Hybrid-Secure MPC: Trading Information-Theoretic Robustness for Computational Privacy*

Christoph Lucas†, Dominik Raub‡, and Ueli Maurer†

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

‡Department of Computer Science, Århus University, Denmark,
raub@cs.au.dk

Abstract. Most protocols for distributed, fault-tolerant computation, or multi-party computation (MPC), provide security guarantees in an *all-or-nothing* fashion: If the number of corrupted parties is below a certain threshold, these protocols provide all specified security guarantees. Beyond this threshold, they provide no security guarantees at all. Furthermore, most previous protocols achieve either information-theoretic (IT) security, in which case this threshold is low, or they achieve only computational security, in which case this threshold is high. In contrast, a hybrid-secure protocol provides different security guarantees depending on the set of corrupted parties and the computational power of the adversary, without being aware of the actual adversarial setting. Thus, hybrid-secure MPC protocols allow for graceful degradation of security.

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 one 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 (privacy and correctness only) for up to $t < n - \rho$. Our construction is secure in the universal composability (UC) framework (based on a network of secure channels, a broadcast channel, and a common reference string). It achieves the bound on the trade-off between robustness and privacy shown by Ishai et al. [CRYPTO'06] and Katz [STOC'07], the bound on fairness shown by Cleve [STOC'86], and the bound on IT security shown by Kilian [STOC'00], and is the first protocol that achieves all these bounds simultaneously.

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).

Keywords: Multi-party computation, information-theoretic security, computational security, hybrid security, universal composability, party emulation.

* An extended abstract of this paper appeared at PODC 10 [LRM10]. This work was partially supported by the Zurich Information Security Center, and by the Swiss National Science Foundation (SNF), SPP project no. 200020-113700/1.

1 Introduction

1.1 Secure Multi-Party Computation

The goal of *multi-party computation* (MPC) is to perform a computation in a distributed, private, and fault-tolerant way [Yao82]. For this purpose, a fixed set of n parties runs a protocol that tolerates an adversary corrupting a subset of the participating parties. Actively corrupted parties may deviate arbitrarily from the protocol, whereas passively corrupted parties follow the protocol and, intuitively speaking, only try to violate privacy. 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 parties' inputs than what can be deduced from the inputs and outputs of the corrupted parties. *Correctness* means that the protocol either outputs the intended value or no value at all. Privacy and correctness are the two basic requirements. Possible additional requirements are notions of output guarantees, which we discuss in order of decreasing strength: A protocol achieves *robustness* if an adversary cannot abort the computation, preventing the honest parties from obtaining output. *Fairness* is achieved if the honest parties obtain at least 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).

Goldreich et al. [GMW87] provide a first general solution to the MPC problem, based on computational (CO) intractability assumptions and a broadcast (BC) channel. They achieve full security against $t < \frac{n}{2}$ actively corrupted parties, or privacy and correctness only (no fairness or robustness) against $t < n$ actively corrupted parties. If no BC channel is available, privacy and correctness against $t < n$ actively corrupted parties can also be obtained using the BC construction from [FHHW03]. Robust MPC without BC channel is possible if and only if $t < \frac{n}{3}$ parties are corrupted in both the CO and the IT setting [PSL80]. The protocols of both [BGW88] and [CCD88] are IT secure, require no BC channel, and achieve this bound. When a BC channel is available [RB89], or if no robustness but only fairness is required [FHHW03], the bound for IT MPC can be improved to $t < \frac{n}{2}$.

The adversarial setting is defined by the combination of the computational power of the adversary and the cardinality of the set of corrupted parties. Impossibility proofs show that most security guarantees can be achieved simultaneously (i.e. by a single protocol) only in a subset of all adversarial settings. Cleve [Cle86] shows that fairness for general MPC can be achieved only for $t < \frac{n}{2}$ actively corrupted parties. The same bound holds for IT security given a broadcast channel [Kil00]. Ishai et al. [IKLP06] and Katz [Kat07] show that a protocol which guarantees robustness for up to ρ corrupted parties can be secure with abort against at most $n-\rho$ corrupted parties, and describe CO secure protocols that match these bounds.

1.2 Hybrid Security and our Contribution

Conventional, non-hybrid MPC protocols distinguish only between adversarial settings in which they provide all specified security guarantees, and settings in which they provide no security guarantees at all. In contrast, MPC protocols with hybrid security provide different security guarantees for each adversarial setting, without being aware of the actual setting. Hence, they allow for graceful degradation of security.

Specifically, we discuss a protocol providing strong security guarantees for few corruptions in the IT setting, and weaker security guarantees for many corruptions in the CO setting. More precisely, for any robustness parameter $\rho < \frac{n}{2}$ and any static adversary actively corrupting t parties, we describe an MPC protocol π^ρ that simultaneously provides IT security with robustness, correctness and privacy for $t \leq \rho$, IT security with fairness, correctness, and privacy for $t < \frac{n}{2}$, and CO security with agreement on abort, correctness, and privacy for $t < n - \rho$. Hence, our protocol is optimal under the bounds on fairness [Cle86], IT security [Kil00], and the trade-off between robustness and privacy [IKLP06,Kat07].

Our protocol is based on a complete network of synchronous secure channels, a synchronous authenticated broadcast channel, and a common reference string (CRS). The security is proven in the universal composability model [Can01]. In [Luc08,Rau10], we also present results for the stand-alone setting without CRS.

Furthermore, we present a modified version of the protocol with weaker security guarantees, which does not require a broadcast channel.

1.3 Related Work

Chaum [Cha89] sketches a protocol construction secure against passive adversaries that simultaneously guarantees CO privacy for any number of corrupted parties, and IT privacy for a corrupted minority. In contrast to our work, [Cha89] does not discuss the active setting, and hence does not guarantee correctness, fairness, or robustness in case of active corruptions. It is not evident how this protocol would be extended to the active setting. A crucial technique of both Chaum's approach and ours is *party emulation*, which allows a set of parties to implement an additional virtual party for a higher level protocol. This technique was first used in [Bra85] to improve the threshold of broadcast protocols in a setting where privacy is not relevant. Damgård et al. [DIK+08] extend the technique to improve the threshold of general MPC protocols. In [HM00], this technique was discussed in the stand-alone setting for perfectly secure MPC and applied to general adversary structures.

Fitzi et al. [FHHW03] improve 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.

Another work by Fitzi et al. [FHW04] also combines 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 if the underlying PKI is consistent. 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.

Both [FHHW03] and [FHW04] work in a setting without BC channel. When a BC channel is provided, our results improve substantially upon those of [FHHW03,FHW04]. As [FHHW03] only treats IT MPC and [FHW04] only treats robust MPC, both [FHHW03,FHW04] do not reach beyond $t < \frac{n}{2}$ corrupted parties, nor are they easily extended. In contrast, 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].

2 Security Definitions

2.1 Universal Composability

We follow the Universal Composability (UC) paradigm [Can01,BPW04], which uses a simulation-based security model. Here, we only give a high-level overview of the paradigm, see [Can01] for technical details. The security of a protocol (the real world) is defined with respect to a *Trusted Third Party* or *Ideal Functionality* F that correctly performs all computations (the ideal world). Informally, a protocol π is secure if whatever an adversary can achieve in the real world could also be achieved in the ideal world.

More precisely, let $\mathcal{P} = \{1, \ldots, n\}$ be the set of all parties. We only consider static corruptions and use $\mathcal{H} \subseteq \mathcal{P}$ to denote the set of honest parties, and $\mathcal{A} = \mathcal{P} \setminus \mathcal{H}$ to denote the set of corrupted parties. In the *real world*, there is a given set of resources R to which, for each honest party $i \in \mathcal{H}$, a protocol machine π_i is connected. Apart from interacting with the resources, protocol machines provide an interface to higher level protocols for input and output, called an I/O-channel. Corrupted parties access the resources directly which models that they do not adhere to the protocol. The complete real world is denoted by $\pi_{\mathcal{H}}(\mathsf{R})$. In [Can01], resources are modeled as ideal functionalities available in a hybrid model[1] (e.g., authentic or secure channels, broadcast channels). The *ideal world* consists of the ideal functionality F and an ideal adversary (or simulator) $\sigma_{\mathcal{A}}$ connected to F via the interfaces of the corrupted parties $\mathcal{A}$. This ideal world is denoted by $\sigma_{\mathcal{A}}(\mathsf{F})$.

A protocol π is said to securely implement a functionality F if, for every possible set $\mathcal{A}$ of corrupted parties, there is a simulator $\sigma_{\mathcal{A}}$ such that no distinguisher D can tell the real world and the ideal world apart.[2] For this purpose, the distinguisher directly interacts either with the real or with the ideal world, by connecting to all open interfaces, and then outputs a decision bit. This interaction is denoted by $\mathsf{D}(X)$, where $X \in \{\pi_{\mathcal{H}}(\mathsf{R}), \sigma_{\mathcal{A}}(\mathsf{F})\}$.

