Optimally Hybrid-Secure MPC

Christoph Lucas, Dominik Raub, and Ueli Maurer

Department of Computer Science, ETH Zurich, Switzerland,
{clucas, raubd, maurer}@inf.ethz.ch

Abstract. Most protocols for multi-party computation (MPC) are secure either against information-theoretic (IT) or against computationally bounded adversaries. Hybrid-secure MPC protocols guarantee different levels of security, depending on the power of the adversary. We present a hybrid-secure MPC protocol that provides an optimal trade-off between IT robustness and computational privacy: For any robustness parameter $\rho < \frac{n}{2}$ we obtain an MPC protocol that is simultaneously IT secure with robustness for up to $t \leq \rho$ actively corrupted parties, IT secure with fairness (no robustness) for up to $t < \frac{n}{2}$ and computationally secure with agreement on abort (no fairness) for up to $t < n - \rho$. Our construction is secure in the universal composability (UC) framework (with broadcast and CRS), and achieves the bounds of Ishai et al. [CRYPTO'06], Katz [STOC'07], and Cleve [STOC'86] on trade-offs between robustness and privacy, and on fairness.

For example, in the special case $\rho = 0$ our protocol simultaneously achieves non-robust MPC for up to $t < n$ corrupted parties in the computational setting (like Goldreich et al. [STOC'87]) while providing security with fairness in the IT setting for up to $t < \frac{n}{2}$ corrupted parties (like Rabin and Ben-Or [STOC'89] though without robustness).

A crucial technique in our construction is player emulation, first suggested by Chaum [CRYPTO'89]. In this work we provide a formal and detailed treatment of emulated players in the UC setting.

Keywords: multi-party computation, information-theoretic security, computational security, hybrid security, robustness, fairness, agreement on abort, universal composability, player emulation.

1 Introduction

1.1 Secure Multi-Party Computation

In [Yao82], Yao introduced multi-party computation (MPC): Given an arbitrary but fixed function f and a set of n mutually distrusting parties, an MPC protocol enables these parties to compute the function f on their inputs securely, even if some of the parties are corrupted by an adversary. This first notion (often called Secure Function Evaluation) has meanwhile been extended to reactive and randomized functionalities.

Security requirements for MPC in the literature (e.g. [Gol01]) include privacy, correctness, robustness, fairness, and agreement on abort. *Privacy* is achieved if the adversary cannot learn more about the honest players' inputs than what is implied by the inputs and outputs of the corrupted players. *Correctness* means that the protocol output equals the intended function value $f(x_1, \ldots, x_n)$ of the inputs or there is no output. Privacy and correctness are the two basic requirements. Possible additional requirements are notions of output guarantees, which we dicuss in order of decreasing strength: A protocol achieves *robustness* if an adversary cannot abort the computation and prevent the honest players from obtaining output. *Fairness* is achieved if the honest parties obtain as much information about the output as the adversary. *Agreement on abort* means that all honest parties detect if one of them aborts (and then generally make no output).

A first general solution to the MPC problem was given by [GMW87], based on computational (CO) intractability assumptions and a broadcast (BC) channel. They achieve security with robustness against $t < \frac{n}{2}$ actively corrupted parties or with agreement on abort against $t < n$ actively corrupted parties as described in [Gol04]. On the other hand [BGW88], and independently [CCD88], presented protocols which are information-theoretically (IT) secure and require no BC channel. However, they prove that security can only be achieved as long as $t < \frac{n}{3}$ parties are corrupted. When no robustness is required (detectable MPC) [FHHW03] or if a BC channel is available [RB89], this bound can be improved to $t < \frac{n}{2}$.

MPC is generally treated in a setting where parties are connected by a complete and synchronous network of secure channels. Additionally an authenticated synchronous BC channel or a public key infrastructure (PKI) may be available. Universally composable (UC) MPC protocols usually also require a common reference string (CRS) [Can01,CF01]. Unless otherwise stated we assume a complete and synchronous network of secure channels, a synchronous and authenticated broadcast channel, and a CRS.

1.2 Hybrid Security

Most MPC protocols are designed to be secure either against IT or against CO adversaries. IT MPC protocols have the disadvantage that only a corrupted minority can be tolerated without compromising security. On the other hand, CO protocols can tolerate any number of corrupted parties if robustness and fairness are not required, but they are based on unproven intractability assumptions. Invalidation of the underlying assumptions generally leads to a complete loss of security, even if only a single party is corrupted.

MPC protocols with hybrid security provide different levels of security, depending on the number of corrupted parties. Thus they allow for graceful degradation of security. Specifically, we discuss a protocol offering IT security in case of few corruptions, but still providing CO security in case of many corruptions.

1.3 Contributions and Related Work

We provide UC secure MPC protocols that combine IT and CO security and allow for flexible trade-offs between security with robustness, fairness, or agreement on abort. For any robustness parameter $\rho < \frac{n}{2}$ we describe an MPC protocol π^ρ that, given a CRS, simultaneously provides IT security with robustness against static adversaries actively corrupting $t \leq \rho$ parties, IT security with fairness (no robustness) for $t < \frac{n}{2}$, and CO security with agreement on abort (no fairness) for $t < n - \rho$. Furthermore, we provide proofs in the UC model as well as the stand-alone model without CRS (for the stand-alone case, see full version).

In [Cha89] Chaum sketches a construction aimed at simultaneously guaranteeing CO privacy for any number of actively corrupted parties and IT privacy, given that only a minority of the parties is corrupted. In contrast to our work, [Cha89] does not discuss fairness or robustness. Furthermore, several critical details are neglected. In fact, correctness is not guaranteed in [Cha89]. A central element of both Chaum's approach and ours is the emulation of a player in one protocol using another protocol. In [HM00], this technique was discussed in the stand-alone setting for perfectly secure MPC and applied to general adversary structures. We contribute a formal treatment of this technique in the UC setting.

Fitzi et al. [FHW04] also combine IT and CO security: Up to a first threshold t_p, the security is IT. Between t_p and a second threshold t_σ IT security is guaranteed conditional on the consistency of the underlying PKI. Finally, between t_σ and T the protocol is as secure as the signature scheme in use. Fitzi et al. show that their notion of hybrid MPC is achievable for $(2T + t_p < n) \wedge (T + 2t_\sigma < n)$, which they prove to be tight.

Another work by Fitzi et al. [FHHW03] improves upon [BGW88,CCD88] in the IT setting when no BC channel is available by allowing for two thresholds t_v and t_c where $t_v = 0$ or $t_v + 2t_c < n$. For $t \leq t_v$ corrupted parties, fully secure MPC is achieved while for $t_v < t \leq t_c$ corrupted parties non-robust (but fair) MPC is accomplished.

Both [FHW04] and [FHHW03] largely focus on a setting without BC channel. When a BC channel is provided our results improve substantially upon those of [FHW04,FHHW03]. As [FHHW03] focuses exclusively on IT MPC and [FHW04] only treats robust MPC, both [FHW04,FHHW03] do not reach beyond $t < \frac{n}{2}$ corrupted parties, nor are they easily extended, whereas we can guarantee CO security with agreement on abort for $t < n - \rho$. In the setting without BC channel and for $\rho > 0$ our results match those of [FHHW03] (which they prove optimal for this case). However, for the special case that $\rho = 0$ (i.e., no robustness is required) our construction achieves IT fairness for $t < \frac{n}{2}$, and CO security with agreement on abort for $t < n$ corrupted parties, which goes beyond [FHHW03].

In [IKLP06] and [Kat07] trade-offs between robust and non-robust MPC are discussed, but only in the CO setting. They show that a protocol which guarantees robustness for up to ρ corrupted players can be secure with abort against at most $n - \rho$ corrupted players, and give CO secure protocols that match these bounds. Our protocol π^ρ is optimal under the bounds of [IKLP06,Kat07] but beyond that also provides IT security for $t < \frac{n}{2}$. Furthermore, we match the bound $t < \frac{n}{2}$ on fairness for general MPC put forth in [Cle86], and the bound $t < \frac{n}{2}$ on IT security in the presence of active adversaries (e.g. [Kil00]).

In conclusion we obtain a flexible and optimal trade-off between IT robustness, IT fairness and CO security with agreement on abort, and give proofs of these properties in the UC setting. On the technical side, we also contribute a treatment of player emulation in the UC setting.

2 Security Definitions and Notations

We follow the Universal Composability (UC) paradigm [PW00,Can01,BPW04][1], which defines a simulation-based security model. The security of a protocol (the real world) is defined with respect to an ideal world, where the computation is performed by a *Trusted Third Party* or *Ideal Functionality* F. Informally, a protocol π achieves security if whatever an adversary can achieve in the real world could also be achieved in the ideal world.

More precisely, let $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ be the set of parties, and define $[n] := \{1, \ldots, n\}$ Then, in the *real world*, there is a given set of resources R (e.g., authentic or secure channels, BC channels, a PKI) and for each honest party P_i a protocol machine π_i is connected to the resources R. Let $\mathcal{H} \subseteq \mathcal{P}$ denote the set of such honest parties. Corrupted parties P_i access the resources directly. Let $\mathcal{A} = \mathcal{P} \setminus \mathcal{H}$ denote the set of corrupted parties. The *ideal world* consists of the ideal functionality F and an ideal adversary (or simulator) S connected to F.

A protocol π achieves security if, for every possible set of corrupted parties $\mathcal{A}$, there is a simulator S such that no environment or distinguisher D can tell the real world and the ideal world apart.[2] For this purpose, the distinguisher directly interacts with either one of the two systems, and in the end outputs a decision bit.

In contrast to [Can01] we use a synchronous communication model with static corruption. We work in the crs-model to avoid the impossibility results of [Can01,CF01], where a common reference string crs drawn from a prescribed distribution is made available to all parties. So, we will generally assume as resources R a common

[1] We follow the UC model of [Can01] in spirit, but do not adhere to the notation of [Can01].

[2] In this model, the adversary is thought of as being part of the distinguisher. Canetti [Can01] shows that this is equivalent since the security definition quantifies over all distinguishers.

reference string crs and a complete network net^n of synchronous secure channels including a synchronous authenticated BC channel.[3] In the full version of this paper we also present results without a crs for the stand-alone setting. In the UC setting though, a correctly chosen crs is a prerequisite to the security of our protocols.[4]

In this model, a strong composition theorem can be proven [PW00,Can01,BPW04]. In other words, UC security states that wherever a protocol π is used, we can indistinguishably replace this protocol by the corresponding ideal functionality F together with an appropriate simulator. This follows from the free interaction between the distinguisher and the system during the execution, which implicitly models that outputs of the system can be used in arbitrary other protocols, even before the execution ends.[5]

Definition 1 (Universally Composable (UC) Security). *A protocol π UC securely implements an ideal functionality F if $\forall \mathcal{A},\ \exists \mathsf{S}_{\mathcal{A}},\ \forall \mathsf{D}:\ |Pr[\mathsf{D}(\mathsf{S}_{\mathcal{A}}(\mathsf{F})) = 1] - Pr[\mathsf{D}(\pi_{\mathcal{H}}(\mathsf{R})) = 1]| \leq \varepsilon(\kappa)$. Here $\varepsilon(\kappa)$ denotes a negligible function in the security parameter κ, F denotes the ideal functionality to be implemented, $\pi_{\mathcal{H}}$ denotes the protocol machines of the honest parties in $\mathcal{H}$, and R denotes the resources available to the protocol machines. If we admit computationally unbounded distinguishers we obtain information-theoretic (IT) security, if we restrict ourselves to efficient distinguishers and simulators we arrive at computational (CO) security.*

We generally only consider efficient simulators, since otherwise, IT security does not imply CO security. We discuss hybrid-secure protocols that provide different security properties depending on the number of corrupted parties and on the computational setting. As such we will use corruption and computational model aware functionalities that exhibit different behavior depending on the number t of corrupted parties and on the computational setting (CO or IT). We will say that a protocol π UC securely implements an ideal functionality F if π securely implements F in both the CO and the IT setting.

We will, in the following, be interested in securely implementing an arbitrary n party functionality F. The only restrictions on functionality F are that it provides an I/O-interface to each of the n parties and notifies the adversary when it receives input from a party P_i. For simplicity, we assume that functionality F is symmetric, i.e., when functionality F makes output, it provides the same output y to *all* participants. This is wlog, since, as shown e.g. in [CDG88], securely implementing an asymmetric functionality, providing a separate output for each party, can IT securely be reduced to securely implementing a symmetric functionality.

