. Forward the output $y$ to simulator $\sigma^C$.
9. Forward the abort flag from simulator $\sigma^C$ to $I_{f}^{\text{abort}}$.
10. The output of simulator $\sigma^{\Pi_f, \text{abort}}$ is the output of simulator $\sigma^C$.

There is no need anymore to check whether the adversary changed the inputs $x_i^N$ belonging to the honest parties. In contrast to the situation in Section 5.2, the protocol would not be aborted in this case, but a callback would take place and the honest party could replace a faulty value with the correct one. Of course, this is only true as long as the adversary cannot compute incorrect (but consistent) opening information, which holds by assumption. Hence, the simulation is indistinguishable with respect to this part, except with negligible probability.

Furthermore, according to the specification of the ideal functionality $I_{f}^{\text{abort}, cb}$, the callback values provided by honest parties remain private and are not forwarded to the simulator $\sigma^C$. This is in contrast to the first two cases above, where the privacy of callback values belonging to non-designated but honest parties was not protected. As a consequence, we do not have to deal with callbacks to honest parties. On the other hand, the indistin-
Adapting the proof for protocol $\Pi_f$

Again, up to step 5 the simulation is identical to the one in Section 5.2.2 and we can focus here on the last part. Here we do not need to substitute the output $y$ in case some check fails. In the real protocol execution, the adversary may always choose to receive some output $y \neq \perp$. Based on the discussion above that the adversary cannot compute incorrect but consistent opening information for the commitments, with overwhelming probability this result was computed on the correct inputs as provided by the honest parties. Finally, the adversary decides whether to abort or not.

The same holds for the simulation with the only difference that the result is always computed on the correct inputs as provided by the honest parties. Hence, there is only a negligible difference between the output $y$ in simulation and the output in the real protocol execution.

Hence, the simulator $\sigma_{\Pi_f,\text{abort}}$ fulfills the requirement

$$\operatorname{adv}^D((y_A, y_{t+1}, \ldots, y_n), (y_{\sigma_{\Pi_f,\text{abort}}}, y_{t+1}, \ldots, y_n)) \leq \epsilon(\kappa)$$

for the class of efficient distinguishers $D = D^{\text{eff}}$, thereby proving the last part of Theorem 7.1.

This concludes our proof of Theorem 7.1. We have shown how to adapt the original protocol $\Pi_f$ in two ways: we first included fairness (Section 6), which was possible without losing any security guarantees. Now, in this section, we also included robustness. The tradeoff we had to make is also necessary according to [IKLP06, Kat07]. This implies the

Figure 32: The simulator for the computational scenario including callback.
optimaly of our results. As we have stated in Theorem 6.2 for fairness and in Theorem 7.2 for robustness, the same results can be achieved in the UC setting. The proofs go along the same lines of the proofs in the stand-alone setting, and can thus be omitted here.

8 Protocols without broadcast channel

In this section we investigate the implications of dropping the assumption that a broadcast channel or an equivalent setup is given (of course, this only holds for the stand-alone setting). In the UC setting we still allow the setups needed for e.g. commitments. Here we only discuss the implications for the modified protocol $\Pi_f$ (i.e. the one in Section 7 including robustness) since the other scenarios appear as special cases therein.

First, in Section 8.1 we discuss how the ideal broadcast can be replaced with a construction provided in [FHHW03]. After configuring this construction for our needs in Section 8.2, we present the results of this replacement in Section 8.3 and prove them in Section 8.4.

8.1 Replacing the ideal broadcast

When no ideal broadcast channel is given, there are at least two options: Either, one can use protocols that do not require a broadcast channel. However, it is not clear whether protocols without broadcast channel that fulfill all the requirements exist. Or one can replace the ideal broadcast with a protocol implementing a broadcast channel, based on a complete and synchronous network of secure channels. The discussion in this section follows this approach.

The first idea that comes to mind is to use the construction from the [BGW88] broadcast. However, we would then lose any kind of security above $\frac{n}{3}$. This yields nothing new since information-theoretic security for this threshold was already achieved in the same paper. Furthermore, this broadcast is secure including robustness, more than we actually demand.

Instead, we use the broadcast with extended consistency and validity detection provided in [FHHW03], denoted $\text{ExtConsBC}$. This broadcast channel provides information-theoretic security. Hence, we can use it for both the computational and information-theoretic protocols. This simplifies the further discussion.