In [Can01], all protocol machines, simulators, ideal functionalities, and distinguishers are modeled as interactive Turing machines (ITM). We define Σ^{all} as the set of *all* ITMs, and Σ^{eff} as the set of *polynomially bounded* ITMs. Note that, in this paper, ITMs are specified on a higher level of abstraction.

Definition 1. (Universally Composable (UC) Security) *A protocol* π UC securely *implements an ideal functionality* F *if* $\forall \mathcal{A}\ \exists \sigma_{\mathcal{A}} \in \Sigma^{\mathsf{eff}}\ \forall \mathsf{D} \in \Sigma^{\mathsf{eff/all}}$:

$$|Pr[\mathsf{D}(\sigma_{\mathcal{A}}(\mathsf{F})) = 1] - Pr[\mathsf{D}(\pi_{\mathcal{H}}(\mathsf{R})) = 1]| \leq \varepsilon(\kappa)$$

where $\varepsilon(\kappa)$ *denotes a negligible function in the security parameter* κ*. For* $\mathsf{D} \in \Sigma^{\mathsf{eff}}$*, the security is* computational *(CO). For* $\mathsf{D} \in \Sigma^{\mathsf{all}}$*, the security is* information-theoretic *(IT).*

Simulators must be efficient not only in the CO, but also in the IT setting, since otherwise, IT security does not imply CO security. Our formalization of hybrid security uses ideal functionalities that are aware of both the set of corrupted parties and the computational power of the adversary. In other words, the behavior of the functionality, and hence the security guarantees, varies depending on both parameters. A protocol π UC securely implements an ideal functionality F with hybrid-security if π securely implements F in both the CO and the IT setting. Note that, in contrast to [Can01], we use a synchronous communication model with static corruption. The bounds for MPC mentioned in Section 1.1 apply to this model.

In the UC setting, a strong composition theorem can be proven [Can01,BPW04]. This theorem 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.[3]

2.2 Ideal Functionalities for MPC

MPC protocols implement ideal functionalities $\mathcal{E}$ that perform certain computations which are described in some specified language. We describe computations as *programs*[4], i.e. arbitrary sequences of operations on values from a predefined finite field, where each operation is one of input, addition, multiplication, or output.[5] Given such a program $\mathcal{C}$, an MPC functionality $\mathcal{E}[\mathcal{C}]$ evaluates the program operation-wise, and stores intermediate values internally in a vector of

[1] Note that the term "hybrid model" is not related to the notion of hybrid security.

[2] In this model, the adversary is thought of as being part of the distinguisher. Canetti [Can01] shows that this model without adversary is essentially equivalent to a model with adversary, since the security definition quantifies over all distinguishers.

[3] 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. This is in contrast to a stand-alone definition of security where the distinguisher receives output only at the end.

[4] A computation could equivalently be modeled as a circuit.

[5] It is out of scope of this paper how the program is determined. We simply assume that all entities "know" what to do next.

registers. Basically, it works as follows: In case of an input operation, $\mathcal{E}[\mathcal{C}]$ receives an input from a certain party over the corresponding I/O-channel and stores it internally in a new register. In case of an addition operation, $\mathcal{E}[\mathcal{C}]$ adds the two corresponding values internally, and stores the result in a new register. The case of a multiplication operation is handled analogously. In case of an output operation, $\mathcal{E}[\mathcal{C}]$ retrieves the corresponding value from the register, and outputs it to all parties.

The operations defined above capture only deterministic computations with symmetric output. However, probabilistic computations with asymmetric output can be reduced to these operations [Blu81,CDG88]. Furthermore, note that such programs can also perform the computations necessary for realizing an arbitrary protocol machine, thereby allowing for party emulation.

In our work, we are interested in ideal functionalities that evaluate programs with various security guarantees (i.e. only with a subset of the security properties described in Section 1.1, but generally at least encompassing privacy and correctness), depending on the adversarial setting.

Full Security implies *privacy, correctness, and robustness*. Hence, given a program $\mathcal{C}$, an ideal functionality evaluating $\mathcal{C}$ with full security follows the program without deviation.

Fair Security implies *privacy, correctness, and fairness*. Given a program $\mathcal{C}$, an ideal functionality evaluating $\mathcal{C}$ with fair security executes input, addition, and multiplication operations without deviation. However, in case of an output operation, the functionality first requests an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$ the functionality executes the output operation with output to *all* parties, for $o = 0$, the functionality halts.

Abort Security implies *privacy, correctness, and agreement on abort*. Given a program $\mathcal{C}$, an ideal functionality evaluating $\mathcal{C}$ with abort security executes input, addition, and multiplication operations without deviation. In case of an output operation, the functionality first outputs the corresponding value to the adversary and requests an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$, the functionality executes the output operation with output to *all* parties, for $o = 0$, the functionality halts.[6]

No Security: Given a program $\mathcal{C}$, an ideal functionality evaluating $\mathcal{C}$ with no security forwards all inputs from the honest parties to the adversary and lets the adversary determine all outputs for honest parties. As a simulator $\sigma_{\mathcal{A}}^{\mathsf{noSec}}$ can use the inputs of honest parties to simulate honest protocol machines, this already proves the following (rather trivial) lemma:

Lemma 1. *Given a program $\mathcal{C}$, any protocol π UC securely implements the ideal functionality evaluating $\mathcal{C}$ with no security.*

We therefore omit this case from the description of the simulators in this work.

3 A Protocol Overview

Our result for hybrid-secure MPC is formalized by the ideal functionality $\mathcal{E}_{\rho}^{\mathsf{hyb}}[\mathcal{C}]$. This functionality evaluates a program $\mathcal{C}$ with IT full security for $t \leq \rho$ corrupted parties, with IT fair security for $t < \frac{n}{2}$ corrupted parties, and with CO abort security for $t < n - \rho$ corrupted parties (see Figure 1).

We present a protocol π^{ρ} that UC securely implements the functionality $\mathcal{E}_{\rho}^{\mathsf{hyb}}[\mathcal{C}]$ from an n-party communication resource com^n (consisting of a complete network of synchronous secure channels and a synchronous authenticated n-party broadcast channel), and an n-party CRS crs^n drawn from a predefined distribution. A CRS (or an equivalent resource) is required to avoid the impossibility results of [Can01,CF01]. It is possible to minimize the reliance on the crs^n such that our protocols tolerate an adversarially chosen crs^n for few corrupted parties by applying techniques from [GK08,GO07] and a $(t, 2t-1)$-combiner for commitments (e.g. [Her05]).

[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.

Given a robustness parameter $\rho < \frac{n}{2}$ and a program $\mathcal{C}$, the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ evaluates $\mathcal{C}$ according to the number t of corrupted parties and according to the computational power of the adversary (CO or IT).

Setting		Security Guarantees
$t \leq \rho$	IT/CO	evaluate $\mathcal{C}$ with full security
$\rho < t < \frac{n}{2}$	IT/CO	evaluate $\mathcal{C}$ with fair security
$\frac{n}{2} \leq t < n - \rho$	CO	evaluate $\mathcal{C}$ with abort security
$\frac{n}{2} \leq t < n$	IT	evaluate $\mathcal{C}$ with no security
$n - \rho \leq t < n$	CO	

Fig. 1. The ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.

However, this construction is beyond the scope of this paper. In [Luc08,Rau10], we also discuss results for the stand-alone setting without CRS.

The proof of Theorem 1 stating the security of the protocol π^ρ is an application of the UC composition theorem to Lemma 2, Lemma 3, Corollary 1, and Lemma 5.

Theorem 1. *Given a program $\mathcal{C}$ and a robustness parameter $\rho < \frac{n}{2}$, protocol π^ρ UC securely implements the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ evaluating $\mathcal{C}$, from a complete and synchronous network of secure channels and an authenticated broadcast channel com^n, and a common reference string crs^n, in the presence of a static and active adversary.*

3.1 Overview and Key Design Concepts

Basically, the protocol π^ρ consists of three protocol layers: On the lowest layer, the n parties use a technique called *party emulation* to emulate another, $(n+1)^{\mathrm{st}}$ party. The emulation of a party makes it harder for the adversary to control this party: It is not sufficient to corrupt a single party, but, in our case, it is necessary to corrupt at least $\frac{n}{2}$ real parties to control the emulated party. Hence, more trust can be placed in this emulated party. On the middle layer, an $(n+1)$-party MPC protocol is carried out among the n original parties and the emulated party. This protocol provides IT guarantees given that a designated party is honest (this designated party is the emulated party), and CO guarantees otherwise. While it already achieves the correctness, robustness, and fairness requirements (both IT and CO), only the input of the designated party is IT private. The remaining parties obtain only CO privacy. To additionally fulfill the privacy requirement, on the highest layer, the parties exploit this asymmetry: Each party splits its input into two halves, and provides one half as input via a regular party, and the other half as input via the designated party. Additional techniques are required here to maintain correctness and robustness.