We now model implementing a functionality F with subsets of the security properties described in Sec. 1.1, generally at least encompassing privacy and correctness. We describe the following four specific security notions:

Full Security. Computing functionality F with *privacy, correctness and robustness*, which implies all the security notions mentioned above, is modelled by functionality F itself, since, in the setting which we consider, demanding a secure implementation of functionality F already amounts to demanding full security.

Fair Security. Demanding *privacy, correctness and fairness* (which implies agreement on abort) only for functionality F is captured by the ideal functionality $\mathsf{F}^{\mathsf{fair}}$, which operates as follows: $\mathsf{F}^{\mathsf{fair}}$ internally runs F. Any inputs to F are forwarded, as are any messages F may output to the adversary. If F makes an output y, then $\mathsf{F}^{\mathsf{fair}}$ request an output flag $o \in \{0, 1\}$ from the adversary, defaulting to $o = 1$ if the adversary makes no suitable input. Finally, for $o = 1$ functionality $\mathsf{F}^{\mathsf{fair}}$ makes output y to *all* parties, for $o = 0$ it halts.

Abort Security. The functionality F^{ab}, specifying *privacy, correctness and agreement on abort* only, works like $\mathsf{F}^{\mathsf{fair}}$ but forwards output y to the adversary before requesting an output flag.[6]

No Security. The functionality $\mathsf{F}^{\mathsf{noSec}}$ models demanding no security whatsoever: Functionality $\mathsf{F}^{\mathsf{noSec}}$ turns control over to the adversary by forwarding all inputs from the honest parties to the adversary and letting the adversary fix all outputs to honest parties.

[3] In [Can01], resources R are modeled as ideal functionalities available in a hybrid model.

[4] It is possible to minimize the reliance on the crs such that our protocols tolerate an adversarially chosen crs for few corrupted parties by applying techniques from [GK08,GO07] and a $(t, 2t-1)$-combiner for commitments (e.g. [Her05]). However, this construction is beyond the scope of this paper.

[5] This is in contrast to a stand-alone definition of security where the distinguisher is restricted to providing input in the beginning of the computation, and receiving output only at the end.

[6] We could relax the definition further by allowing the adversary to send one output flag for each party, dropping agreement on abort. However, all our protocols will achieve agreement on abort.

As a simulator $\mathsf{S}^{\mathsf{noSec}}$ can use the inputs of honest parties to simulate honest protocol machines, this already proves the following (rather trivial) lemma:

Lemma 1. *Any protocol π UC securely implements the ideal model* $\mathsf{F}^{\mathsf{noSec}}$.

3 Hybrid-secure MPC: The Protocol π^ρ

We present a protocol π^ρ that UC securely implements hybrid-secure MPC from a common reference string crs and a complete n party network net^n (consisting of secure channels and an authenticated n party broadcast channel bc^n).[7] More precisely, given a robustness parameter $\rho < \frac{n}{2}$ and an n party functionality F, protocol π^ρ implements F simultaneously providing IT full security in the presence of up to $t \leq \rho$ actively corrupted parties, IT fair security (no robustness) for $t < \frac{n}{2}$ and CO abort security (no fairness) for $t < n - \rho$. These security requirements are formalized via the ideal functionality F^ρ in Fig. 1.

We construct protocol π^ρ in three steps:

1. We show how to IT securely emulate and integrate an additional party P_0 given an n party network net^n (see Sec. 4). This amounts to emulating the protocol π_0 of party P_0 and an $n+1$ party network net^{n+1}.
2. We exhibit an $n+1$ party MPC protocol $\pi^{\mathsf{des},\rho}$ (see Sec. 5) that has a *designated party property*: Protocol $\pi^{\mathsf{des},\rho}$ is run among the n parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$, and a special, designated party P_0. Fairness and robustness of protocol $\pi^{\mathsf{des},\rho}$ depend centrally on the honesty of the designated party P_0. Furthermore, the designated party P_0 has IT privacy and correctness guarantees. In contrast, the parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ have only CO privacy and correctness guarantees. Such a designated party protocol $\pi^{\mathsf{des},\rho}$ can be obtained by modifying the protocol of [CLOS02] as described in Sec. 5.
   The strong security guarantees of protocol $\pi^{\mathsf{des},\rho}$ for the designated party P_0 are then transferred to the remaining parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ by having them emulate P_0: As long as the emulation is secure (for $t < \frac{n}{2}$), the emulated party P_0 can be regarded as honest and the resulting protocol will have the strong fairness, robustness, and correctness properties which protocol $\pi^{\mathsf{des},\rho}$ exhibits if the designated party P_0 is honest.
3. We provide an input protocol π^{in} (see Sec. 6), that transforms a designated party MPC into a hybrid-secure MPC, exploiting the designated party property.

By the UC theorem, these three steps can be aggregated into a protocol π^ρ that securely realizes hybrid-secure MPC as fomalized by functionality F^ρ. Protocol π^ρ relies on the basic resources needed for the construction above, namely a crs and an n party network net^n.[8] An overview of our construction can be found in Fig. 2.

Functionality F^ρ behaves as specified by the following table. That is, for a given computational setting (CO or IT) and number t of corrupted parties functionality F^ρ behaves exactly like the corresponding functionality listed under behavior below.

Adversarial Power		Behavior	
$t \leq \rho$	CO/IT	F	(implement F with full security)
$\rho < t < \frac{n}{2}$	CO/IT	$\mathsf{F}^{\mathsf{fair}}$	(implement F with fair security)
$\frac{n}{2} \leq t < n - \rho$	CO	F^{ab}	(implement F with abort security)
	IT	$\mathsf{F}^{\mathsf{noSec}}$	(no guarantees)
$n - \rho \leq t$	CO/IT	$\mathsf{F}^{\mathsf{noSec}}$	(no guarantees)

Fig. 1. The ideal functionality F^ρ.

[7] In the full version of our paper we also provide a stand-alone secure protocol π^ρ_{SA} which provides the same guarantees as protocol π^ρ without relying on a crs.

[8] Note that our construction uses multiple instances of the network net^n. However, it is easy to securely implement multiple instances of net^n from a single instance of net^n by multiplexing.

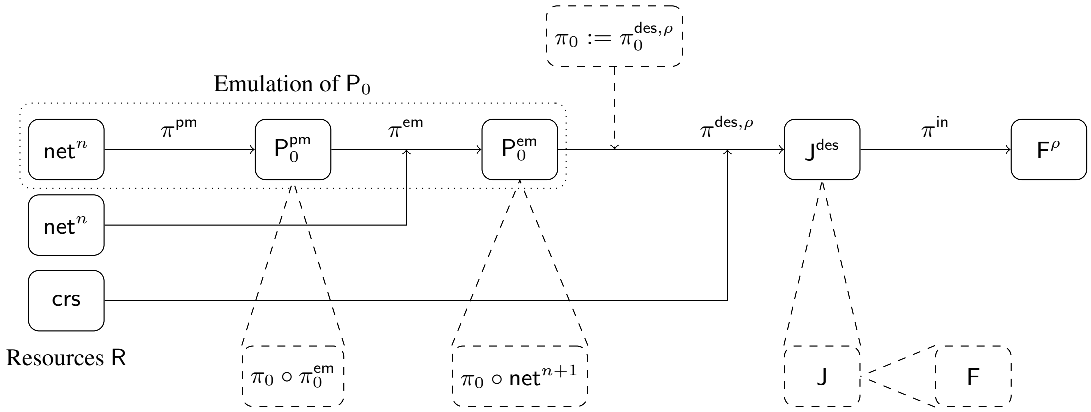

Fig. 2. Protocol π^ρ: The construction of the hybrid MPC functionality F^ρ. Protocol π^ρ is the composition of the protocols π^{pm}, π^{em}, $\pi^{\mathsf{des},\rho}$, and π^{in} in the figure. The dashed boxes beneath the ideal functionalities contain hints on their behavior.

Theorem 1 (UC Security of π^ρ). *Let F be an ideal n party functionality and let $\rho < \frac{n}{2}$ be a robustness parameter. Let a crs setup and a network net^n (encompassing a complete and synchronous network of secure channels and an authenticated BC channel) be given. Protocol π^ρ then implements functionality F^ρ UC securely against static adversaries corrupting any number t of parties. That is, π^ρ implements the ideal functionality F*

1. *with IT full security, given that $t \le \rho$,*
2. *with IT fair security (as formalized by $\mathsf{F}^{\mathsf{fair}}$), given that $t < \frac{n}{2}$, and*
3. *with CO abort security (as formalized by F^{ab}), given that $t < n - \rho$.*

Note that our protocol does not tolerate an adversarially chosen crs, even for $t \le \rho$.[9] However, in our protocol π^ρ the crs is only needed for the perfectly hiding or perfectly binding UC secure commitments of [DN02]. Thus, in the commitment hybrid model we achieve the same security guarantees as above without a crs. CO assumptions sufficient for implementing the necessary CO primitives for protocol π^ρ include the p-subgroup assumption or the decisional composite residuosity assumption [DN02].

In the full version of this paper we also discuss a stand-alone secure variation π^ρ_{SA} of protocol π^ρ which stand-alone securely implements F^ρ *without* reliance on a crs. Protocol π^ρ_{SA} is obtained by substituting subprotocol $\pi^{\mathsf{des},\rho}_{\mathsf{SA}}$ for subprotocol $\pi^{\mathsf{des},\rho}$ in protocol π^ρ. Protocols $\pi^{\mathsf{des},\rho}_{\mathsf{SA}}$ and $\pi^{\mathsf{des},\rho}$ are both discussed in Sec. 5.

4 Emulating a Party

We now describe how to IT securely emulate a party P_0 and its network connection from an n party network net^n. Emulating party P_0 amounts to emulating the protocol machine π_0 which P_0 is supposed to run. Running a given emulated party protocol π_0 on an $n+1$ party network net^{n+1} is formalized by means of the emulation functionality $\mathsf{P}_0^{\mathsf{em}}$ described next.

4.1 The Emulation Functionality $\mathsf{P}_0^{\mathsf{em}}$

Functionality $\mathsf{P}_0^{\mathsf{em}}$ (Fig. 3) formalizes a given emulated party protocol π_0 connected to an $n+1$ party network net^{n+1}. Here, π_0 may be an arbitrary $n+1$ party MPC protocol machine with a communication interface connecting to net^{n+1} and n I/O-interfaces corresponding to the emulating parties $\mathsf{P}_1, \dots, \mathsf{P}_n$. Functionality $\mathsf{P}_0^{\mathsf{em}}$ will then run $\mathsf{P}_0^{\mathsf{em}} = \pi_0 \circ \mathsf{net}^{n+1}$ for $t < \frac{n}{2}$. For $t \ge \frac{n}{2}$, functionality $\mathsf{P}_0^{\mathsf{em}}$ only formalizes a network net^{n+1}, in short $\mathsf{P}_0^{\mathsf{em}} = \mathsf{net}^{n+1}$ for $t < \frac{n}{2}$. We may think of the emulated party P_0 running π_0 as being honest for $t < \frac{n}{2}$ and

[9] It is possible to minimize the reliance on the crs such that our protocols tolerate an adversarially chosen crs for $t \le \rho$ by applying techniques from [GK08,GO07] and a $(t, 2t-1)$-combiner for commitments (e.g. [Her05]). However, this construction is beyond the scope of this work.

corrupted for $t \geq \frac{n}{2}$. This is optimal as IT fully secure general MPC is only achievable for $t < \frac{n}{2}$ corrupted parties [Cle86].

For $t < \frac{n}{2}$, protocol machine π_0 is connected to net^{n+1} via its communication interface. The remaining interfaces (the n I/O-interfaces of protocol π_0 and the n open interfaces to net^{n+1}) constitute the interfaces of functionality $\mathsf{P}_0^{\mathsf{em}}$: Functionality $\mathsf{P}_0^{\mathsf{em}}$ gives each party P_i ($i \in [n]$) access to one I/O-interface to π_0 and one interface to net^{n+1}. For $t \geq \frac{n}{2}$, we have $\mathsf{P}_0^{\mathsf{em}} = \mathsf{net}^{n+1}$ and we consider the emulated party P_0 which is supposed to run protocol machine π_0 as corrupted. Accordingly, the interface of net^{n+1} connected to π_0 for $t < \frac{n}{2}$ is exposed to the adversary for $t \geq \frac{n}{2}$.

Adversarial Power		Behavior	
$t < \frac{n}{2}$	CO/IT	$\pi_0 \circ \mathsf{net}^{n+1}$	(emulated party protocol and $n+1$-network)
$\frac{n}{2} \leq t$	CO/IT	net^{n+1}	($n+1$-network)

Fig. 3. The ideal functionality $\mathsf{P}_0^{\mathsf{em}}$.