Furthermore, $\text{ExtConsBC}$ is a two-threshold broadcast: Standard broadcast is achieved when $t \leq t_v$. When $t_v \leq t \leq t_c$, then either broadcast is achieved, or every player learns that there are too many faults. Note that honest players agree on whether or not broadcast is achieved. For the two thresholds we have that $t_v \leq t_c$, and either $t_v = 0$ or $t_v + 2t_c < n$.

Another important fact about $\text{ExtConsBC}$ is that the broadcast channel is established during an independent precomputation. So, either this precomputation succeeds and a broadcast with full resilience (robustness) is subsequently available. Or all honest parties detect the failure and do not start the protocol execution. As a consequence, for the case that $t \leq t_v$ parties are corrupted this transformation preserves all security properties of

```
the protocol based on this broadcast. For the case that $t \leq t_c$ parties are corrupted the transformation still preserves all security properties of the protocol except robustness. [FHHW03] calls this a Detectable Precomputation.

Now, we can replace the ideal broadcast channel with the broadcast channel $ExtConsBC$. The complete protocol $\Pi^f_{\rho}$, including the subprotocols $\pi^C$ and $\pi^N$, can be based on this broadcast channel, and no changes are necessary. It remains to discuss, which security guarantees can be given and for which bounds. This depends on the choice of the parameters $\rho$, $t_v$, and $t_c$, which we analyze in the next section.

8.2 On the choice of $\rho$, $t_v$, $t_c$

When using the broadcast channel $ExtConsBC$ in the protocol presented in Section 7 as explained above, we need to investigate the possible choices for the parameter $\rho$ of the protocol $\Pi^f_{\rho}$. First, it makes no sense to set $\rho > t_v$ since we have no robust broadcast beyond the validity threshold $t_v$ of $ExtConsBC$. Hence, we cannot claim robustness for $t > t_v$ corrupted parties.

On the other hand, when letting $\rho < t_v$, we would lose robustness between $\rho$ and $t_v$. Given an ideal broadcast channel, we could gain security with abort for $t < n - \rho$ corrupted parties. Thus, smaller values of $\rho$ would make sense in that case. Now, given that $t_v > 0$ (otherwise $\rho < t_v$ is not possible), we have that $t_c < \frac{n}{2}$ (this follows from the restriction $t_v + 2t_c < n$ proved in [FHHW03]). Furthermore, for $\rho \leq \frac{n}{2}$ we have $\frac{n}{2} \leq n - \rho$, and thus $t_c < n - \rho$ for any value of $\rho$. Due to the fact that we lose any security guarantees for the broadcast channel beyond the consistency threshold $t_c$ of $ExtConsBC$, the gain that was achievable given an ideal broadcast, cannot be achieved using the broadcast $ExtConsBC$.

As a consequence, the only logical value for $\rho$ is $\rho = t_v < \frac{n}{3}$. Given this fact, in the next section we discuss the possible security guarantees, the complete protocol $\Pi^f_{\rho}$, based on the broadcast $ExtConsBC$, achieves.

8.3 The result

In the previous sections we have seen how the protocol $\Pi^f_{\rho}$ can be based completely on a protocol implementing a broadcast channel, instead of an ideal broadcast. In this section, we present the results that can be obtained with this construction. The corresponding proof is given in the last section.

The result can be split into two parts depending on whether $t_v = 0$ or not. Given that $t_v > 0$ we have that $t_v + 2t_c < n$. In that case we achieve full security as specified by the ideal functionality $F^{sec}_f$ in Section 2.1 for $t \leq t_v = \rho$ corrupted parties. For $t_v \leq t \leq t_c$ corrupted parties, we still obtain fairness, correctness and privacy as specified by the ideal functionality $F^{for}_f$ in Section 6.1.1. Beyond $t_c$ we lose all security. These results are information-theoretic and were already achieved in [FHHW03].

For $t_v = 0$ we have $t_c = n$, i.e. we have a non-robust broadcast for any number of corrupted parties with agreement on abort. In this setting we achieve information-theoretic fairness
as specified by the ideal functionality $\mathcal{I}_f^{\text{far}}$ in Section 6.1.1 for $t < \frac{n}{2}$, and computational security with abort as specified by the ideal functionality $\mathcal{I}_f^{\text{abort}}$ in Section 2.2.1 for $t < n$.