More formally, the protocol π^ρ is modularized into three subprotocols: the emulation, the designated party, and the input subprotocol. Each subprotocol implements an ideal functionality on which the next subprotocol is based (see Figure 2).

The goal of the *emulation subprotocol* (Section 4) is to implement a protocol machine π^* belonging to an $(n+1)^{\mathrm{st}}$ party, as formalized by the ideal functionality $\mathsf{emul}[\pi^*]$ (Figure 4). This subprotocol is based on the given resources com^n and crs^n. On the one hand, a protocol machine performs certain computations that can be modeled as a program (see Section 2.2). On the other hand, it communicates with given resources. As a consequence, the subprotocol for party emulation is again modularized into two subprotocols π^{prog} and π^{con}. The subprotocol π^{prog} implements an MPC functionality to evaluate the program of π^* (Section 4.1). The subprotocol π^{con} enables the interaction between the emulated π^* and the resources com^{n+1} and crs^{n+1} (Section 4.2).

The *designated party subprotocol* $\pi^{\mathsf{des},\rho}$ (Section 5) is basically an $(n+1)$-party protocol and implements the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ that provides stronger security guarantees if a designated party is honest. This subprotocol is based on $\mathsf{emul}[\pi^*]$. While we think of protocol

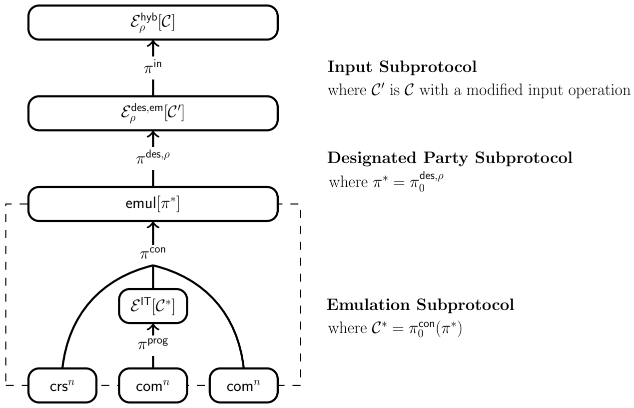

Fig. 2. A visualization of the construction underlying our hybrid-secure MPC protocol π^ρ.

$\pi^{\mathsf{des},\rho}$ as an $n+1$ party protocol, one of the protocol machines ($\pi_0^{\mathsf{des},\rho}$) is actually emulated by the ideal functionality $\mathsf{emul}[\pi^*]$, i.e. in fact we set the parameter $\pi^* = \pi_0^{\mathsf{des},\rho}$ and run an n-party protocol $\pi_1^{\mathsf{des},\rho}, \ldots, \pi_n^{\mathsf{des},\rho}$ on a resource that emulates the $(n+1)^{\text{st}}$ party $\pi_0^{\mathsf{des},\rho}$.

Finally, the *input subprotocol* π^{in} (Section 6) implements the ideal functionality for hybrid security, $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$, based on $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. For this purpose, we define the program $\mathcal{C}'$ evaluated by $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ to be equivalent to the program $\mathcal{C}$ evaluated by $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$, however with a modified input operation taking into account the input splitting and the techniques to preserve correctness and robustness mentioned above.

4 Party Emulation

In this section, we describe a technique for party emulation. This technique allows to emulate an arbitrary protocol machine π^* and to employ it in a higher level protocol. A protocol machine fulfills two tasks which we discuss separately: On the one hand, it performs certain computations (discussed in Section 4.1), on the other hand, it sends and receives messages over its interfaces (discussed in Section 4.2).

4.1 Protocol π^{prog}: Emulating the Computation of π^*

A protocol machine π^* internally performs certain computations. These computations can be modeled as a program that can be evaluated by an MPC functionality, as discussed in Section 2.2. For the purpose of achieving hybrid-secure MPC, we require the emulation of π^* to be IT full secure for $t < \frac{n}{2}$ corrupted parties. This security requirement is captured by the ideal functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ evaluating a program $\mathcal{C}^*$ (Figure 3). Note that in Section 4.2, we define $\mathcal{C}^*$ such that it contains not only the operations performed by π^*, but additional operations enabling communication over the interfaces of π^*.

Given a program $\mathcal{C}^*$, the ideal functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ evaluates $\mathcal{C}^*$, according to the number t of corrupted parties:

Setting		Security Guarantees
$t < \frac{n}{2}$	IT/CO	evaluate $\mathcal{C}^*$ with full security
$\frac{n}{2} \leq t$	IT/CO	evaluate $\mathcal{C}^*$ with no security

Fig. 3. The ideal functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$.

The subprotocol π^{prog} has to implement the ideal functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ from a complete network of synchronous secure channels and a BC channel. As stated in Lemma 2, the protocol presented in [RB89] already fulfills all requirements.

Lemma 2 ([RB89,CDD+99,Can01]). *Given an arbitrary program $\mathcal{C}^*$, there is a protocol that UC securely implements the ideal functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ from a complete network of synchronous secure channels and a BC channel com^n, in the presence of a static and active adversary.*

4.2 Protocol π^{con}: Connecting π^* to the Resources

In the previous section, we described a protocol π^{prog} that emulates the computations of a protocol machine π^*. In this section, we describe a protocol π^{con} that enables the communication over the interfaces of the emulated π^*. This communication includes the interaction with the resources (i.e. a complete network of synchronous secure channels and a BC channel for $n+1$ parties, denoted by com^{n+1}, and an $(n+1)$-party CRS crs^{n+1}), as well as the input from and output to the emulating parties. Before we present the protocol π^{con}, we give a formal description of the ideal functionality $\mathsf{emul}[\pi^*]$ implemented by π^{con}.

The Ideal Functionality for Emulated Parties We now give a description of the ideal functionality $\mathsf{emul}[\pi^*]$ (summarized in Figure 4). Basically, $\mathsf{emul}[\pi^*]$ internally emulates the resources com^{n+1} and crs^{n+1}, and a protocol machine π^*. This protocol machine is given to $\mathsf{emul}[\pi^*]$ as a parameter, and is an arbitrary protocol machine with the condition that it has a com^{n+1} and a crs^{n+1} interface, and n I/O-channels (instead of a single one as usual). These n I/O-channels belong to the n parties emulating π^*. Accordingly, the interface of $\mathsf{emul}[\pi^*]$ for each party consists of three subinterfaces: Each party has access to its corresponding interfaces of com^{n+1} and crs^{n+1}, and may provide input to and receive output from π^* over its corresponding I/O-channel to π^*.

For $t < \frac{n}{2}$ corrupted parties, $\mathsf{emul}[\pi^*]$ internally emulates the ideal functionalities com^{n+1} and crs^{n+1}, and the protocol machine π^*, and connects them accordingly. In this setting, the protocol machine π^* works according to its specification, i.e. can be considered honest. We denote this behavior of $\mathsf{emul}[\pi^*]$ with $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}, \pi^*]$.

For $t \geq \frac{n}{2}$, $\mathsf{emul}[\pi^*]$ transfers control over the (emulated) protocol machine π^* to the adversary. Hence, $\mathsf{emul}[\pi^*]$ only emulates com^{n+1} and crs^{n+1}, and gives the adversary access not only to the interfaces of com^{n+1} and crs^{n+1} belonging to corrupted parties, but also to the interfaces belonging to the emulated protocol machine π^*. Furthermore, the I/O-channels from honest parties to π^* are directly connected to the adversary. We denote this behavior of $\mathsf{emul}[\pi^*]$ with $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}]$.

Given a protocol machine π^*, the ideal functionality $\mathsf{emul}[\pi^*]$ evaluates a program either for $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}, \pi^*]$ or for $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}]$, according to the number t of corrupted parties:

Setting		Security Guarantees
$t < \frac{n}{2}$	IT/CO	evaluate the program for $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}, \pi^*]$ with full security
$\frac{n}{2} \leq t$	IT/CO	evaluate the program for $[\mathsf{crs}^{n+1}, \mathsf{com}^{n+1}]$ with full security

Fig. 4. The ideal functionality $\mathsf{emul}[\pi^*]$.