4.2 Implementing Functionality $\mathsf{P}_0^{\mathsf{em}}$

As a first step towards implementing the emulation functionality $\mathsf{P}_0^{\mathsf{em}}$ from an n party network net^n, we provide a communication protocol π^{em}, which implements the emulation functionality $\mathsf{P}_0^{\mathsf{em}}$ from an n party network net^n and an intermediate functionality $\mathsf{P}_0^{\mathsf{pm}}$. This functionality $\mathsf{P}_0^{\mathsf{pm}}$ runs the designated party protocol π_0 with a communication protocol π_0^{em} instead of a network net^{n+1} as described below.

Functionality $\mathsf{P}_0^{\mathsf{pm}}$ Functionality $\mathsf{P}_0^{\mathsf{pm}}$ runs (for $t < \frac{n}{2}$, see Fig. 4) the emulated party protocol π_0 and a communication protocol π_0^{em}. All interfaces of protocol π_0 (the n I/O-interfaces and the communication interface) are connected to protocol π_0^{em}. The communication protocol π_0^{em} in turn provides n I/O-interfaces corresponding to the emulating parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ which become the n I/O-interfaces of functionality $\mathsf{P}_0^{\mathsf{pm}}$. The concrete communication protocol π_0^{em} we use is described with the protocol π^{em} below.

Adversarial Power		Behavior	
$t < \frac{n}{2}$	CO/IT	$\pi_0 \circ \pi_0^{\mathsf{em}}$	(emulated party protocol)
$\frac{n}{2} \leq t$	CO/IT	$\mathsf{F}^{\mathsf{noSec}}$	(no guarantees)

Fig. 4. The ideal functionality $\mathsf{P}_0^{\mathsf{pm}}$.

Protocol π^{em} The communication protocol π^{em} (Fig. 6) implements a network net^{n+1} connected to the designated party protocol π_0 (functionality $\mathsf{P}_0^{\mathsf{em}}$, Fig. 3) from a communication protocol π_0^{em} connected to the designated party protocol π_0 (functionality $\mathsf{P}_0^{\mathsf{pm}}$, Fig. 4) and a network net^n.

The communication protocol machines π_0^{em} (run by functionality $\mathsf{P}_0^{\mathsf{pm}}$) and π_i^{em} ($i \in [n]$) are designed to interact with each other. Each protocol machine π_i^{em} connects to the I/O-interface of P_i to $\mathsf{P}_0^{\mathsf{pm}}$ and to the interface of net^n to P_i. In turn it provides P_i with a communication interface (to the emulated net^{n+1}) and with an I/O-interface (to the emulated π_0). Recall that $\mathsf{P}_0^{\mathsf{pm}}$ exposes the I/O-interfaces of π_0^{em} as its own I/O-interfaces, as such the π_i^{em} connect directly to the I/O-interfaces of π_0^{em} (for $t < \frac{n}{2}$). Protocol π_0^{em} operates as detailed in Fig. 5, the π_i^{em} ($i \in [n]$) are described in Fig. 6 below.

Protocol π^{em} emulates net^{n+1} by making use of the fact that the parties P_i ($i \in [n]$) emulating P_0 are the same parties P_i that are supposed to interact with P_0 via net^{n+1}. The I/O-interface of P_i to $\mathsf{P}_0^{\mathsf{pm}}$ can therefore serve as a secure channel between P_i (running π_i^{em}) and the emulated P_0 (running π_0^{em}). Protocol π^{em} then integrates P_0 into the network net^n which is available as a resource by forwarding messages to $\mathsf{P}_0^{\mathsf{pm}}$ (i.e. to π_0^{em}) by means of its

I/O-interfaces. Messages, inputs, and outputs between P_i and the emulation of P_0 can directly be forwarded in this fashion. As the emulated party P_0 is only expected to honestly run π_0 for $t < \frac{n}{2}$, the broadcast bc^n available from the resource net^n can be extended to a broadcast bc^{n+1} by having each π_i^{em} act as a forwarder and performing a majority vote.

π_0^{em} connects to *all* interfaces of protocol π_0, that is, to the communication interface and to the n I/O-interfaces corresponding to the parties $\mathsf{P}_1, \dots, \mathsf{P}_n$. π_0^{em} provides n I/O-interfaces of its own to the n parties $\mathsf{P}_1, \dots, \mathsf{P}_n$. π_0^{em} then processes messages as follows:

Secure Channels: Inputs on the I/O-interface of P_i to π_0^{em} that are labeled as messages from P_i are forwarded to the communication interface of π_0 as messages from P_i. Messages for P_i from the communciation interface of π_0 are labeled as message from P_0 and output on the I/O-interface of P_i to π_0^{em}.

Broadcasts: Inputs labeled as broadcast messages from P_i, are forwarded to the communication interface of π_0 as broadcast messages from P_i if received identically at more than $\frac{n}{2}$ I/O-interfaces of π_0^{em} or otherwise ignored. Broadcast messages from the communciation interface of π_0 are labeled as broadcast messages from P_0 and output on all I/O-interfaces of π_0^{em}.

I/O for π_0: Unlabeled inputs from the I/O-interface of P_i to π_0^{em} are forwarded to the protocol machine π_0 as input on the I/O-interface of P_i. Outputs from π_0 on the I/O-interface of P_i are output on the I/O-interface of P_i to π_0^{em}.

Fig. 5. The protocol machine π_0^{em}.

π_i^{em} connects to the interfaces of net^n and $\mathsf{P}_0^{\mathsf{pm}}$ belonging to P_i and offers P_i a communication interface to the emulated net^{n+1} and an I/O-interface to π_0. π_i^{em} then processes messages as follows:

Secure Channels: Messages for P_j arriving on the communication interface are forwarded to P_j via net^n. Messages for P_0 arriving on the communication interface are labeled as messages from P_i and forwarded to the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$. Inputs from the the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$ labeled as messages from P_0 are forwarded to the communication interface as messages from P_0. Messages from P_j arriving on the interface to net^n are forwarded to the communication interface as messages from P_j.

Broadcasts from $\mathsf{P}_1, \dots, \mathsf{P}_n$: Broadcast messages arriving on the communication interface are forwarded to net^n as broadcast messages and to the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$ labeled as broadcasts from P_i. Broadcast messages from a P_j arriving on the interface to net^n, unless labeled as originating from P_0, are forwarded both to the communication interface as broadcast messages from P_j and to the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$, labeled as broadcast from P_j.

Broadcasts from P_0: Inputs from the the I/O-interface of π_i^{pm} labeled as broadcast messages from P_0 are forwarded as broadcast messages to net^n, including the label. Broadcast messages arriving on the interface to net^n labeled as originating from P_0 are forwarded to the communication interface as broadcast message from P_0 if received identically from more than $\frac{n}{2}$ parties P_j, including the copy possibly received from $\mathsf{P}_0^{\mathsf{pm}}$ directly.

I/O for P_0: Inputs from the I/O-interface are forwarded to the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$. Unlabeled inputs from the I/O-interface of $\mathsf{P}_0^{\mathsf{pm}}$ are output on the I/O-interface.

Fig. 6. The protocol machine π_i^{em}.

Lemma 2. *Protocol π^{em} UC securely implements $\mathsf{P}_0^{\mathsf{em}}$ from net^n and $\mathsf{P}_0^{\mathsf{pm}}$ against static adversaries.*

A proof of Lem. 2 can be found in App. A.

4.3 Implementing Functionality $\mathsf{P}_0^{\mathsf{pm}}$

To complete the emulation of a party P_0, it remains to IT securely implement the functionality $\mathsf{P}_0^{\mathsf{pm}}$ (Fig. 4) from a network net^n by means of a protocol π^{pm}. Recall that functionality $\mathsf{P}_0^{\mathsf{pm}}$ runs the designated party protocol π_0 and the communication protocol π_0^{em} for $t < \frac{n}{2}$, exposing one I/O-interface to each party P_i ($i \in [n]$). For $t \geq \frac{n}{2}$, functionality $\mathsf{P}_0^{\mathsf{pm}}$ turns control over to the adversary.

Any such ideal functionality $\mathsf{P}_0^{\mathsf{pm}}$ can be realized using an MPC protocol π^{pm} that, for $t < \frac{n}{2}$, provides full security in the IT setting. The existence of such a protocol is guaranteed by the following lemma taken from [RB89,Can01]:

Lemma 3 ([RB89,CDD⁺99,Can01]). *Given a (well-formed [Can01]) ideal functionality* F *there is a protocol* π^{pm} *that implements the ideal functionality* F *with IT full security from a complete and synchronous network of secure channels and a BC channel in the UC setting, against static adversaries corrupting* $t < \frac{n}{2}$ *parties.*

5 Implementing a Designated Party MPC $\mathsf{J}^{\mathsf{des}}$

We exhibit a designated party MPC protocol $\pi^{\mathsf{des},\rho}$ which implements an $n+1$ party designated party MPC from a common reference string crs and an $n+1$ party network net^{n+1}. For our purposes, the designated party P_0 (running protocol $\pi_0^{\mathsf{des},\rho}$ as specified below) will be emulated as described in Sec. 4. More formally, we provide a protocol $\pi^{\mathsf{des},\rho}$ that implements the designated party MPC functionality $\mathsf{J}^{\mathsf{des}}$ from a crs and the emulation functionality $\mathsf{P}_0^{\mathsf{em}}$, running the emulated party protocol $\pi_0 = \pi_0^{\mathsf{des},\rho}$.

5.1 Functionality $\mathsf{J}^{\mathsf{des}}$

We define a designated party MPC functionality $\mathsf{J}^{\mathsf{des}}$ that formally captures computing a functionality J with the designated party property.

Functionality $\mathsf{J}^{\mathsf{des}}$ takes as parameter an arbitrary $n+1$ party functionality J which has $2n$ interfaces. Here, each party P_i ($i \in [n]$) has one I/O-interface to J and a second I/O-interface formally belonging P_0, but available to P_i. We refer to this second interface as the P_0-interface of P_i to J. Functionality $\mathsf{J}^{\mathsf{des}}$ then evaluates functionality J, providing the following guarantees: In case the designated party P_0 is honest, functionality $\mathsf{J}^{\mathsf{des}}$ guarantees privacy of P_0's input, correctness, and fairness against arbitrarily many IT corrupted parties, as well as robustness against $t \leq \rho$ IT corrupted parties. In case the designated party P_0 is corrupted, functionality $\mathsf{J}^{\mathsf{des}}$ still guarantees correctness and privacy to the honest parties against $t < n - \rho$ CO corrupted parties.[10] Recall that, by design of the designated party functionality $\mathsf{P}_0^{\mathsf{em}}$, we think of the emulated party P_0 as honest for $t < \frac{n}{2}$ and corrupted for $t \geq \frac{n}{2}$. Keeping this in mind we arrive at the functionality $\mathsf{J}^{\mathsf{des}}$ described in Fig. 7.

Functionality $\mathsf{J}^{\mathsf{des}}$ models computing a functionality J with the designated party property. Like functionality J, functionality $\mathsf{J}^{\mathsf{des}}$ provides one I/O-interface and one P_0-interface to each party P_i ($i \in [n]$). Functionality $\mathsf{J}^{\mathsf{des}}$ operates as follows:[10]

1. If P_0 is corrupted ($t \geq \frac{n}{2}$), and addtionally we are in the IT setting or $t \geq n - \rho$, turn control over to the adversary by running functionality $\mathsf{F}^{\mathsf{noSec}}$. Otherwise, run functionality J.
2. Forward any inputs from the I/O-interfaces to J and, in the IT setting, copy them to the adversary.
3. If P_0 is honest ($t < \frac{n}{2}$), forward any inputs from the P_0-interfaces to functionality J, if P_0 is corrupted ($t \geq \frac{n}{2}$), expose all P_0-interfaces of functionality J directly to the adversary, and forward any inputs of honest parties to P_0-interfaces to the adversary.
4. If functionality J makes output:
   (a) If P_0 is honest ($t < \frac{n}{2}$) and $t \leq \rho$ set $o := 1$.
   (b) Elsif P_0 is honest ($\rho < t < \frac{n}{2}$) request an output flag $o \in \{0,1\}$ (default to $o = 1$) from the adversary.
   (c) Elsif P_0 is corrupted ($\frac{n}{2} \leq t < n - \rho$, in the CO setting) make output to the adversary and take an output flag $o \in \{0,1\}$ (default to $o = 1$) as input from the adversary.
   (d) If $o = 1$ forward the remaining outputs, otherwise ($o = 0$) halt.
5. Any messages from J to the adversary are forwarded.