Whereas the first result for $t_v > 0$ was already known from [FHHW03], the computational part of the second result for $t_v = 0$ is due to the special construction of the protocol $\Pi^f_{\rho}$. It shows that the same result as claimed by Chaum in [Cha89], on which this work is based, can be achieved even without a broadcast channel and enhanced with fairness for an honest majority. The results are graphically illustrated in Figure 1.

### 8.4 Proof of security

In this section we prove that the construction of the protocol $\Pi^f_{\rho}$ based on the broadcast channel $\text{ExtConsBC}$ achieves the security guarantees claimed in the previous section. The proof goes along the lines of the result, and hence is also split into two parts depending on the value of $t_v = \rho$. Since we only replaced the underlying broadcast channel, the following discussion refers repeatedly to Section 7 and Theorem 7.1, where we introduced the robust protocol $\Pi^f_{\rho}$ and stated its security.

First we consider the setting where $\rho = t_v > 0$. Given that $t \leq t_v$ parties are corrupted, we have a completely secure broadcast channel. Hence, the situation is identical to the first property of Theorem 7.1 and the same arguments hold. For $t_v \leq t \leq t_c$, we do not have a robust broadcast channel. However, since the broadcast channel is established during a precomputation, either we do obtain a robust broadcast channel, or all honest parties agree on abort. In the first case the situation is once again identical to the second property in Theorem 7.1. In the latter case, fairness, correctness, and privacy are trivially given, since the protocol execution has not yet started.

Finally we consider the case where $t_v = \rho = 0$ and $t_c = n$. Now, either the precomputation results in a fully secure broadcast channel. In this case, we are confronted with the same situation as in the case where $\rho = 0$ in Section 7 (concerning robustness). Or, if the precomputation fails, fairness, privacy and correctness are still achieved because the precomputation provides agreement on abort and is independent from the actual protocol execution.

These arguments conclude our proof of security.

### 9 Conclusions

In this work we formalized the protocol sketched in [Cha89] and proved its security in the stand-alone as well as in the UC setting. In both cases we achieve security with abort, information-theoretically for $t < \frac{n}{2}$, and computationally for $t < n$ corrupted parties. Furthermore we extended the security first to fairness, and second to robustness. Fairness can be achieved information-theoretically until $t < \frac{n}{2}$. For robustness we have to make a tradeoff: We can fix an arbitrary $\rho < \frac{n}{2}$ and then achieve full IT security for $t < \rho$, but lose any security beyond $n - \rho$. All those results are tight with respect to the bounds stated in [Cle86, Kat07, IKLP06].
Finally we dropped the assumption that a broadcast channel is provided. Using [FHHW03] instead, we examined what we still can achieve. The interesting and new result is that we obtain information-theoretic fairness for $t < \frac{n}{2}$ and computational security with abort for $t < n$. That means that Chaum’s result (extended with fairness, though without our robustness extension) can be achieved even without a broadcast channel. According to [Cle86], fairness cannot be achieved without an honest majority. Furthermore, according to [Kat07, IKLP06], when demanding security with abort for any number of corrupted parties, robustness cannot be achieved at all. Hence, this result is tight.

Interesting open questions mainly concern the setups needed for the UC setting. All setups in this work can be used only once. [CDPW07] and independently [HMU] present the notion of global, reusable setups. However, the solutions presented therein could not be applied directly to our work. Both are based on computationally secure signature schemes with a single, publicly know verification key. Since we are interested in information-theoretic security, we would use unconditionally secure pseudo-signatures. Alas, such a scheme does not have a single, but multiple verification keys, and can hence not replace the computationally secure scheme readily.

10 Acknowledgements

I am first and foremost pleased to thank Dominik Raub for his guidance, input, and discussions throughout this work. Not only that his comprehensive knowledge about the subject was a great help for me in countless situations, but he was also anytime willing to share and communicate it. His doors were never closed when a problem emerged.