Protocol π^{con} Here, we describe a protocol π^{con} that implements the ideal functionality $\mathsf{emul}[\pi^*]$, based on the ideal functionalities $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ (evaluating a program $\mathcal{C}^*$ to be defined below),

com^n, and crs^n.[7] The task of π^{con} is twofold: On the one hand, it turns the n-party resources com^n and crs^n into resources for $n+1$ parties. On the other hand, it connects the emulated protocol machine π^* (running as part of $\mathcal{C}^*$ on $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$) to these resources. The protocol π^{con} consists of the n protocol machines $\pi_1^{\mathsf{con}}, \ldots, \pi_n^{\mathsf{con}}$ run by the real parties, and an additional protocol machine denoted by π_0^{con} that runs as part of $\mathcal{C}^*$ on $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$. In fact, π_0^{con} is a wrapper for π^*, and we parametrize $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ with the program that emulates π_0^{con} connected to all interfaces of π^*, which we denote by $\mathcal{C}^* = \pi_0^{\mathsf{con}}(\pi^*)$.

Each protocol machine π_i^{con} $(i \in \{1, \ldots, n\})$ is connected to the resources com^n, crs^n and $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$. In turn, via its I/O-channel to higher level protocols, π_i^{con} provides access to a com^{n+1}, a crs^{n+1}, and one I/O-channel of π^*. The protocol machine π_0^{con} is connected to all interfaces of π^*. Furthermore, each one of the n interfaces of $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ (the ideal functionality evaluating $\mathcal{C}^* = \pi_0^{\mathsf{con}}(\pi^*)$) serves as a secure channel between π_0^{con} and a protocol machine π_i^{con} $(i \in \{1, \ldots, n\})$. Using these secure channels, protocol π^{con} enables the interaction between π^* and the resources com^n and crs^n, and thereby extends these resources to $(n+1)$-party resources.

Basically, the protocol machines π_i^{con} only receive messages and forward them on the correct interface and with the correct label. For broadcast messages and the CRS, consensus among all parties is guaranteed by additionally performing a majority vote on messages relating to π_0^{con}. Given that $t < \frac{n}{2}$ parties are corrupted, this results in a correct $(n+1)$-party broadcast and CRS. For $t \geq \frac{n}{2}$ corrupted parties, π^* is controlled by the adversary and it is not necessary to securely extend the resources. Figures 5 and 6 provide a technical description of the protocol machines π_0^{con} and π_i^{con} $(i \in \{1, \ldots, n\})$, respectively.

Lemma 3. *Protocol π^{con} UC securely implements $\mathsf{emul}[\pi^*]$ from com^n, crs^n, and the functionality $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ evaluating $\mathcal{C}^* = \pi_0^{\mathsf{con}}(\pi^*)$, in the presence of a static and active adversary.*

Proof. In order to prove Lemma 3, we need to provide a simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ such that the ideal model $\sigma_{\mathcal{A}}^{\mathsf{con}}(\mathsf{emul}[\pi^*])$ becomes IT indistinguishable from the real model $\pi_{\mathcal{H}}^{\mathsf{con}}(\mathsf{com}^n \parallel \mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*] \parallel \mathsf{crs}^n)$. The simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ connects to all interfaces of $\mathsf{emul}[\pi^*]$ associated with corrupted parties. The interface exposed by $\mathsf{emul}[\pi^*]$ to an honest party consists of a subinterface to the network com^{n+1}, a subinterface to the crs^{n+1}, and a subinterface to one I/O-interface of the protocol machine π^* (all these machines are internally emulated by $\mathsf{emul}[\pi^*]$). The interface provided to the simulator depends on the number of corrupted parties. In case $t < \frac{n}{2}$, the interfaces for the corrupted parties are identical to the interfaces provided to honest parties. In case $t \geq \frac{n}{2}$, intuitively speaking, the emulation of π^* fails and is controlled by the adversary. Hence, in addition to the previously mentioned interfaces, the I/O-channels to π^* belonging to honest parties are connected to the simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$. Furthermore, $\sigma_{\mathcal{A}}^{\mathsf{con}}$ has access to the com^{n+1}- and crs^{n+1}-interfaces belonging to party 0.

The simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ internally simulates an instance $\widetilde{\mathsf{com}^n}$ of com^n, an instance $\widetilde{\mathsf{crs}^n}$ of crs^n, and copies $\widetilde{\pi_i^{\mathsf{con}}}$ of π_i^{con} for the honest parties $i \in \mathcal{H}$. Furthermore, for $t < \frac{n}{2}$, $\sigma_{\mathcal{A}}^{\mathsf{con}}$ internally simulates a copy $\widetilde{\pi_0^{\mathsf{con}}}$ of π_0^{con}. Note that $\widetilde{\pi_0^{\mathsf{con}}}$ is not connected to a protocol machine $\widetilde{\pi^*}$, i.e. in addition to the usual n interfaces to the π_i^{con} $(i \in [n])$, $\widetilde{\pi_0^{\mathsf{con}}}$ has interfaces corresponding to the n I/O-interfaces, the com^{n+1} and the crs^{n+1} of π^*.

These simulated machines are connected as in protocol π^{con}, i.e. each $\widetilde{\pi_i^{\mathsf{con}}}$ is connected to $\widetilde{\mathsf{com}^n}$ and $\widetilde{\mathsf{crs}^n}$. The simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ uses the (unconnected) I/O-interface of the $\widetilde{\pi_i^{\mathsf{con}}}$ for the simulation. The (unconnected) interfaces of $\widetilde{\mathsf{com}^n}$ and $\widetilde{\mathsf{crs}^n}$ belonging to corrupted parties are exposed to the distinguisher. For $t < \frac{n}{2}$, the channels from $\widetilde{\pi_0^{\mathsf{con}}}$ towards the honest π_i^{con} are connected to the simulated $\widetilde{\pi_i^{\mathsf{con}}}$ $(i \in \mathcal{H})$. The corresponding channels towards the corrupted π_i^{con} are exposed to the distinguisher. The interfaces of $\widetilde{\pi_0^{\mathsf{con}}}$ to the I/O-channels of π^* belonging to corrupted parties are connected to the corresponding subinterfaces of $\mathsf{emul}[\pi^*]$. The interfaces

[7] As stated in Theorem 1, we assume a single resource com^n. However, for the emulation subprotocol, we require two independent copies of com^n, which can be achieved by multiplexing the given com^n.

of $\widetilde{\pi_0^{\mathsf{con}}}$ to the I/O-channels of π^* belonging to honest parties are left unconnected (we argue below why there are no messages over these channels). The simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ uses the (unconnected) interface of $\widetilde{\pi_0^{\mathsf{con}}}$ to the com^{n+1}- and crs^{n+1}-interface of π^* for the simulation. If $t \geq \frac{n}{2}$, there is no $\widetilde{\pi_0^{\mathsf{con}}}$, and the channels from $\widetilde{\pi_i^{\mathsf{con}}}$ towards π_0^{con} are exposed to the distinguisher. Furthermore, as mentioned above, $\mathsf{emul}[\pi^*]$ connects the I/O-channels to π^* belonging to honest parties directly to $\sigma_{\mathcal{A}}^{\mathsf{con}}$, which in turn forwards them to the distinguisher.

At the beginning of the simulation, the simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ retrieves the CRS from $\mathsf{emul}[\pi^*]$ and sets the simulated $\widetilde{\mathsf{crs}^n}$ to the same value. Now, when a (private or BC) message m is received from an honest party i via com^{n+1} ($i \in \mathcal{H}$), the simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ inputs m over the I/O-interface to the simulated $\widetilde{\pi_i^{\mathsf{con}}}$, which in turn will send corresponding messages over the simulated $\widetilde{\mathsf{com}^n}$ and/or to π_0^{con}. On the other hand, when a $\widetilde{\pi_i^{\mathsf{con}}}$ outputs a (private or BC) message m from a corrupted party j on the I/O-interface, this means party j has sent the message m and simulator $\sigma_{\mathcal{A}}^{\mathsf{con}}$ forwards m to com^{n+1} via the interface of party j.

For $t < \frac{n}{2}$, (private and BC) messages on the com^{n+1}-interface of $\widetilde{\pi_0^{\mathsf{con}}}$ towards π^* are handled in the same way as for honest parties. Messages on the crs^{n+1}-interface of $\widetilde{\pi_0^{\mathsf{con}}}$ towards π^* are ignored. Now we argue why there are no messages on the interfaces of $\widetilde{\pi_0^{\mathsf{con}}}$ to the I/O-channels of π^* belonging to honest parties. In the real world, messages from honest parties to the I/O-interace of π^* are first input to the corresponding π_i^{con}, from there forwarded to π_0^{con} and further onwards to π^* (or the other way round). In the ideal world, the simulator does not obtain these messages. Hence, these interfaces are unused (indistinguishability is maintained, see below).