Fig. 7. The ideal functionality $\mathsf{J}^{\mathsf{des}}$.

Functionality $\mathsf{J}^{\mathsf{des}}$ thus computes the $n+1$ party functionality J with properties as summarized in Table 1.

[10] The number t of corrupted parties always pertains to the real parties $\mathsf{P}_1, \ldots, \mathsf{P}_n$ and never includes the emulated party P_0.

[11] Correctness is maintained in the sense that the ideal functionality still performs the desired computation. However, the adversary may make inputs dependent on the inputs of honest parties in the current and previous input phases.

[12] Our protocol $\pi^{\mathsf{des},\rho}$ actually achieves correctness here in the sense that it still performs the desired computation. However, the adversary may make inputs dependent on the state of the protocol, i.e. on the inputs of honest parties in previous but not the current input phases. For our subsequent results though, we need not demand correctness here.

Adversarial Power[10]		Guarantees				
IT/CO	t	Cor.	Priv. P_0	Priv. P_i	Fair.	Rob.
IT	$t \leq \rho$	yes[11]	yes	no	yes	yes
IT	$\rho < t < \frac{n}{2}$	yes[11]	yes	no	yes	yes
IT	$\frac{n}{2} \leq t$	no	n/a (corrupted)	no	no	no
CO	$t \leq \rho$	yes	yes	yes	yes	yes
CO	$\rho < t < \frac{n}{2}$	yes	yes	yes	yes	no
CO	$\frac{n}{2} \leq t < n - \rho$	yes	n/a (corrupted)	yes	no	no
CO	$n - \rho \leq t$	no[12]	n/a (corrupted)	no	no	no

Table 1. Security Guarantees formalized by functionality $\mathsf{J}^{\mathsf{des}}$.

5.2 Protocol $\pi^{\mathsf{des},\rho}$

We now describe a designated party MPC protocol $\pi^{\mathsf{des},\rho}$ which implements an $n+1$ party designated party MPC from a common reference string crs and an $n+1$ party network net^{n+1}. We then emulate party P_0, having the emulation functionality $\mathsf{P}_0^{\mathsf{em}}$ run protocol machine $\pi_0^{\mathsf{des},\rho}$ and provide the network net^{n+1}. As a result, protocol $\pi^{\mathsf{des},\rho}$ will implement the designated party MPC functionality $\mathsf{J}^{\mathsf{des}}$ from a crs and emulation functionality $\mathsf{P}_0^{\mathsf{em}}$, running the emulated party protocol $\pi_0 = \pi_0^{\mathsf{des},\rho}$. We obtain protocol $\pi^{\mathsf{des},\rho}$ by adapting the CO MPC protocol of [CLOS02] to our needs ([CLOS02] is in turn an adaption of [GMW87] to the UC setting. For the stand-alone setting we may directly adapt [GMW87] obtaining a stand-alone protocol $\pi_{\mathsf{SA}}^{\mathsf{des},\rho}$.).

Modifying [CLOS02] Before we adapt the protocols of [CLOS02,GMW87] to satisfy the requirements laid out in Sec. 5.1 we first give an overview of [CLOS02,GMW87]:

The protocols in [CLOS02,GMW87] proceed in stages, each consisting of three phases: an input phase, a computation phase, and an output phase. If the functionality to be implemented is non-interactive, a single stage suffices; interactive functionalities require several stages. In the input phase the players commit to their inputs and share them among the participants according to a prescribed secret sharing scheme. In [CLOS02] this is a simple XOR n-out-of-n sharing, but as described in [Gol04] a different sharing can be used to trade privacy for robustness. In the computation phase, [CLOS02,GMW87] use oblivious transfer (OT) to evaluate the desired function on the inputs. All intermediate results are computed as sharings, where the parties commit to their share and prove it correct using zero-knowledge (ZK) proofs thus achieving security against active adversaries. In the output phase the results of the computation are reconstructed by the players by opening their commitments to the shares of the final result.

The security requirements of Sec. 5.1 for protocol $\pi^{\mathsf{des},\rho}$ can be grouped into four cases:

1. In the CO case where P_0 is honest we require full security for $t \leq \rho$ and tolerate only the loss of robustness beyond that bound.
2. In the CO case where P_0 is corrupted, we only require privacy and correctness up to $t < n - \rho$.
3. In the IT case where P_0 is honest, we require correctness, privacy for P_0, fairness, and robustness for up to $t \leq \rho$. Again, we tolerate the loss of robustness beyond that bound.
4. In the IT case where P_0 is corrupted, we do not require any security guarantees.

We show that the MPC protocols of [CLOS02,GMW87] can be modified accordingly. Like [GMW87] the MPC protocol in [CLOS02] operates on shares, utilizes oblivious transfer (OT) for multiplications, and uses the compiler of [GMW87] which is based on commitments and zero-knowledge (ZK) proofs to achieve security against active adversaries. We demonstrate how these components can be modified to provide additional guarantees without compromising their original security properties.

Modifying the Computational Primitives We need to modify [CLOS02,GMW87] such that privacy and correctness are IT for player P_0. All three CO primitives employed in [CLOS02,GMW87] (i.e. OT, commitments, and ZK proofs) can be implemented CO securely while IT protecting one (in our case always P_0) of the participants. That is, we can implement [CLOS02,GMW87] using primitives that remain secure if P_0 is involved in their computation and honest, even if arbitrarily many other players P_i are IT corrupted.

This serves our purpose: Using such primitives is merely a refinement of [CLOS02,GMW87], thus the resulting $n+1$ party protocol is still correct and private in presence of arbitrarily many actively corrupted parties in the CO setting. Furthermore, given these modifications, the protocol is private for player P_0 even in presence of arbitrarily many IT corrupted parties. Finally, as long as player P_0 is honest, the protocol is correct in the IT setting.

A more detailed discussion of suitable CO primitives for [CLOS02,GMW87] which IT protect the designated party P_0 can be found in App. B

Modifying Sharing and Output Reconstruction We now describe how to modify the sharing and output reconstruction underlying [CLOS02,GMW87] in order to meet our robustness and fairness requirements.

We have to robustly tolerate up to $t \leq \rho$ corruptions among the parties P_i ($i \in [n]$), while preserving the unconditional privacy of P_0. This can be accomplished by modifying the underlying sharing of [CLOS02,GMW87] as described in [Gol04, Sec. 7.5.5]. We use a sharing where any set M of $n-\rho+1$ parties that *includes* P_0 is qualified, i.e. can reconstruct. Such a sharing can efficiently be implemented using a $(2n-\rho)$-out-of-$(2n)$ Shamir-sharing where P_0 receives n shares and each remaining party P_i obtains a single share. Here, we inherently trade privacy for robustness: Any qualified set M of parties can reconstruct the input of the remaining parties. So any qualified set M of honest parties can recover the input of up to ρ corrupted parties P_i ($i \in [n]$). This ensures robustness, should up to $t \leq \rho$ corrupted parties try to disrupt the computation. On the other hand, any such qualified set M of corrupted parties can violate the privacy of the remaining parties.

Finally we have to guarantee fairness whenever P_0 is honest. As noted in [Gol04] only the player opening his commitments last in the output phase can violate fairness. If we specify that P_0 should open last, and only if he can contribute sufficiently many shares that all players can reconstruct the result, then the resulting protocol $\pi^{\mathsf{des},\rho}$ is fair in the IT setting as long as P_0 is honest.

5.3 The Security of the Designated Party Protocol $\pi^{\mathsf{des},\rho}$

In summary, we have constructed a protocol $\pi^{\mathsf{des},\rho}$ from [CLOS02] securely implementing $\mathsf{J}^{\mathsf{des}}$ in the UC setting:

Lemma 4. *For any robustness parameter $\rho < \frac{n}{2}$ there is a protocol $\pi^{\mathsf{des},\rho}$ that implements $\mathsf{J}^{\mathsf{des}}$ UC securely against static adversaries from a* crs *setup and an emulation functionality $\mathsf{P}_0^{\mathsf{em}}$ running $\pi_0^{\mathsf{des},\rho}$ as designated party protocol.*

Furthermore, we have constructed a protocol $\pi_{\mathsf{SA}}^{\mathsf{des},\rho}$ from [GMW87] securely implementing $\mathsf{J}^{\mathsf{des}}$ in the stand-alone setting, *without* reliance on a crs.

A proof-sketch of Lem. 4 can be found below. CO assumptions sufficient for implementing the necessary CO primitives for protocol $\pi^{\mathsf{des},\rho}$, in particular perfectly hiding or perfectly binding UC secure commitments [DN02], are for instance the p-subgroup assumption or the decisional composite residuosity assumption. For the stand-alone setting, weaker assumptions, e.g. enhanced trapdoor one-way permutations are sufficient [Gol04]. A similar approach, where *all* players use primitives that IT disclose no undesired information is used in [KMQR09] to achieve long-term security for specific functions.

A proof sketch for Lem. 4 can be found in App. C.

6 Implementing a Hybrid-Secure MPC F^ρ

It remains to provide a protocol π_i^{in} implementing a hybrid-secure MPC functionality F^ρ (Fig. 1) from the designated party MPC functionality $\mathsf{J}^{\mathsf{des}}$. Protocol π^{in} does so by ensuring IT privacy for $t < \frac{n}{2}$ and CO privacy for $t < n-\rho$. This is achieved by having π_i^{in} share any input x_i as $x_i = x_i^{\mathsf{em}} \oplus x_i^{\mathsf{des}}$, where x_i^{em} is chosen uniformly at random over the input space.[13] Protocol π_i^{in} then inputs x_i^{des} at the I/O-interface of functionality $\mathsf{J}^{\mathsf{des}}$,

[13] Wlog, we assume a group structure with operation $\oplus$ over the input space. Assuming inputs from a finite field is a common convention in MPC, or we may think of bitstrings, with XOR as operation.

while entering the share x_i^{em} via the P_0-interface of functionality $\mathsf{J}^{\mathsf{des}}$. As functionality $\mathsf{J}^{\mathsf{des}}$ guarantees IT privacy for P_0, this results in a protocol where privacy is IT as long as P_0 is honest, i.e. for $t < \frac{n}{2}$. At the same time the CO privacy of all parties is guaranteed by functionality $\mathsf{J}^{\mathsf{des}}$ for $t < n - \rho$. Hence, we obtain a protocol with CO privacy for $t < n - \rho$.

To maintain CO correctness for $t \geq \frac{n}{2}$ (when the emulated party P_0 is corrupted) additional measures are needed: For $t \geq \frac{n}{2}$, functionality $\mathsf{J}^{\mathsf{des}}$ turns the P_0-interface over to the adversary, who could manipulate the x_i^{em} at will, effectively manipulating the inputs x_i to produce *incorrect* results. We solve this problem by using commitments. So, in the following let commit and open denote the respective procedures for a UC secure IT hiding commitment scheme (see [DN02], App. E). Then, π_i^{in} may compute an IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{em}})$ to x_i^{em}. Protocol π_i^{in} inputs the commitment c_i together with x_i^{em} at the P_0-interface of functionality $\mathsf{J}^{\mathsf{des}}$ while entering the matching opening information o_i togther with x_i^{des} at the I/O-interface of functionality $\mathsf{J}^{\mathsf{des}}$. We then have functionality $\mathsf{J}^{\mathsf{des}}$ check these commitments. In case a commitment fails to open correctly, we can abort the computation. This construction achieves CO correctness because a CO adversary controlling the P_0-interfaces cannot open such a commitment incorrectly. At the same time, the unconditional privacy of the x_i^{em} is unaffected as the commitments c_i are IT hiding.

Finally, we need to guarantee robustness for $t \leq \rho$. Thus, we may not abort if a commitment c_i fails to open correctly. Instead, the functionality $\mathsf{J}^{\mathsf{des}}$ outputs a complaint, requesting that P_i directly inputs x_i via the I/O-interface of $\mathsf{J}^{\mathsf{des}}$. This procedure does not affect privacy since commitments c_i only fail to open correctly if either P_i is corrupted or if the emulated party P_0 is controlled by the adversary. In the first case, we need not guarantee privacy to P_i. In the latter case, we have $t \geq \frac{n}{2}$, so we only need to guarantee CO privacy, which $\mathsf{J}^{\mathsf{des}}$ already does. Correctness is maintained since privacy is maintained and a party can only replace its own input.