I would also like to thank Matthias Fitzi for helpful remarks and interesting discussions, especially concerning broadcast protocols.

I am also grateful to Prof. Ueli Maurer for drawing my attention to this field of research in many motivating and inspiring lectures, as well as providing the resources to complete this work.

And last but not least many thanks to my parents Sabine and Gert Lucas for their steady support and encouragement throughout my education.
References


REFERENCES


[HMU] Dennis Hofheinz, Jörn Müller-Quade, and Dominique Unruh. Universally composable zero-knowledge arguments and commitments from signature cards.


A Appendix

A.1 Notation

We use the following notation:

- $X_{1:n} \equiv X_1, \ldots, X_n$
- $[n] = \{1, 2, \ldots, n\}$

Parties:

- $P_i$ denotes a party where $i$ is the index of the party
- $\mathcal{H}$ denotes the set of honest parties
- $A$ denotes the adversary
- $\mathcal{A}$ denotes the set of corrupted parties
- $\mathcal{A}^s$ denotes the class of adversaries where $s \in \{\text{eff}, \text{all}\}$ for the class of efficient or the class of all adversaries, respectively
- $P_N$ denotes the simulated party as introduced in Section 3.3

Protocols:

- $\pi^f$ denotes an arbitrary protocol where $f$ denotes the task the protocol is supposed to perform
- $\pi_i$ denotes an arbitrary protocol machine for protocol $\pi^f$ which is run by party $i$
- $\pi^N$ denotes the IT protocol used to simulate $P_N$
- $\pi_i^N$ denotes a protocol machine for protocol $\pi^N$ which is run by party $i$
- $\pi^C$ denotes the computational protocol used to compute the output
- $\pi_i^C$ denotes a protocol machine for protocol $\pi^C$ which is run by party $i$
- $\Pi_f$ denotes the full protocol where $f$ is the function to be computed

Ideal Functionalities:

- $\mathcal{I}_f^{\text{type}}$ denotes an ideal functionality where $\text{type}$ indicates the type of security guarantee and $f$ indicates which function is to be computed
- $BC$ denotes a broadcast channel
- $\pi_i^v$ denotes a local machine implementing the majority vote which is run by party $i$

Simulators:

- $\sigma_i^\pi$ denotes a simulator for protocol $\pi$ attached to the interface belonging to party $i$
- $\Sigma^s$ denotes a class of simulators where $s \in \{\text{eff}, \text{all}\}$
Distinguishers:
- $D^i$ denotes a distinguisher indexed by $i$
- $D^s$ denotes a class of distinguishers where $s \in \{eff, all\}$

Systems:
- $R$ denotes the underlying communication resources, i.e. a network of secure channels and possibly a broadcast channel

### A.2 Multiplexing the broadcast channel

Alas, for the whole protocol there is generally only a single (standard) broadcast channel available that we have to multiplex for the two subprotocols $\pi^C$ and $\pi^N$. To this end each message is prefixed with a bit indicating to which subprotocol it belongs: a “0” for protocol $\pi^N$, and a “1” for protocol $\pi^C$. Now each party runs a multiplexing algorithm that is connected to $BC$ and the broadcast interfaces of the protocols $\pi^C$ and $\pi^N$ and behaves as follows:

**Case: Receive message with prefix 0.** This is a broadcast message for protocol $\pi^N$ and is only forwarded to the protocol machine $\pi^N_i$.

**Case: Receive message with prefix 1.** This is a broadcast message belonging to protocol $\pi^C$. If the sender is $P_N$, then the multiplexer receives $l \leq n$ messages since corrupted parties may suppress their messages (in that case a default value $\bot$ is used instead). The multiplexer subsequently forwards all messages in one chunk to the protocol machine $\pi^C_i$. Otherwise, it forwards the message to both protocol machines $\pi^C_i$ and $\pi^N_i$.

**Case: Protocol machine $\pi^N_i$ sends a message.** In this case, the multiplexer prefixes a 0 and sends the message to $BC$.

**Case: Protocol machine $\pi^C_i$ sends a message.** In this case, the multiplexer prefixes a 1 and sends the message to $BC$. Furthermore, it stores the message as it is the first of $n$ parts of a broadcast message from the designated party $P_N$ to the rest of the parties.

This construction provides us with a normal broadcast channel $BC$ for $\pi^N$, and a modified broadcast channel $BC'$ for $\pi^C$. The security of the construction is straightforward and we omit the proof. Altogether, this is primarily a technical detail and we will omit it throughout the rest of the discussion.