For $t \geq \frac{n}{2}$, the simulator connects both the channels from $\widetilde{\pi_0^{\mathsf{con}}}$ ($i \in \mathcal{H}$) towards π_0^{con}, as well as the I/O-channels from honest parties towards π^* (provided by $\mathsf{emul}[\pi^*]$ for $t \geq \frac{n}{2}$) directly to the distinguisher (note that this is perfectly indistiguishable from the real world).

It is fairly straightforward to see that the real and the ideal world are perfectly indistinguishable. All information that is revealed to the adversary in the real world, is revealed to the simulator in the ideal world. Therefore, the simulation consists only of copying and duplicating the messages on the corresponding channels (as described by the protocol π^{con}). Note that this observation holds also for messages sent and received by party 0 in the case $t < \frac{n}{2}$. For private and BC messages, the argument is identical as for honest parties. The I/O-channels to π^* belonging to corrupted parties in the ideal world are connected identically as in the real world. Note that messages over I/O-channels to π^* belonging to honest parties are not simulated. As described above, in the real world, these messages pass through the system without the adversary noticing it (in the same sense as for secret messages between honest parties). Hence, a simulation is not necessary.

5 MPC with a Designated Party

In this section, we present an $(n+1)$-party protocol $\pi^{\mathsf{des},\rho}$ that implements an ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ for designated party MPC from a common reference string crs^{n+1} and an $(n+1)$-party communication resource com^{n+1}. The behavior (and hence the provided security guarantees) of the ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ depend in particular on a designated party. For ease of notation, we assume that this designated party is party 0, and that the remaining parties are numbered $1, \dots, n$.

For the sake of simplicity, we describe the designated party protocol as a usual $(n+1)$-party protocol, without taking into account that one of the parties is emulated. The security of the corresponding subprotocol in our construction can easily be derived from the security of this $(n+1)$-party protocol, and is stated in Corollary 1.

π_0^{con} connects to *all* interfaces of protocol machine π^*, i.e. to the com^{n+1} interface, the crs^{n+1} interface and to the n I/O-interfaces. In turn, π_0^{con} makes direct use of the n interfaces of $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ as secure channels to the π_i^{con} ($i \in \{1,\ldots,n\}$). π_0^{con} then processes messages as follows:

Secure Channels: Messages from π_i^{con} that are labeled as secret message from party i are forwarded to the com^{n+1}-interface of π^*. Messages for party i from the com^{n+1}-interface of π^* are labeled as secret message from the emulated party and sent to π_i^{con}.

I/O for π^*: Messages from π_i^{con} labeled as input to π^* are forwarded to the protocol machine π^* as input to the I/O-interface of party i. Outputs from π^* on the I/O-interface of party i are labeled as such and sent to π_i^{con}.

Broadcasts: Messages labeled as broadcast messages from party j, if received identically from more than $\frac{n}{2}$ protocol machines π_i^{con}, are forwarded to the com^{n+1}-interface of π^* as broadcast messages from party j. Otherwise, they are ignored. Broadcast messages from the com^{n+1}-interface of π^* are labeled as broadcast messages from the emulated party and sent to all π_i^{con} ($i \in \{1,\ldots,n\}$).

CRS: A message labeled as CRS, if received identically from more than $\frac{n}{2}$ protocol machines π_i^{con}, is forwarded to the crs^{n+1} interface of π^*. Otherwise, it is ignored.

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

π_i^{con} connects to the interfaces of com^n, crs^n and $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ belonging to party i. The interface to $\mathcal{E}^{\mathsf{IT}}[\mathcal{C}^*]$ serves as a secure channel to π_0^{con}. Via its I/O-channel, π_i^{con} provides access to a com^{n+1}, a crs^{n+1}, and one I/O-channel of π^*. π_i^{con} then processes messages as follows:

Secure Channels: Messages arriving on the I/O-channel labeled as secret messages for party j ($j \in \{1,\ldots,n\}$) are forwarded to com^n. Messages from party j arriving on the com^n-interface are output over the I/O-channel as messages from party j. Messages arriving on the I/O-channel labeled as secret messages for π^* are forwarded to π_0^{con}. Messages from π_0^{con} labeled as secret message from π^* to party i are output over the I/O-channel as messages from π^*.

I/O for π^*: Messages arriving on the I/O-channel labeled as inputs to π^* are forwarded to π_0^{con}. In turn, messages from π_0^{con} labeled as outputs from π^* are output over the I/O-channel.

Broadcasts from party $i \in \{1,\ldots,n\}$: Broadcast messages arriving on the I/O-channel are forwarded to com^n and to π_0^{con}, labeled as broadcast messages from party i. Broadcast messages from party j arriving on the com^n-interface (unless labeled as originating from π^*) are both output over the I/O-channel and forwarded to π_0^{con}, labeled as broadcast from party j.

Broadcasts from π^*: Messages from π_0^{con} labeled as broadcast messages from π^* are forwarded as broadcast messages to com^n (including the label). Broadcast messages arriving on the com^n-interface labeled as originating from π^*, if received identically from more than $\frac{n}{2}$ parties j, are output over the I/O-channel as broadcast message from π^*.

CRS: On first activation, π_i^{con} retrieves the CRS from the crs^n functionality, labels it accordingly, and both outputs it over the I/O-channel and sends it to π_0^{con}.

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

5.1 The Ideal Functionality for Designated Party MPC

The ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ for $n+1$ parties evaluates an arbitrary program $\mathcal{C}'$ and provides stronger security guarantees if the designated party 0 is honest. In the following descriptions, the number t of corrupted parties always pertains to the parties $1,\ldots,n$, and never includes the designated party 0, which is treated separately. If the designated party 0 is honest, functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ guarantees IT security with correctness and fairness for all parties, as well as IT privacy for the input from party 0, against any number of corrupted parties. Additionally, it guarantees IT robustness against $t \leq \rho$ corrupted parties. If the designated party 0 is corrupted, functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ still provides CO security with correctness and privacy to the honest parties against $t < n - \rho$ corrupted parties. The functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ is described in Figure 7.

5.2 A Designated Party MPC Protocol

We now describe a designated party MPC protocol $\pi^{\mathsf{des},\rho}$ which implements the $(n+1)$-party functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ for designated party MPC from a common reference string crs^{n+1} and

Given a robustness parameter $\rho < \frac{n}{2}$ and a program $\mathcal{C}'$, the ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ evaluates $\mathcal{C}'$ according to the computational power of the adversary (CO or IT), the honesty of party 0, and the number t of corrupted parties in $\{1, \ldots, n\}$. $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ provides one interface to each party $i \in \{1, \ldots, n\}$, and n interfaces to party 0.

Setting			**Security Guarantees**				
IT/CO	Party 0	t	Priv. party 0	Priv. party i	Cor.	Fair.	Rob.
IT	honest	$t \leq \rho$	yes	no	yes[8]	yes	yes
		$\rho < t \leq n$	yes	no	yes[8]	yes	no
	corrupted	$0 \leq t \leq n$	*(corrupted)*	no	no	no	no
CO	honest	$t \leq \rho$	yes	yes	yes	yes	yes
		$\rho < t \leq n$	yes	yes	yes	yes	no
	corrupted	$t < n - \rho$	*(corrupted)*	yes	yes	no	no
		$n - \rho \leq t \leq n$	*(corrupted)*	no	no[9]	no	no

Addition and multiplication operations are always executed without deviation. In contrast, the execution of input and output operations depend on the security guarantees:
Privacy for party $j \in \{0, \ldots, n\}$: Execute input operations for party j without deviation.
No privacy for party $j \in \{0, \ldots, n\}$: The input is additionally sent to the adversary.
Robustness: Execute output operations without deviation.
No robustness but fairness: In case of an output operation, request an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$, execute the output operation with output to *all* parties, for $o = 0$ halt.
No fairness but correctness: In case of an output operation, output the corresponding value to the adversary and request an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$, execute the output operation with output to *all* parties, for $o = 0$ halt.
No correctness: Receive a value from the adversary and output this value to all parties.

Fig. 7. The ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$.

an $n + 1$ party communication resource com^{n+1}. We obtain protocol $\pi^{\mathsf{des},\rho}$ by adapting the CO MPC protocol of [CLOS02] to our needs.

The protocol in [CLOS02] evaluates programs consisting of input, addition, multiplication, and output operations (see Section 2.2). During an *input* operation, the party providing input commits to its input and shares it among all parties according to a predefined 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. *Addition* operations are evaluated locally. To evaluate a *multiplication* operation, [CLOS02] uses oblivious transfer (OT) primitives. All intermediate results are computed as sharings, and each party is committed to its shares. To achieve security against active adversaries, each party proves the correctness of the messages it sends using zero-knowledge (ZK) proofs. During *output* operations, each party opens the commitment to its share of the final result, which can then be reconstructed locally.