The fairness properties of $\mathsf{J}^{\mathsf{des}}$ are unaffected by the measures described above, so the resulting protocol is fair whenever the emulated party P_0 is honest, i.e. whenever $t < \frac{n}{2}$.

Summarizing the measures above, we obtain an input protocol π^{in} (Fig. 8) and a matching functionality J to be run by functionality $\mathsf{J}^{\mathsf{des}}$. Protocol π^{in} takes care of sharing inputs, providing commitments and answering complaints. Functionality J reconstructs the inputs, checks commitments, makes complaints, and finally evaluates the target functionality F.

Protocol machine π_i^{in} connects to the I/O- and P_0-interfaces of P_i to functionality $\mathsf{J}^{\mathsf{des}}$. In turn π_i^{in} offers an I/O-interface to P_i. Protocol machine π_i^{in} then proceeds as follows:

1. On receiving an input on the I/O-interface: Choose x_i^{em} uniformly at random and compute $x_i^{\mathsf{des}} := x_i \oplus x_i^{\mathsf{em}}$. Using an IT hiding commitment scheme compute $[c_i, o_i] = \mathsf{commit}(x_i^{\mathsf{em}})$. Pass input (x_i^{em}, o_i) to the P_0-interface and $(x_i^{\mathsf{des}}, c_i)$ to the I/O-interface of $\mathsf{J}^{\mathsf{des}}$. Receive a complaint vector $\boldsymbol{e}$ on the I/O-interface of $\mathsf{J}^{\mathsf{des}}$. If $e_i = 0$ then input x_i to the I/O-interface of $\mathsf{J}^{\mathsf{des}}$.
2. On receiving an output on the I/O-interface of $\mathsf{J}^{\mathsf{des}}$, forward y to the I/O-interface to P_i.

Fig. 8. The protocol machine π_i^{in}.

Functionality J connects to the n I/O-interfaces of functionality F and in turn provides one P_0-interface and one I/O-interface per party P_i. Functionality J then proceeds as follows:

1. Run functionality F.
2. On receiving input on an I/O-interface in a given round: Parse inputs on the I/O-interfaces of the P_i as $(x_i^{\mathsf{des}}, c_i)$. Parse inputs on the P_0-interfaces of the P_i as (x_i^{em}, o_i). Output a complaint vector $\boldsymbol{e} = (x_i^{\mathsf{em}} \stackrel{?}{=} \mathsf{open}(c_i, o_i))_{i \in [n]}$ via the I/O-interfaces of the P_i. For all $i \in [n]$ where $e_i = 1$ compute $x_i := x_i^{\mathsf{des}} \oplus x_i^{\mathsf{em}}$. Take new inputs x_i on the I/O-interfaces of P_i where $e_i = 0$, default to $x_i = \perp$ if no input is provided. Forward all inputs $x_i \neq \perp$ to functionality F.
3. On receiving an output y from F, forward y to the I/O-interfaces of the P_i.
4. Forward any messages from F to the adversary.

Fig. 9. The functionality J.

Lemma 5. *For any robustness parameter $\rho < \frac{n}{2}$ protocol π^{in} UC securely implements F^{ρ} against static adversaries from a designated party MPC functionality $\mathsf{P}_0^{\mathsf{em}}$ running functionality* J.

A proof Lem. 5 can be found in App. D.

7 Protocols Without Broadcast Channel

We now describe what can be achieved without assuming a BC channel. As our protocol relies on a BC channel, we have to implement one from pairwise secure channels. We make use of the IT secure BC with extended consistency and validity detection $\mathsf{bc}_{\mathsf{extCons}}$ of [FHHW03]. For two thresholds t_v and t_c, where $t_v \leq t_c$ and either $t_v = 0$ or $t_v + 2t_c < n$, $\mathsf{bc}_{\mathsf{extCons}}$ delivers a robust BC for $t \leq t_v$ and a BC with fairness (but without robustness) for $t_v < t \leq t_c$. Actually, $\mathsf{bc}_{\mathsf{extCons}}$ performs a *detectable precomputation* which either establishes a setup for a robust BC (for $t \leq t_v$ always) or aborts with agreement on abort.

For a robustness bound $\rho > 0$ we let $t_v = \rho < \frac{n}{3}$ and $t_c = \lceil \frac{n-t_v}{2} \rceil - 1$. This achieves IT full security (with robustness) for $t \leq \rho$ and IT fair security (no robustness) for $t < \frac{n-\rho}{2}$. Unfortunately these results do not (and cannot) go beyond those of [FHHW03] which they have proven optimal for this case.

However, for robustness bound $\rho = 0$, we let $t_v = \rho = 0$ and $t_c = n$. In this case we achieve IT fair security (no robustness) for $t < \frac{n}{2}$ and CO abort security for $t < n$. This result is new and actually matches the result for $\rho = 0$ according to Thm. 1 in the case where a BC channel is provided. We refer to π^{ρ} for $\rho = 0$, running with the above BC implementation as π^0 and have:

Theorem 2. *Let F be an ideal n party functionality and let $\rho < \frac{n}{2}$ be a robustness parameter. Let a crs setup and a complete and synchronous network of secure channels (*without *BC channel) be given. Protocol π^0 then implements functionality F^{ρ} UC securely against static adversaries corrupting any number t of parties. That is, π^0 implements the ideal functionality* F

1. *with IT full security, given that $t = 0$,*
2. *with IT fair security (as formalized by $\mathsf{F}^{\mathsf{fair}}$), given that $t < \frac{n}{2}$, and*
3. *with CO abort security (as formalized by F^{ab}), always.*

8 Conclusions

We describe a hyrbid secure MPC protocol π^{ρ} that provides a flexible and optimal trade-off between IT full security (with robustness), IT fair security (no robustness), and CO abort security (no fairness). More precisely, for an arbitrarily chosen robustness parameter $\rho < \frac{n}{2}$, the hybrid-secure MPC protocol π^{ρ} is IT full secure for $t \leq \rho$, IT fair secure for $t < \frac{n}{2}$, and CO abort secure for $t < n - \rho$ actively and statically corrupted parties. These results are optimal with respect to the bounds stated in [Cle86,Kat07,IKLP06]. On the technical side, we provide a first formal treatment of player emulation in the UC setting.

We prove the UC security of π^{ρ} in the synchronous secure channels model with broadcast (BC) and a crs. We also show a simple variation π^{ρ}_{SA} of protocol π^{ρ} that relies on [GMW87] instead of [CLOS02] and is stand-alone secure in the synchronous secure channels model with BC *without* a crs.

Furthermore we discuss the synchronous secure channels model *without* BC. Here we find that for robustness parameter $\rho > 0$ the results of [FHHW03] are already optimal, but for $\rho = 0$ our protocol achieves the same results as in the case where BC is provided, indicating that a BC channel is only helpful if one aims for robustness.

References

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC'88*, pages 1–10. ACM, 1988.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. A general composition theorem for secure reactive systems. In *TCC'04*, volume 2951 of *LNCS*, pages 336–354. Springer, 2004.

[Can01] R. Canetti. Universally composable security: a new paradigm for cryptographic protocols. *FOCS'01*, pages 136–145, 2001.

[CCD88] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols. In *STOC'88*, pages 11–19. ACM, 1988.

[CDD+99] Ronald Cramer, Ivan Damgård, Stefan Dziembowski, Martin Hirt, and Tal Rabin. Efficient multiparty computations secure against an adaptive adversary. In *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer-Verlag, 1999.

[CDG88] David Chaum, Ivan Damgård, and Jeroen van de Graaf. Multiparty computations ensuring privacy of each party's input and correctness of the result. In *CRYPTO'87*, pages 87–119. Springer, 1988.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO'01*, pages 19–40. Springer, 2001.

[Cha89] David Chaum. The spymasters double-agent problem: Multiparty computations secure unconditionally from minorities and cryptographically from majorities. In *CRYPTO'89*, pages 591–602. Springer, 1989.

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *STOC'86*, pages 364–369, New York, NY, USA, 1986. ACM Press.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC'02*, pages 494–503. ACM, 2002.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO'02*, volume 2442 of *LNCS*, pages 581–596. Springer, 2002.

[FHHW03] Matthias Fitzi, Martin Hirt, Thomas Holenstein, and Jürg Wullschleger. Two-threshold broadcast and detectable multi-party computation. In *EUROCRYPT'03*, volume 265 of *LNCS*, pages 51–67. Springer, 2003.

[FHW04] Matthias Fitzi, Thomas Holenstein, and Jürg Wullschleger. Multi-party computation with hybrid security. In *EUROCRYPT'04*, volume 3027 of *LNCS*, pages 419–438. Springer, 2004.

[GK08] Vipul Goyal and Jonathan Katz. Universally composable multi-party computation with an unreliable common reference string. In *TCC'08*, volume 4948 of *LNCS*, pages 142–154. Springer, 2008.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *STOC'87*, pages 218–229. ACM, 1987.

[GO07] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *CRYPTO'07*, volume 4622 of *LNCS*, pages 323–341. Springer, 2007.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, 2001.

[Gol04] Oded Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, 2004.

[Her05] Amir Herzberg. On tolerant cryptographic constructions. In *CT-RSA'05*, volume 3376 of *LNCS*, pages 172–190. Springer, 2005.

[HM00] Martin Hirt and Ueli Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, 2000. Extended abstract in *Proc. 16th of ACM PODC '97*.

[IKLP06] Yuval Ishai, Eyal Kushilevitz, Yehuda Lindell, and Erez Petrank. On combining privacy with guaranteed output delivery in secure multiparty computation. In *CRYPTO'06*, volume 4117/2006, pages 483–500. Springer, 2006.

[Kat07] Jonathan Katz. On achieving the "best of both worlds" in secure multiparty computation. In *STOC'07*, pages 11–20. ACM, 2007.

[Kil00] Joe Kilian. More general completeness theorems for secure two-party computation. In *STOC'00*, pages 316–324. ACM, 2000.

[KMQR09] Robin Künzler, Jörn Müller-Quade, and Dominik Raub. Secure computability of functions in the IT setting with dishonest majority and applications to long-term security. In *TCC'09*, volume 5444 of *LNCS*, pages 238–255. Springer, 2009.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *ACM CCS'00*, pages 245–254, 2000.

[RB89] T. Rabin and M. Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority. In *STOC'89*, pages 73–85. ACM, 1989.

[Wul07] Jürg Wullschleger. *Oblivious-Transfer Amplification*. PhD thesis, ETH Zürich, 2007.

[Yao82] Andrew C. Yao. Protocols for secure computations (extended abstract). In *FOCS'82*, pages 160–164. IEEE, 1982.

A Proof of Lem. 2

In order to prove Lem. 2, we need to provide a simulator S^{em} such that the ideal model $\mathsf{S}^{\mathsf{em}}_{\mathcal{A}} \circ \mathsf{P}^{\mathsf{em}}_0$ becomes IT indistiguishable from the real model $\pi^{\mathsf{em}}_{\mathcal{H}} \circ \mathsf{net}^n \circ \mathsf{P}^{\mathsf{pm}}_0$. The simulator S^{em} connects to all interfaces of $\mathsf{P}^{\mathsf{em}}_0$ associated with corrupted parties. The interfaces exposed by $\mathsf{P}^{\mathsf{em}}_0$ are those to the network net^{n+1} it runs, and, for $t < \frac{n}{2}$, the n I/O-interfaces of the protocol π_0 of P_0. The cases $t \geq \frac{n}{2}$ and $t < \frac{n}{2}$ differ: In case $t \geq \frac{n}{2}$ the simulator S^{em} has to handle the interfaces of corrupted parties among the $\mathsf{P}_1, \dots, \mathsf{P}_n$ to net^{n+1} and the P_0 interface to net^{n+1}. In case $t < \frac{n}{2}$ the simulator S^{em} has to handle the interfaces of corrupted parties to protocol π_0 and to net^{n+1} (but not the P_0 interface to net^{n+1}). We can treat both cases jointly if we consider the emulated party P_0 honest for $t < \frac{n}{2}$ and corrupted $t \geq \frac{n}{2}$ as suggested above.

The simulator S^{em} then internally simulates an instance $\widetilde{\mathsf{net}^n}$ of net^n and copies $\widetilde{\pi^{\mathsf{em}}_i}$ of π^{em}_i for the honest parties (including P_0 for $t < \frac{n}{2}$). These machines are connected as in protocol π^{em}. Here, for $t < \frac{n}{2}$ the machine $\widetilde{\pi^{\mathsf{em}}_0}$

connects to the other $\widetilde{\pi_i^{\mathsf{em}}}$ like $\mathsf{P}_0^{\mathsf{pm}}$ in the real model and also connects to the I/O-interfaces of corrupted parties to protocol π_0. The simulator S^{em} internally makes use of the communication and the I/O-interface of the $\widetilde{\pi_i^{\mathsf{em}}}$ intended for P_i. The remaining interfaces are exposed to the distinguisher, i.e. the interfaces of corrupted parties to $\widetilde{\mathsf{net}^n}$, and for $t < \frac{n}{2}$ the interfaces of $\widetilde{\pi_0^{\mathsf{em}}}$ for connecting to the $\widetilde{\pi_i^{\mathsf{em}}}$ of corrupted parties or for $t \geq \frac{n}{2}$ the interfaces of $\widetilde{\pi_i^{\mathsf{em}}}$ for connecting to $\widetilde{\pi_0^{\mathsf{em}}}$.

Now when a (private or BC) message m is received from an honest party P_i via net^{n+1} the simulator S^{em} inputs m to the communication interface of $\widetilde{\pi_i^{\mathsf{em}}}$ for sending to the appropriate destination. On the other hand, when a $\widetilde{\pi_i^{\mathsf{em}}}$ outputs a (private or BC) message m from a corrupted P_j on the communication interface, this means P_j has sent the message m and simulator S^{em} forwards m to net^{n+1} via the interface of P_j. It is fairly straightforward to see that real system and simulation are perfectly indistinguishable.

B Primitives Protecting Party P_0 IT

Below we discuss suitable CO primitives for [CLOS02,GMW87], namely OT, commitments, (perfectly) zero-knowledge arguments of knowledge (ZK-AoK), and CO zero-knowledge proofs of knowledge (cZK-PoK) which IT protect the designated party P_0.

Oblivious Transfer. As shown in [CLOS02, Sec. 4.1.1] the OT protocol of [GMW87,Gol04, pp. 640–643] is UC secure. Furthermore, it is easy to see that it IT protects the receiver. The [CLOS02,GMW87] protocols make no restriction as to which participant of an OT execution acts as sender or receiver. So we may use said OT protocol and still IT protect P_0 by making P_0 the receiver in every invocation of OT involving P_0. Alternatively, a UC secure OT protocol that IT protects the sender can be obtained by "turning around" the above OT as shown in [Wul07, Thm. 4.1]. Thus, security for P_0 is guaranteed in any OT invocation, even with an IT adversary.

Commitment. For [CLOS02], we use the UC secure, one-to-many IT hiding and IT binding commitment schemes in the crs-model described in Sec. E. For our purpose, we employ the IT binding variation for commitments issued by the parties P_i ($i \in [n]$) and the IT hiding variation for commitments issued by the designated party P_0. Thus we obtain a UC secure realization of the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com,1:M}}$ that guarantees security for P_0 against any IT adversary.

In the stand-alone case, for [GMW87], we may directly use IT hiding and IT binding commitment schemes which do not rely on a crs.

ZK proofs. [CLOS02, Prop. 9.4] shows how to UC securely implement the ZK functionality $\mathsf{F}_{\mathsf{ZK,1:M}}$ from the commitment functionality $\mathsf{F}_{\mathsf{Com,1:M}}$ without use of further CO assumptions.[14] The one-to-many ZK protocol of [CLOS02] is based on the two party protocol of [CF01, Sec. 5]. This protocol in turn is based on the Hamiltonian Cycles ZK proof. Hence, on the one hand, when using an IT binding commitment scheme, we obtain cZK-PoKs. On the other hand, when using an IT hiding commitment scheme, we obtain ZK-AoKs. The one-to-many property is obtained by repeating the two-party protocol with each player over the broadcast channel. In addition, the proof is accepted only if the transcripts of all invocations constitute valid proofs. Now, if P_0 is the prover, we instantiate the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com,1:M}}$ with the IT hiding scheme described above. Otherwise, we instantiate $\mathsf{F}_{\mathsf{Com,1:M}}$ with the IT binding scheme. Thus we obtain a protocol that is always secure for P_0, even against any IT adversary.

In the stand-alone case, for [GMW87], we may directly use cZK-PoKs and ZK-AoKs which do not rely on a crs.

Commit-and-Prove. Instead of directly working with commitment and ZK functionalities (as [GMW87] does), [CLOS02] introduces a new primitive called one-to-many commit-and-prove $\mathsf{F}_{\mathsf{CP,1:M}}$. [CLOS02, Sec. 7.1] provides a protocol implementing the two-party[15] functionality F_{CP} secure against static adversaries, which relies

[14] Actually, this protocol is secure against adaptive adversaries. For static adversaries a non-interactive protocol might be used. However, for ease of discussion, we directly use the adaptive protocol.

[15] The one-to-many protocol presented in [CLOS02, Prop. 9.5] encompasses adaptive adversaries.

only on F_{ZK} and a standard commitment scheme (together with the corresponding computational assumptions). Since this protocol is non-interactive, it can easily be extended into a one-to-many protocol by having the sender broadcast all messages and use $\mathsf{F}_{\mathsf{ZK,1:M}}$ instead of F_{ZK}. Hence, when implementing $\mathsf{F}_{\mathsf{ZK,1:M}}$ as described above, we can use IT hiding or IT binding commitments in the implementation of $\mathsf{F}_{\mathsf{CP,1:M}}$ to IT protect the designated party P_0. Further CO assumptions are not required. Thus, we can implement the commit-and-prove functionality $\mathsf{F}_{\mathsf{CP,1:M}}$ UC securely for P_0, even against any IT adversary.

C Proof Sketch for Lem. 4

We will now show that the modified version of [CLOS02] described in Sec. 5.2 fulfills the requirements stated in Lem. 4.

Case 1: CO security with robustness for $t \leq \rho$ and fairness beyond, P_0 honest ($t < \frac{n}{2}$). This claim follows immediately from the IT security guarantees of protocol $\pi^{\mathsf{des},\rho}$ shown in Case 3, and the CO security guarantees shown in Case 2.

Case 2: CO security with abort for $t < n - \rho$ corrupted parties, P_0 corrupted ($t \geq \frac{n}{2}$). CO correctness and privacy are already implied by [CLOS02,GMW87]. Our modifications to CO primitives and opening procedures are within the limits of the original protocol and only apply restrictions as to what kinds of primitives are used in specific situations. The modification to the sharing can be treated as in [Gol04]. As shares observed by corrupted parties are still uniformly random for $t < n - \rho$ corrupted parties the modifications to the simulator remain trivial. As such the proof in [CLOS02,GMW87] remains applicable with minimal modifications and we obtain a CO secure implementation of the ideal functionality in this case.

Note that agreement on abort is achieved: 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 a BC channel, this will be noted by all honest parties and they will all abort.

Case 3: IT security for honest P_0 ($t < \frac{n}{2}$) with robustness for $t \leq \rho$, and with fairness for $t < n - \rho$, P_0. We sketch a simulator to demonstrate that [CLOS02], tweaked as described above, UC securely implements the ideal functionality $\mathsf{J}^{\mathsf{des}}$ in the given setting, i.e. an IT adversary corrupting $t \leq n - \rho$ parties not including party P_0. We have to show that correctness, privacy for P_0, and fairness are guaranteed. In case of $t \leq \rho$ we have to show that robustness is maintained as well. The proof for [GMW87] in the stand-alone setting works analogously, but relies on rewinding to facilitate simulation, instead of extractability and equivocability of UC commitments by means of the crs. Hence we refrain from describing a separate simulator for the stand-alone setting.

The simulator will receive the inputs of all honest parties except P_0 from the ideal functionality $\mathsf{J}^{\mathsf{des}}$. Furthermore corrupted parties have to commit to their input using binding UC commitments[16], so by extractability the simulator can extract their inputs and forward them to $\mathsf{J}^{\mathsf{des}}$. The simulator then simulates the protocol machines of all honest parties, with an *arbitrary* input x'_0 for the simulated party P_0, and the inputs of the other honest parties as obtained from $\mathsf{J}^{\mathsf{des}}$. The simulation then proceeds up to the point where an output y is opened, i.e. where the simulator receives a y from $\mathsf{J}^{\mathsf{des}}$. Recall that party P_0, which is honest by assumption and thus simulated internally by the simulator, is supposed to broadcast its opening information last, and only if sufficiently many (i.e. $n - \rho$) parties have broadcasted their opening information correctly, in order to guarantee correct reconstruction of y.

We first consider the case where $\rho < t \leq n - \rho$ parties are corrupted. Thus, we only need to guarantee fairness. If at least $t - \rho$ corrupted parties broadcast their opening information correctly, then the simulator makes use of the equivocability of commitments to have the internally simulated P_0 open to the output y and sets the output flag to $o = 1$, otherwise it sets $o = 0$.

Finally, given that $t \leq \rho$ parties are corrupted, the adversary can no longer prevent the honest parties from reconstructing the output. Hence, P_0, regardless of the shares distributed by the adversary, opens the output to y, again making use of the equivocability of commitments.

[16] Providing extractability and equivocability by means of the crs.

The behavior of the simulator is IT indistinguishable from the real protocol. As in the real protocol, the designated party P_0 in the simulation only converses with the other parties by means of hiding commitments, ZK proofs and OT invocations IT protecting party P_0. Furthermore, the sharing scheme is such that without cooperation of the designated party P_0, which is honest by assumption, no information can be recovered. As such no information whatsoever is disseminated by the designated party P_0 to the corrupted parties until reconstruction takes place in an opening phase.

Case 4: IT, P_0 *corrupted (*$t \geq \frac{n}{2}$*), no security guarantees.* As we make no security guarantees in this case, there is nothing to show.

D Proof of Lem. 5

We have to show that the protocol π^{in} implements F^{ρ} from functionality $\mathsf{J}^{\mathsf{des}}$ for the choice of J described in Fig. 9. We do so by providing an appropriate simulator S^{in} that renders the ideal model $\mathsf{S}^{\mathsf{in}} \circ \mathsf{F}^{\rho}$ indistinguishable from the real model $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$. We treat the settings $t < \frac{n}{2}$ and $\frac{n}{2} \leq t < n - \rho$ separately. For $t \geq n - \rho$ functionality F^{ρ} gives up, so there is nothing to show. For $\frac{n}{2} \leq t < n - \rho$, we have to show that protocol π^{in} implements F^{ρ} in the CO setting. For $t < \frac{n}{2}$, it suffices to show that protocol π^{in} implements F^{ρ} in the IT setting.

D.1 Proof of Lem. 5 for $\frac{n}{2} \leq t < n - \rho$

We show that, for $\frac{n}{2} \leq t < n - \rho$, there is a simulator S^{in} which renders ideal model $\mathsf{S}^{\mathsf{in}} \circ \mathsf{F}^{\rho}$ indistinguishable from the real model $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$ in the CO setting.

In the CO setting, for $\frac{n}{2} \leq t < n - \rho$, functionality $\mathsf{J}^{\mathsf{des}}$ is correct and private for inputs at its I/O-interfaces, but gives the adversary control over inputs at its P_0-interfaces (we may consider the emulated party P_0 corrupted) and guarantees no robustness or fairness, only agreement on abort.

The simulator S^{in} is connected to the interfaces of the corrupted parties to the ideal functionality F^{ρ}. In turn the simulator S^{in} simulates the I/O-interfaces of functionality $\mathsf{J}^{\mathsf{des}}$ belonging to corrupted parties and the P_0-interface of functionality $\mathsf{J}^{\mathsf{des}}$ to the distinguisher.

For $\frac{n}{2} \leq t < n - \rho$, the simulator S^{in} then operates as follows:

1. When an honest party makes input to F^{ρ} or the distinguisher makes input via the I/O-interface of a corrupted party:
   (a) For all honest parties P_i making input, choose $\tilde{x}^{\mathsf{it}}_i$ at random and compute IT hiding commitments $(c_i, \tilde{o}_i) = \mathsf{commit}(\tilde{x}^{\mathsf{it}}_i)$.
   (b) Give the $(\tilde{x}^{\mathsf{it}}_i, \tilde{o}_i)$ as output to the distinguisher over the P_0-interface.
   (c) Receive some (x^{em}_i, o_i) from the distinguisher over the P_0-interface.
   (d) Receive some $(x^{\mathsf{des}}_i, c_i)$ from the distinguisher over the I/O-interfaces of the $\mathsf{P}_i \in \mathcal{A}$.
   (e) Output a complaint vector $\boldsymbol{e} = (x^{\mathsf{em}}_i \stackrel{?}{=} \mathsf{open}(c_i, o_i))_{i \in [n]}$ to the distinguisher via the I/O-interfaces.
   (f) Receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to F^{ρ} and halt.
   (g) For the $\mathsf{P}_i \in \mathcal{A}$ where $e_i = 1$ compute $x_i := x^{\mathsf{des}}_i \oplus x^{\mathsf{em}}_i$.
   (h) Take new inputs x_i on the I/O-interfaces of the $\mathsf{P}_i \in \mathcal{A}$ where $e_i = 0$, default to $x_i = \perp$ if no input is provided.
   (i) Forward all inputs $x_i \neq \perp$ $(i \in \mathcal{A})$ to functionality F.
2. When functionality F^{ρ} makes output:
   (a) Forward the output y of F^{ρ} to the distinguisher via the I/O-interfaces of the $\mathsf{P}_i \in \mathcal{A}$
   (b) Receive an output flag o from the distinguisher, default to $o = 1$ if no output flag is provided.
   (c) Forward the output flag o to F^{ρ}, and in case $o = 0$ halt.