Essentially, to achieve the asymmetric security guarantees of $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$, we need to provide IT security guarantees to the designated party, without compromising the original CO security guarantees for the remaining parties. In the following, we describe how the components in [CLOS02] can be modified accordingly.

Summary of Modifications. The protocol in [CLOS02] is based on three primitives: OT, commitment, and ZK proofs. Basically, these primitives are two-party primitives, and each one can be implemented such that one of the two parties obtains IT security guarantees, while the other party still has CO guarantees. Below, a detailed discussion of suitable primitives can be found. Hence, in each invocation between the designated party and a normal party, the

[8] 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.

[9] Our protocol $\pi^{\mathsf{des},\rho}$ could be modified to achieve correct and input-independent non-interactive MPC in this case. For our subsequent results, though, we need not demand correctness here.

designated party can be protected against an IT adversary. Using such primitives is merely a refinement of [CLOS02], so, in the CO setting, the original security guarantees are still valid.

Furthermore, we need to modify the sharing scheme and the output reconstruction in [CLOS02] in order to meet the robustness and fairness requirements. We have to robustly tolerate $t \leq \rho$ corruptions among the parties $i \in \{1, \ldots, n\}$, while preserving the unconditional privacy of party 0. This can be accomplished by modifying the underlying sharing of [CLOS02] as described in [Gol04, Section 7.5.5]. We use a sharing scheme where any set M of $n - \rho + 1$ parties that *includes* party 0 is qualified, i.e., can reconstruct. Such a sharing can efficiently be implemented using a $(2n - \rho)$-out-of-$(2n)$ Shamir-sharing where party 0 receives n shares and each remaining party 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, on the one hand, any qualified set M of honest parties can recover the input of up to ρ corrupted parties $i \in \{1, \ldots, n\}$. This ensures robustness, when $t \leq \rho$ 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 party 0 is honest. As noted in [Gol04], a party can violate fairness only if it holds the last share required for reconstruction and all other parties already opened their commitments. In our case, party 0 is always required for reconstruction. Hence, if we specify that party 0 opens last and only if it can contribute sufficiently many shares such that all parties can reconstruct the result, then the resulting protocol $\pi^{\mathsf{des},\rho}$ is fair in the IT setting as long as party 0 is honest.

Primitives Providing IT Security for a Designated Party. We discuss suitable CO primitives for [CLOS02], 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 0.

Oblivious Transfer. As shown in [CLOS02, Section 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] 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 party 0 by making party 0 the receiver in every invocation of OT involving party 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, Theorem 4.1]. Thus, security for party 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 Appendix A. For our purpose, we employ the IT binding variation for commitments issued by the parties $i \in \{1, \ldots, n\}$, and the IT hiding variation for commitments issued by the designated party 0. Thus we obtain a UC secure realization of the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com},1:\mathsf{M}}$ that guarantees security for party 0 against any IT adversary.

ZK proofs. [CLOS02, Prop. 9.4] shows how to UC securely implement the ZK functionality $\mathsf{F}_{\mathsf{ZK},1:\mathsf{M}}$ from the commitment functionality $\mathsf{F}_{\mathsf{Com},1:\mathsf{M}}$ without use of further CO assumptions.[10] The one-to-many ZK protocol of [CLOS02] is based on the two party protocol of [CF01, Section 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

[10] 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.

is obtained by repeating the two-party protocol with each party over the broadcast channel. In addition, the proof is accepted only if the transcripts of all invocations constitute valid proofs. Now, if party 0 is the prover, we instantiate the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com},1:\mathsf{M}}$ with the IT hiding scheme described above. Otherwise, we instantiate $\mathsf{F}_{\mathsf{Com},1:\mathsf{M}}$ with the IT binding scheme. Thus we obtain a protocol that is always secure for party 0, even against any IT adversary.

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 $\mathsf{F}_{\mathsf{CP},1:\mathsf{M}}$. [CLOS02, Section 7.1] provides a protocol implementing the two-party[11] functionality F_{CP} secure against static adversaries, which relies only on F_{ZK} and a standard commitment scheme C (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 $\mathsf{F}_{\mathsf{ZK},1:\mathsf{M}}$ instead of F_{ZK}. Furthermore, we modify the implementation of $\mathsf{F}_{\mathsf{CP},1:\mathsf{M}}$ to use the functionality $\mathsf{F}_{\mathsf{ZK},1:\mathsf{M}}$ as described above, and (standard) IT hiding (party 0) or IT binding (parties $1,\ldots,n$) commitments for C to IT protect the designated party 0. Further CO assumptions are not required. Thus, we can implement the commit-and-prove functionality $\mathsf{F}_{\mathsf{CP},1:\mathsf{M}}$ UC securely for party 0, even against any IT adversary.

5.3 The Security of the Designated Party Protocol

The modifications to the protocol from [CLOS02] result in a protocol $\pi^{\mathsf{des},\rho}$ that provides the designated party 0 with IT security guarantees, while protecting the remaining parties with CO security.

Lemma 4. *Given an arbitrary program $\mathcal{C}'$ and a robustness parameter $\rho < \frac{n}{2}$, protocol $\pi^{\mathsf{des},\rho}$ UC securely implements the ideal functionality $\mathcal{E}^{\mathsf{des}}_{\rho}[\mathcal{C}']$ evaluating $\mathcal{C}'$, from a complete and synchronous network of secure channels and an authenticated broadcast channel com^{n+1}, and a common reference string crs^{n+1}, in the presence of a static and active adversary.*

Proof. We will now show that the modified version of [CLOS02] described in Section 5.2 fulfills the security requirements formalized by the ideal functionality $\mathcal{E}^{\mathsf{des}}_{\rho}[\mathcal{C}']$ (Figure 7).

Case 1: CO security with robustness for $t \leq \rho$ and fairness beyond, party 0 honest. 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, party 0 corrupted. CO correctness and privacy are already implied by [CLOS02]. 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]. The shares observed by corrupted parties are still uniformly random for $t < n - \rho$ corrupted parties. Therefore, the modifications to the simulator described in [CLOS02] remain trivial, and the proof in [CLOS02] remains applicable with minimal modifications. In summary, 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.

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

Case 3: IT security for an honest party 0 *with robustness for* $t \leq \rho$*, and with fairness for* $t < n - \rho$*.* We sketch a simulator to demonstrate that [CLOS02], tweaked as described above, UC securely implements the ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ in the given setting. That means, the protocol guarantees correctness, privacy for party 0, and fairness for an IT adversary corrupting $t < n - \rho$ parties (not including party 0), and additionally guarantees robustness for $t \leq \rho$ (again not including party 0).

The simulator receives the inputs of all honest parties except party 0 from the ideal functionality $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$. It then simulates the protocol machine of the (honest) party 0 on arbitrary input x'_0, and the protocol machines of the remaining honest parties on the input received by $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$. Furthermore corrupted parties have to commit to their input using binding UC commitments[12], so by extractability the simulator can extract their inputs and forward them to $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$. The simulation then proceeds up to the point where an output y is opened, i.e. where the simulator receives a y from $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$. Recall that party 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 party 0 open the commitments to its shares of the output such that the value y is reconstructed, and sets the output flag to $o = 1$. If less then $t - \rho$ corrupted parties open their commitments, the simulator only sets the output flag to $o = 0$ and halts.

Given that $t \leq \rho$ parties are corrupted, the adversary can no longer prevent the honest parties from reconstructing the output. Hence, party 0, regardless of the shares distributed by the adversary, opens the output to y, again making use of the equivocability of commitments.

The behavior of the simulator is IT indistinguishable from the real protocol. Note that the only distinguishable difference is the value of the input of party 0. As in the real protocol, the designated party 0 in the simulation only converses with the other parties by means of hiding commitments, ZK proofs and OT invocations IT protecting party 0. Furthermore, the sharing scheme is such that without cooperation of the designated party 0, which is honest by assumption, no information can be recovered. As such no information whatsoever is disseminated by the designated party 0 to the corrupted parties until reconstruction takes place in an opening phase. Therefore, the simulated input of party 0 x'_0 cannot be distinguished from the real input x_0.

Case 4: IT, party 0 *corrupted* ($t \geq \frac{n}{2}$)*, no security guarantees.* As we make no security guarantees in this case, indistinguishability follows from Lemma 1.

Computational assumptions sufficient for implementing the necessary 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. A similar approach, where *all* parties use primitives that IT disclose no undesired information is used in [KMR09] to achieve long-term security for specific functions.