We now argue that the simulator S^{in} indeed renders the ideal model $\mathsf{S}^{\mathsf{in}} \circ \mathsf{F}^{\rho}$ indistinguishable from the real model $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$.

When input is made by some party P_i, protocol machine π^{in}_i in $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$ first splits its input into $x_i = x_i^{\mathsf{des}} \oplus x_i^{\mathsf{em}}$ (where x_i^{em} is uniformly random) and computes the IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{em}})$. Then, π^{in}_i provides (x_i^{em}, o_i) as input to functionality $\mathsf{J}^{\mathsf{des}}$ at the P_0-interface which is controlled by the adversary. S^{in} simulates this indistinguishably by providing random values $(\tilde{x}_i^{\mathsf{it}}, \tilde{o}_i)$ with appropriate opening information to the distinguisher over the P_0-interface.

Furthermore, protocol machine π^{in}_i provides $(x_i^{\mathsf{des}}, c_i)$ as input to $\mathsf{J}^{\mathsf{des}}$ via the I/O-interface. Functionality $\mathsf{J}^{\mathsf{des}}$ then issues a boolean complaint vector $\boldsymbol{e} = (x_i^{\mathsf{em}} = \mathsf{open}(c_i, o_i))_{i \in [n]}$, indicating for which parties the opening failed. The complaint vector $\boldsymbol{e}$ is first handed to the adversary and upon receipt of an output flag $o = 1$ to the remaining parties. Functionality $\mathsf{J}^{\mathsf{des}}$ then allows these parties to answer the complaint with a new x_i, and computes $x_i = x_i^{\mathsf{des}} \oplus x_i^{\mathsf{em}}$ for the remaining parties. S^{in} simulates this behavior identically to the corrupted parties. Finally the ideal functionality $\mathsf{J}^{\mathsf{des}}$ forwards the x_i to F, which simulator S^{in} simulates by inputting the x_i to F^{ρ}.

This simulation is faithful as long as the adversary does not manage to open a commitment c_i to a value other than x_i^{em} (which being CO bounded it cannot).[17]

When output is made, functionality $\mathsf{J}^{\mathsf{des}}$ delivers the output y to the adversary and awaits an output flag deciding output delivery to honest parties. Outputs are simply forwarded by π^{in}_i. Functionality F^{ρ} behaves identically and as such the simulator S^{in} need only forward the messages in question.

Hence the protocol π^{in} CO securely implements the functionality F^{ρ} for $\frac{n}{2} \leq t \leq n - \rho$.

D.2 Proof of Lem. 5 for $t < \frac{n}{2}$

We show that, for $t < \frac{n}{2}$, there is a simulator S^{in} which renders ideal model $\mathsf{S}^{\mathsf{in}} \circ \mathsf{F}^{\rho}$ indistinguishable from the real model $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$ in the CO setting.

In the IT setting for $t < \frac{n}{2}$, functionality $\mathsf{J}^{\mathsf{des}}$ is fair, correct and private for inputs at its P_0-interfaces (we may consider the emulated party P_0 honest) but forwards inputs at its I/O-interfaces to the adversary. For $t \leq \rho$, functionality $\mathsf{J}^{\mathsf{des}}$ is additionally robust.

The simulator S^{in} is connected to the interfaces of the corrupted parties to the ideal functionality F^{ρ}. In turn the simulator S^{in} simulates the I/O- and P_0-interfaces of functionality $\mathsf{J}^{\mathsf{des}}$ belonging to corrupted parties to the distinguisher.

For $t < \frac{n}{2}$, the simulator S^{in} then operates as follows:

1. When an honest party makes input to F^{ρ} or the distinguisher makes input via the I/O-interface of a corrupted party:
   (a) For all honest parties P_i making input, choose x_i^{des} and x_i^{em} at random and compute IT hiding commitments $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{em}})$.
   (b) Give the $(x_i^{\mathsf{des}}, c_i)$ as output to the distinguisher.
   (c) Receive inputs (x_i^{em}, o_i) and $(x_i^{\mathsf{des}}, c_i)$ from the distinguisher over the P_0- and I/O-interfaces of the corrupted parties $\mathsf{P}_i \in \mathcal{A}$ respectively.
   (d) For $\rho < t < \frac{n}{2}$, request an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to F^{ρ} and halt.
   (e) Output a complaint vector $\boldsymbol{e} = (x_i^{\mathsf{em}} \stackrel{?}{=} \mathsf{open}(c_i, o_i))_{i \in [n]}$ to the distinguisher via the I/O-interfaces.
   (f) For the $\mathsf{P}_i \in \mathcal{A}$ where $e_i = 1$ compute $x_i := x_i^{\mathsf{des}} \oplus x_i^{\mathsf{em}}$.
   (g) Take new inputs x_i on the I/O-interfaces of the $\mathsf{P}_i \in \mathcal{A}$ where $e_i = 0$, default to $x_i = \perp$ if no input is provided.
   (h) Forward all inputs $x_i \neq \perp$ ($i \in \mathcal{A}$) to functionality F.
2. When functionality F^{ρ} makes output

[17] The commitments to the x_i^{em} and the complaint procedure guarantee that the computation is carried out with correct values x_i^{em}. That is, the input shares x_i^{des} and x_i^{em} have the relation $x_i^{\mathsf{des}} \oplus x_i^{\mathsf{em}} = x_i$. Otherwise, if the adversary controls the P_0-interfaces (as is the case here), he could manipulate the values x_i^{em} leading to a computation with wrong inputs x_i and hence to an incorrect result.

(a) For $\rho < t < \frac{n}{2}$, request an output flag o from the distinguisher, default to $o = 1$ if none is provided. Forward o to F^ρ and, in case $o = 0$, halt.
(b) Forward the output y of F^ρ to the distinguisher via the I/O-interfaces of the $\mathsf{P}_i \in \mathcal{A}$.

We now argue that the simulator S^{in} indeed renders the ideal model $\mathsf{S}^{\mathsf{in}} \circ \mathsf{F}^\rho$ indistinguishable from the real model $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$.

When input is made by some party P_i, protocol machine π^{in}_i in $\pi^{\mathsf{in}}_{\mathcal{H}} \circ \mathsf{J}^{\mathsf{des}}$ first splits its input into $x_i = x^{\mathsf{des}}_i \oplus x^{\mathsf{em}}_i$ (where x^{em}_i is uniformly random) and computes the IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x^{\mathsf{em}}_i)$. Then, π^{in}_i provides (x^{em}_i, o_i) and $(x^{\mathsf{des}}_i, c_i)$ as input to $\mathsf{J}^{\mathsf{des}}$ via the P_0- and the I/O-interface of P_i to $\mathsf{J}^{\mathsf{des}}$ respectively. In the current context, for $t < \frac{n}{2}$ in the IT setting, $\mathsf{J}^{\mathsf{des}}$ forwards $(x^{\mathsf{des}}_i, c_i)$ to the adversary. S^{in} simulates this indistinguishably by providing random values $(x^{\mathsf{des}}_i, c_i)$ to the distinguisher. Here it is important to note that c_i is a hiding commitment, and as such really independent of x^{em}_i.

Functionality $\mathsf{J}^{\mathsf{des}}$ requests an output flag o and for $o = 1$ issues a boolean complaint vector $\boldsymbol{e} = (x^{\mathsf{em}}_i = \mathsf{open}(c_i, o_i))_{i \in [n]}$, indicating for which parties the opening failed. Functionality $\mathsf{J}^{\mathsf{des}}$ then allows these parties to answer the complaint with a new x_i, and computes $x_i = x^{\mathsf{des}}_i \oplus x^{\mathsf{em}}_i$ for the remaining parties. Note that for $t < \frac{n}{2}$ no honest party will ever receive a complaint when trying to give input. S^{in} simulates this behavior identically to the corrupted parties. Finally the ideal functionality $\mathsf{J}^{\mathsf{des}}$ forwards the x_i to F, which simulator S^{in} simulates by inputting the x_i to F^ρ.

When output is made, functionality $\mathsf{J}^{\mathsf{des}}$ for $\rho < t < \frac{n}{2}$ requests an output flag o from the distinguisher, defaulting to $o = 1$ if none is provided. In case $o = 0$, functionality $\mathsf{J}^{\mathsf{des}}$ halts. Otherwise $\mathsf{J}^{\mathsf{des}}$ delivers the output y to all parties, Outputs are simply forwarded by π^{in}_i. Functionality F^ρ behaves identically, so the simulator S^{in} need only forward the messages in question.

Hence the protocol π^{in} IT securely implements the functionality F^ρ for $t < \frac{n}{2}$.

E Perfecly Hiding or Perfectly Binding UC Commitments

We describe a UC secure one-to-many commitment schemes implementing the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com},1:\mathsf{M}}$ that can be either perfectly hiding or perfectly binding. Our one-to-many commitment scheme is derived from the perfectly hiding or perfectly binding UC secure commitment schemes of [DN02].

Functionality $\mathsf{F}_{\mathsf{ComH},1:\mathsf{M}}$ formalizes perfectly hiding UC secure one-to-many commitment schemes, functionality $\mathsf{F}_{\mathsf{ComB},1:\mathsf{M}}$ formalizes perfectly binding UC secure one-to-many commitment schemes.

Functionality $\mathsf{F}_{\mathsf{ComH},1:\mathsf{M}}$ operates as follows:

1. If the committer C is honest or in the CO setting:
   (a) On receipt of a message m from the committer C, output committed to all receivers R_i.
   (b) On receipt of open from the committer C, output m to all receivers R_i.
2. If the committer C is corrupted in the IT setting, turn control over to the adversary.

Functionality $\mathsf{F}_{\mathsf{ComB},1:\mathsf{M}}$ operates as follows:

1. For honest receivers R_i or in the CO setting for all receivers R_i:
   (a) On receipt of a message m from the committer C, output committed to the receiver R_i.
   (b) On receipt of open from the committer C, output m to the receiver R_i.
2. For corrupted receivers R_i in the IT setting,
   (a) On receipt of a message m from the committer C, directly output m to the receiver R_i.
   (b) On receipt of open from the committer C, output open to the receiver R_i.

We no show how to extend the commitment scheme of [DN02] to implement the functionalities $\mathsf{F}_{\mathsf{ComH},1:\mathsf{M}}$ and $\mathsf{F}_{\mathsf{ComB},1:\mathsf{M}}$.

E.1 Mixed Commitments

The construction of our UC commitment scheme is based on the mixed commitment scheme commit_K described in [DN02]. A mixed commitment scheme is parametrized by a system key N with an associated X-trapdoor t_N which determines keyspace $\mathcal{K}_N$ and message space $\mathcal{M}_N$. Both $\mathcal{K}_N$ and $\mathcal{M}_N$ are additive groups. The keyspace $\mathcal{K}_N$ is partitioned into subsets $\mathcal{K}_X$ of X-keys (for extractability), $\mathcal{K}_E$ of E-keys (for equivocability), and $\mathcal{K}_R$ of remaining keys. An overwhelming fraction of keys in $\mathcal{K}_N$ are X-keys in $\mathcal{K}_X$. One can efficiently generate random system keys N, random keys in $\mathcal{K}_N$, random X-keys in $\mathcal{K}_X$, and random E-keys in $\mathcal{K}_E$. All X-keys $K \in \mathcal{K}_X$ have a common trapdoor t_N that can efficiently be generated together with the system key N. In contrast, all E-keys $K \in \mathcal{K}_E$ have their own trapdoor t_K that can efficiently be generated together with the key itself. Furthermore, random keys, X-keys, and E-keys are CO indistinguishable.

The commitment scheme commit_K with key $K \in \mathcal{K}_N$ is equivocable for $K \in \mathcal{K}_E$ and extractable for $K \in \mathcal{K}_X$. So, on the one hand, for a commitment $c = \mathsf{commit}_K(m, r)$ where $K \in \mathcal{K}_X$ one can efficiently determine m from c, K, N, and the trapdoor t_N (extractability). On the other hand, given $K \in \mathcal{K}_E$ and the associated E-trapdoor t_K one can efficiently generate a commitment c that is equivocable, i.e. it is efficiently possible to generate randomness r such that $c = \mathsf{commit}_K(m, r)$ for any $m \in \mathcal{M}_N$. Note that extractability and equivocability together with the CO indistinguishability of random keys, X-keys, and E-keys imply that the mixed commitment scheme commit_K is CO binding and hiding. More details on mixed commitments can be found in [DN02].

E.2 The CRS

UC commitments require a stronger setup than a broadcast channel [CF01]. We will use a common random string (CRS) that is sampled from a prescribed distribution by a trusted functionality.

Our CRS will be $\mathsf{crs} = (N, K_X, K_E, \bar{K}_1, \dots, \bar{K}_n, \mathsf{crs}')$. The first part of the crs, encompassing the $n + 3$ keys $N, K_X, K_E, \bar{K}_1, \dots, \bar{K}_n$, stems from the original protocol in [DN02]. In accordance to this protocol, N is a random system key for our mixed commitment, $K_X \in \mathcal{K}_X$ is a random X-key and $K_E, \bar{K}_1, \dots, \bar{K}_n \in \mathcal{K}_E$ are random E-keys. The second part of the crs, i.e. crs', is a CRS for one-to-many commitments according to [CLOS02,CF01]. This part is only needed for the one-to-many extension of the commitment scheme discussed here.

E.3 The UC Commitment Protocol

Wlog let $C = P_1$ be the committer and the remaining parties be the receivers $R_i = P_i$ ($i \in 2, \dots, n$). Furthermore, let $(\mathsf{commit}', \mathsf{open}')$ denote the one-to-many commitment scheme according to [CLOS02,CF01]. The UC one-to-many commitment protocol then works as follows:

Commit phase:

C.1 On input m, committer C draws a random $K_1 \in \mathcal{K}_N$ and random opening information r_1, and broadcasts $c_1 = \mathsf{commit}_{\bar{K}_1}(K_1, r_1)$.
R.1 The receivers R_i run a coin toss protocol in order to sample a random key K_2:
- **R.1.a** Each receiver R_i draws a random $s_i \in \mathcal{K}_N$, computes $(c'_i, o'_i) = \mathsf{commit}'(s_i, \mathsf{crs}')$, and broadcasts c'_i.
- **R.1.b** Each receiver R_i broadcasts (s_i, o_i).
- **R.1.c** All parties compute $K_2 = \sum_i s_i$ for the s_i where $s_i = \mathsf{open}'(c'_i, o'_i)$.

C.2 Committer C computes $K = K_1 + K_2$, draws random opening information r_2, r_3, and
- for an IT hiding commitment draws $\bar{m}$ and broadcasts $c_2 = \mathsf{commit}_K(\bar{m}+m, r_2)$, $c_3 = \mathsf{commit}_{K_E}(\bar{m}, r_3)$
- for an IT binding commitment broadcasts $c_2 = \mathsf{commit}_K(m, r_2)$, $c_3 = \mathsf{commit}_{K_X}(m, r_3)$

R.2 Each receiver R_i upon receiving c_2 and c_3 outputs $\mathsf{commited}$

Opening phase:

C.1 On input open, committer C broadcasts
- for an IT hiding commitment $(m, \bar{m}, r_2, r_3)$

- for an IT binding commitment (m, r_2, r_3)

R.1 Each receiver R_i verifies that
- for an IT hiding commitment $c_2 = \mathsf{commit}_K(\bar{m} + m, r_2)$, $c_3 = \mathsf{commit}_{K_E}(\bar{m}, r_3)$,
- for an IT binding commitment $c_2 = \mathsf{commit}_K(m, r_2)$, $c_3 = \mathsf{commit}_{K_X}(m, r_3)$

and if so, outputs m.

Note that this protocol is a simple adaption of [DN02] to multiple receivers. We simply replace round R.1 of [DN02] where the single receiver of [DN02] chooses a random K_2 with a CO secure cointoss among our multiple receivers.

E.4 Security of the UC Commitment Protocol

We prove security by providing simulators for the IT hiding and the IT binding case separately. The argument why these simulators achieve indistinguishability does not change substantially and we refer the reader to [DN02].

IT Hiding We now show that the perfectly hiding variation of the scheme above indeed implements functionality $\mathsf{F}_{\mathsf{ComH,1:M}}$. We consider three cases for which we provide different simulators, namely:

1. the adversary is CO or IT, leaves the committer C honest (and corrupts any number of receivers R_i).
2. the adversary is CO, corrupts the committer C (and any number of receivers R_i),
3. the adversary is IT, corrupts the committer C (and any number of receivers R_i).

In the first two cases the commitment functionality $\mathsf{F}_{\mathsf{ComH,1:M}}$ operates as expected and described in [CF01]. Simulator $\mathsf{S}^{\mathsf{it}}_R$ is used in case 1 where C is honest, but any number of receivers R_i are IT or CO corrupted (the simulator works in both cases). First, $\mathsf{S}^{\mathsf{it}}_R$ produces a regular crs with E-key K_E and E-trapdoor t_E. During the commit phase, $\mathsf{S}^{\mathsf{it}}_R$ emulates C on random input to the corrupted R_i. Indistinguishability is preserved because all commitments are equivocable and thus independent of their "content". In the opening phase, $\mathsf{S}^{\mathsf{it}}_R$ receives the correct $m^\star$ from $\mathsf{F}^h_{\mathsf{com}}$. Now, $\mathsf{S}^{\mathsf{it}}_R$ opens the K_E commitment c_3 to $m_3' = m^\star \oplus m_2$ using t_E.

Finally, simulator $\mathsf{S}^{\mathsf{co}}_C$ is used in case 2 where C and any number of receivers are CO corrupted. First, $\mathsf{S}^{\mathsf{co}}_C$ produces a fake $\widetilde{\mathsf{crs}}$ with interchanged keys: On one hand, in $\widetilde{\mathsf{crs}}$, K_X is an *equivocable* key taken from $\mathcal{K}_E$, together with trapdoor t_{K_X} for equivocability. On the other hand, in $\widetilde{\mathsf{crs}}$, K_E is an *extractable* key taken from $\mathcal{K}_X$. Note that K_E has trapdoor t_N for extractability. For a CO adversary, the fake $\widetilde{\mathsf{crs}}$ is indistinguishable from a real crs. Furthermore, $\mathsf{S}^{\mathsf{co}}_C$ internally runs the protocol of honest R_i which can be perfectly simulated since they do not require any input. During the simulation, $\mathsf{S}^{\mathsf{co}}_C$ simply forwards all messages among the (internally simulated) honest R_i and the corrupted parties, i.e. C and corrupted R_i. After the commit phase, $\mathsf{S}^{\mathsf{co}}_C$ uses the known system trapdoor N to extract m_3 from c_3 (X-key by choice of the CRS) and m_2 for c_2 (X-key with overwhelming probability in the regular protocol) and inputs $m^\star = m_3 \oplus m_2$ to $\mathsf{F}^h_{\mathsf{com}}$. In the opening phase, $\mathsf{S}^{\mathsf{co}}_C$ sends an open message to $\mathsf{F}^h_{\mathsf{com}}$ if and only if C provides correct opening information for $m^\star$.

In the last case, committer C and any number of receivers are IT corrupted. By definition of IT hiding commitments, the functionality $\mathsf{F}^h_{\mathsf{com}}$ collapses in this context and turns over control to the simulator $\mathsf{S}^{\mathsf{it}}_C$. Our simulator $\mathsf{S}^{\mathsf{it}}_C$ first produces a regular crs. Then, $\mathsf{S}^{\mathsf{it}}_C$ internally runs the protocol of the honest R_i to the I/O-interface of which it has access via the ideal functionality. During the simulation, $\mathsf{S}^{\mathsf{it}}_C$ simply forwards all messages among the (internally simulated) honest R_i and the corrupted parties, i.e. C and corrupted R_i.

As noted above, the indistinguishability arguments of [DN02] apply, and we refer the reader there for further detail.

IT Binding We now show that the perfectly binding variation of the scheme above indeed implements functionality $\mathsf{F}_{\mathsf{ComB,1:M}}$. Once again, we consider three cases for which we provide different simulators, namely:

1. the adversary is CO or IT, corrupts the committer C and any number of receivers R_i,
2. the adversary is CO, leaves the committer C honest and corrupts any number of receivers R_i,

3. the adversary is IT, leaves the committer C honest and corrupts any number of receivers R_i.

In the first two cases the commitment functionality operates as expected and described in [CF01]. Simulator $\mathsf{S}_C^{\mathsf{it}}$ is used in case 1 where C and any number of receivers are IT or CO corrupted. First, $\mathsf{S}_C^{\mathsf{it}}$ produces a regular crs with X-key K_X and X-trapdoor t_N. Furthermore, $\mathsf{S}_C^{\mathsf{it}}$ internally runs the protocol of honest R_i which can be perfectly simulated since they do not require any input. During the simulation, $\mathsf{S}_C^{\mathsf{it}}$ simply forwards all messages among the (internally simulated) honest R_i and the corrupted parties, i.e. C and corrupted R_i. After the commit phase, $\mathsf{S}_C^{\mathsf{it}}$ extracts the message m from c_3 using the trapdoor t_N, and enters it into $\mathsf{F}_{\mathsf{com}}^b$. In the opening phase, $\mathsf{S}_C^{\mathsf{it}}$ sends an open message to $\mathsf{F}_{\mathsf{com}}^b$ if and only if C provides correct opening information for m.

Simulator $\mathsf{S}_R^{\mathsf{co}}$ is used in case 2 where C is honest, but any number of receivers R_i is CO corrupted. First, $\mathsf{S}_R^{\mathsf{co}}$ produces a fake $\widetilde{\mathsf{crs}}$ with interchanged keys: On one hand, in $\widetilde{\mathsf{crs}}$, K_X is an *equivocable* key taken from $\mathcal{K}_E$, together with trapdoor t_{K_X} for equivocability. On the other hand, in $\widetilde{\mathsf{crs}}$, K_E is an *extractable* key taken from $\mathcal{K}_X$. Note that K_E has trapdoor t_N for extractability. For a CO adversary, the fake $\widetilde{\mathsf{crs}}$ is indistinguishable from a real crs. In the commit phase in step C.1, $\mathsf{S}_R^{\mathsf{co}}$ uses the trapdoor $t_{\bar{K}_C}$ of E-key $\bar{K}_C$ to produce an equivocable commitment c_1. Hence, C is not committed to the first part K_1 of the key K. Then, in step C.2, $\mathsf{S}_R^{\mathsf{co}}$ opens c_1 to a value K_1' such that $K = K_1' \oplus K_2$ is a random E-key with known trapdoor t_{K_E}. Usually, this would be an X-key with overwhelming probability. For the opening phase, $\mathsf{S}_R^{\mathsf{co}}$ receives the correct m from $\mathsf{F}_{\mathsf{com}}^h$. Then, $\mathsf{S}_R^{\mathsf{co}}$ opens the commitment c_3 and the commitment c_2 to $m_3' = m_2' = m$. By choice of the $\widetilde{\mathsf{crs}}$, the commitment c_3 was constructed with the equivocable E-key K_X and trapdoor t_{K_X}. By choice of K_1', the commitment c_2 was constructed with the equivocable E-key $K = K_1' \oplus K_2$ and trapdoor t_K. Hence, $\mathsf{S}_R^{\mathsf{co}}$ can efficiently open both commitments as needed.

In the last case, committer C is honest, but any number of receivers R_i is IT corrupted. By definition of IT binding commitments, the ideal functionality $\mathsf{F}_{\mathsf{com}}^b$ then directly leaks the committed message m to IT corrupted receivers. Honest R_i still receive m from $\mathsf{F}_{\mathsf{com}}^b$ on opening as usual. We use a simulator $\mathsf{S}_R^{\mathsf{it}}$ that exploits this. First, $\mathsf{S}_R^{\mathsf{it}}$ produces a regular crs. Then, it internally runs the protocol of C on input m, and the protocols of honest R_i, which do not need any input, towards the corrupted R_i.

As noted above, the indistinguishability arguments of [DN02] apply, and we refer the reader there for further detail.