5.4 The Ideal Functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$

The protocol $\pi^{\mathsf{des},\rho}$ does not fit directly into the setting of the overall protocol π^ρ: On the one hand, in protocol π^ρ, there are only n parties running a protocol based on the functionality $\mathsf{emul}[\pi^*]$ emulating $\pi^* = \pi_0^{\mathsf{des},\rho}$ and the resources com^{n+1} and crs^{n+1}. On the other hand, these n parties have to implement the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, and not $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$, where the only

[12] Providing extractability and equivocability by means of the crs^n, see Appendix A.

difference is that $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ is specialized for a setting where the designated party 0 is emulated. That means, the security guarantees provided by $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ do not depend on the honesty of party 0, but on the bound $t < \frac{n}{2}$ for a correct emulation (see Section 4.1). More to the point, IT privacy, correctness and fairness can be violated only by a corrupted majority. Furthermore, the input for party 0 is provided by the n real parties. For this purpose, $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ provides a second interface to each party, corresponding to one of the n interfaces of party 0 to $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$. Inputs over these interfaces are treated as input from party 0 when evaluating the program $\mathcal{C}'$. That is, the privacy of these inputs is guaranteed even against IT adversaries.

A formal description of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ is obtained by replacing the condition on the honesty of party 0 in Figure 7 by the bound $t < \frac{n}{2}$, and adjusting the interfaces accordingly (see Figure 8).

Given a program $\mathcal{C}'$, the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ evaluates $\mathcal{C}'$ according to the computational power of the adversary (CO or IT), and the number t of corrupted parties. $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ provides each party i with two interfaces. One interface is the regular interface that is equivalent to the interface provided by $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ to party i. The other interface is called the party-0-interface and corresponds to one of the n interfaces provided by $\mathcal{E}_\rho^{\mathsf{des}}[\mathcal{C}']$ to party 0.

Setting		**Security Guarantees**				
IT/CO	t	Privacy party-0-interfaces	Privacy regular interface i	Cor.	Fair.	Rob.
IT	$t \leq \rho$	yes	no	yes[8]	yes	yes
	$\rho < t < \frac{n}{2}$	yes	no	yes[8]	yes	no
	$\frac{n}{2} \leq t \leq n$	no	no	no	no	no
CO	$t \leq \rho$	yes	yes	yes	yes	yes
	$\rho < t < \frac{n}{2}$	yes	yes	yes	yes	no
	$\frac{n}{2} \leq t < n - \rho$	no	yes	yes	no	no
	$n - \rho \leq t \leq n$	no	no	no[9]	no	no

Addition and multiplication operations are always executed without deviation. In contrast, the execution of input and output operations depend on the security guarantees:
Privacy for party-0-interface or regular interface i: Execute input operations over this interface without deviation.
No privacy for a regular interface: Input over this interface is additionally sent to the adversary.
No privacy for a party-0-interface: The input is sent to the adversary, who might replace it. In other words, in that case, the party-0-interfaces are controlled by the adversary.
Robustness: Execute output operations without deviation.
No robustness but fairness: In case of an output operation, request an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$, execute the output operation normally (with output to *all* parties), for $o = 0$ halt.
No fairness but correctness: In case of an output operation, output the corresponding value to the adversary and request an output flag $o \in \{0, 1\}$ from the adversary (default is $o = 1$ if the adversary makes no suitable input). Then, for $o = 1$, execute the output operation normally (with output to *all* parties), for $o = 0$ halt.
No correctness: Receive a value from the adversary and output this value to all parties.

Fig. 8. The ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$.

Basically, the following corollary is only a technical modification of Lemma 4 to our needs:

Corollary 1. *Given an arbitrary program $\mathcal{C}'$ and a robustness parameter $\rho < \frac{n}{2}$, the protocol machines $\pi_1^{\mathsf{des},\rho}, \ldots, \pi_n^{\mathsf{des},\rho}$ UC securely implement the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ evaluating $\mathcal{C}'$, from the ideal functionality* $\mathsf{emul}[\pi^*]$ *parametrized with $\pi^* = \pi_0^{\mathsf{des},\rho}$, in the presence of a static and active adversary.*

6 Realizing Hybrid-Secure MPC

In this section, we describe an n-party protocol π^{in} implementing a hybrid-secure MPC functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ (Figure 1) based on the designated party MPC functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ (see

Section 5.4), and the program $\mathcal{C}'$ to be evaluated by $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. We introduce one by one the three techniques used to (1) fulfill the privacy requirement of $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$, while (2) preserving the correctness and (3) the robustness provided by $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$.

(1) The functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ specifies *IT privacy* for $t < \frac{n}{2}$ and *CO privacy* for $t < n - \rho$. The protocol π^{in} achieves this requirement by having π_i^{in} share any input x_i as $x_i = x_i^{\mathsf{it}} \oplus x_i^{\mathsf{co}}$, where x_i^{it} is chosen uniformly at random over the input space.[13] Recall that $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ provides each party with two interfaces: a regular one, and one that corresponds to one of the n interfaces of the designated party 0 (called party-0-interface). The protocol machine π_i^{in} inputs x_i^{co} at its regular interface to functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, while entering the share x_i^{it} via its party-0-interface to functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. Functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ guarantees IT privacy for inputs over a party-0-interface for $t < \frac{n}{2}$, and CO privacy for inputs over a regular interface for $t < n - \rho$. The input splitting combines the two privacy guarantees and, hence, accomplishes the privacy requirements of $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.

(2) To preserve *CO correctness* for $t \geq \frac{n}{2}$, additional measures are needed: For $t \geq \frac{n}{2}$, the party-0-interfaces are controlled by the adversary, who could manipulate the input x_i^{it} at will, effectively manipulating the inputs x_i to produce *incorrect* results. This adversarial behavior can be prevented using commitments: Let commit and open denote the respective procedures of a UC secure IT hiding commitment scheme (see [DN02], Appendix A). First, π_i^{in} computes an IT hiding commitment to x_i^{it} together with its opening information $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{it}})$. Then, it inputs the commitment c_i together with x_i^{co} at its regular interface to functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, while entering the matching opening information o_i together with x_i^{it} at its party-0-interface to functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. Functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ checks these commitments. The case where a commitment fails to open correctly is treated in the next paragraph. This construction achieves CO correctness, because a CO adversary controlling the party-0-interfaces cannot open such a commitment incorrectly. At the same time, the unconditional privacy of the x_i^{it} is unaffected as the commitments c_i are IT hiding.

(3) Finally, we need to preserve *IT robustness* for $t \leq \rho$. Essentially, this means that $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ may not simply abort if a commitment c_i fails to open correctly. Instead, it outputs a complaint, requesting that party i inputs x_i directly via its regular interface to $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. This procedure does not affect privacy, since commitments c_i fail to open correctly only if either party i is corrupted or if the party-0-interfaces are controlled by the adversary. In the first case, we need not guarantee privacy to party i. In the latter case, we have $t \geq \frac{n}{2}$ corrupted parties, so we only need to guarantee CO privacy of x_i, which $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ already does. Correctness is maintained since privacy is maintained and a party can only replace its own input.

The IT fairness properties of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ are unaffected by the measures described above, so the resulting protocol is fair whenever $t < \frac{n}{2}$ parties are corrupted, i.e. whenever the designated party 0 is honest.

Summarizing the measures above, we obtain a protocol π^{in} (Figure 9) and a matching program $\mathcal{C}'$ (Figure 10) to be evaluated by functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. Protocol π^{in} takes care of sharing inputs, providing commitments and answering complaints. The program $\mathcal{C}'$ is merely a slight adaption of the (target) program $\mathcal{C}$ evaluated by $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$. It additionally reconstructs the inputs, checks commitments, makes complaints, and only then evaluates $\mathcal{C}$.

Lemma 5. *Given an arbitrary program $\mathcal{C}$ and a robustness parameter $\rho < \frac{n}{2}$, protocol π^{in} UC securely implements the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ evaluating $\mathcal{C}$, from a designated party MPC functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, in the presence of a static and active adversary.*

[13] As mentioned in Section 2.2, we assume a field structure with operation $\oplus$ over the input space.

[14] Note that the output of the complaint bit is a regular output. If robustness is not guaranteed, the adversary might interrupt the computation at this point.

Protocol machine π_i^{in} connects over the regular and the party-0-interface belonging to party i to the functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. In turn π_i^{in} offers an I/O-interface to higher level protocols. The protocol machine π_i^{in} then proceeds as follows:

1. On receiving an input on the I/O-interface: Choose x_i^{it} uniformly at random and compute $x_i^{\mathsf{co}} := x_i \oplus x_i^{\mathsf{it}}$. Using the IT hiding commitment scheme compute $[c_i, o_i] = \mathsf{commit}(x_i^{\mathsf{it}})$. Pass input (x_i^{co}, c_i) to the regular interface and (x_i^{it}, o_i) to the party-0-interface of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. Receive a complaint bit e on the regular interface of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$. If $e = 1$, then input x_i to the regular interface of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$.
2. On receiving an output y on the regular interface of $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, output y on the I/O-interface.

Fig. 9. The protocol machine π_i^{in}.

The program $\mathcal{C}'$ is identical to the (target) program $\mathcal{C}$ except for the input operations. Addition, multiplication and output operations are executed unmodified.
In case of an input operation for party i in program $\mathcal{C}$, program $\mathcal{C}'$ takes an input (x_i^{co}, c_i) over the regular interface to party i, and an input (x_i^{it}, o_i) over the party-0-interface to party i. If $x_i^{\mathsf{it}} \neq \mathsf{open}(c_i, o_i)$, it outputs a complaint bit $e = 1$ to all parties, and takes a new input x_i over the regular interface to party i (default to $x_i = \perp$ if no input is provided). Otherwise, it outputs $e = 0$ and computes $x_i := x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}}$.[14]

Fig. 10. The program $\mathcal{C}'$ to be evaluated by $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$.

Proof. We have to show that the protocol π^{in} implements the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ based on the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ running the program $\mathcal{C}'$ described in Figure 10. We do so by providing an appropriate simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ that renders the ideal model $\sigma_{\mathcal{A}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}])$ indistinguishable from the real model $\pi_{\mathcal{H}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$. We treat the CO ($\frac{n}{2} \leq t < n-\rho$) and the IT ($t < \frac{n}{2}$) setting separately. For $t \geq n - \rho$, functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ does not provide any security guarantees, and indistinguishability follows from Lemma 1.

Case 1: The CO setting for $\frac{n}{2} \leq t < n - \rho$. We show that, for any corruption set $\mathcal{A}$ where $\frac{n}{2} \leq t < n - \rho$, there is a simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ which renders the ideal model $\sigma_{\mathcal{A}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}])$ indistinguishable from the real model $\pi_{\mathcal{H}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$ in the CO setting.

In the CO setting, for $\frac{n}{2} \leq t < n-\rho$, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ is correct and private for inputs at its regular interfaces, but gives the adversary control over inputs at its party-0-interfaces (we may consider the emulated party 0 corrupted) and guarantees no robustness or fairness, only agreement on abort.

The simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ is connected to the interfaces of the corrupted parties to the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$. In turn the simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ simulates the regular interfaces of functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ belonging to corrupted parties and the party-0-interfaces of functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ to the distinguisher.

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

1. When an honest party i makes input to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$:
   (a) Choose $\tilde{x}_i^{\mathsf{it}}$ at random and compute an IT hiding commitment $(c_i, \tilde{o}_i) = \mathsf{commit}(\tilde{x}_i^{\mathsf{it}})$.
   (b) Give $(\tilde{x}_i^{\mathsf{it}}, \tilde{o}_i)$ as output to the distinguisher over the party-0-interface.
   (c) Receive (x_i^{it}, o_i) from the distinguisher over the party-0-interface.
   (d) If $x_i^{\mathsf{it}} \neq \mathsf{open}(c_i, o_i)$, set the complaint bit $e = 1$, otherwise set $e = 0$. Output the complaint bit e to the distinguisher.
   (e) Receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ and halt.
   (f) Simulate the output of the complaint bit e to all parties.
2. When the distinguisher makes input via the regular interface of a corrupted party j:
   (a) Receive (x_j^{it}, o_j) from the distinguisher over the party-0-interface.
   (b) Receive (x_j^{co}, c_j) from the distinguisher over the regular interface of party j.

(c) If $x_j^{\mathsf{it}} \neq \mathsf{open}(c_j, o_j)$, set the complaint bit $e = 1$, otherwise set $e = 0$. Output the complaint bit e to the distinguisher.
(d) Receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ and halt.
(e) Simulate the output of the complaint bit e to all parties.
(f) **If** $e = 1$ take a new input x_j on the regular interface of party j, default to $x_j = \perp$ if no input is provided.
Otherwise, compute $x_j := x_j^{\mathsf{co}} \oplus x_j^{\mathsf{it}}$.
(g) Forward input x_j to functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.

3. When functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ makes output:
(a) Forward the output y of $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ to the distinguisher.
(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 $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$, and in case $o = 0$ halt.
(d) Simulate the output of the value y to all parties.

We now argue that the simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ indeed renders the ideal model $\sigma_{\mathcal{A}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}])$ indistinguishable from the real model $\pi_{\mathcal{H}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$.

When input is made by some honest party i, protocol machine π_i^{in} in $\pi_{\mathcal{H}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$ first splits its input into $x_i = x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}}$ (where x_i^{it} is uniformly random) and computes the IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{it}})$. Then, π_i^{in} provides (x_i^{it}, o_i) as input to functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ at the party-0-interface which is controlled by the adversary. $\sigma_{\mathcal{A}}^{\mathsf{in}}$ simulates this indistinguishably by providing a random value with appropriate opening information $(\tilde{x}_i^{\mathsf{it}}, \tilde{o}_i)$ to the distinguisher over the (simulated) party-0-interface.

Furthermore, protocol machine π_i^{in} provides (x_i^{co}, c_i) as input to $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ via the regular interface. Functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ then issues a boolean complaint bit e, indicating whether the opening failed. The complaint bit e is first handed to the adversary and upon receipt of an output flag $o = 1$ to the remaining parties (or halts on $o = 0$). If the opening failed, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ allows party i to answer the complaint with a new x_i. Otherwise, it computes $x_i = x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}}$. $\sigma_{\mathcal{A}}^{\mathsf{in}}$ simulates this behavior identically to the distinguisher for input by both corrupted and honest parties. Finally, the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ processes x_i as input. In case of input by a corrupted party, the simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ simulates this step indistinguishably by inputting the x_i to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.

This simulation is indistinguishable as long as the adversary does not manage to open a commitment c_i to a value other than x_i^{it}. Since we are in the CO setting, security needs to be guaranteed only for computationally bounded adversaries that, by assumption, cannot break the CO binding property of the commitment. The commitments to the x_i^{it} and the complaint procedure guarantee that the computation is carried out with correct values x_i^{it}. That is, the input shares x_i^{co} and x_i^{it} have the relation $x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}} = x_i$. Otherwise, if the adversary controls the party-0-interfaces (as is the case here), he could manipulate the values x_i^{it} leading to a computation with wrong inputs x_i and hence to an incorrect result.

When output is made, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ delivers the output y to the adversary and awaits an output flag deciding output delivery to honest parties. Outputs are simply forwarded by π_i^{in}. Functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ behaves identically and as such the simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ only needs to forward the messages in question.

Hence the protocol π^{in} CO securely implements the functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ for $\frac{n}{2} \leq t \leq n - \rho$.

Case 2: The IT setting for $t < \frac{n}{2}$. We show that, for any corruption set $\mathcal{A}$ where $t < \frac{n}{2}$, there is a simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ which renders the ideal model $\sigma_{\mathcal{A}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}])$ indistinguishable from the real model $\pi_{\mathcal{H}}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$ in the IT setting.

In the IT setting for $t < \frac{n}{2}$, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ is fair, correct and private for inputs at its party-0-interfaces (we may consider the emulated party 0 honest) but forwards inputs at its regular interfaces to the adversary. For $t \leq \rho$, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ is additionally robust.

The simulator $\sigma_\mathcal{A}^{\mathsf{in}}$ is connected to the interfaces of the corrupted parties to the ideal functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$. In turn the simulator $\sigma_\mathcal{A}^{\mathsf{in}}$ simulates the regular and party-0-interfaces of functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ belonging to corrupted parties to the distinguisher.

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

1. When an honest party i makes input to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$:
   (a) Choose x_i^{co} and x_i^{it} at random and compute an IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{it}})$.
   (b) Give (x_i^{co}, c_i) as output to the distinguisher.
   (c) If $\rho < t < \frac{n}{2}$, receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ and halt.
   (d) Simulate the output of a complaint bit $e = 0$ to all parties.
2. When the distinguisher makes input via the regular interface of a corrupted party j:
   (a) Receive input (x_j^{it}, o_j) and (x_j^{co}, c_j) from the distinguisher over the regular and party-0-interfaces of party j, respectively.
   (b) If $\rho < t < \frac{n}{2}$, receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. In case $o = 0$, forward o to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ and halt.
   (c) **If** $x_j^{\mathsf{it}} \neq \mathsf{open}(c_j, o_j)$, simulate the output of a complaint bit $e = 1$ to all parties via the regular interfaces. Take a new input x_j on the regular interface of party j, default to $x_j = \bot$ if no input is provided.
   Otherwise, simulate the output of $e = 0$ and compute $x_j := x_j^{\mathsf{co}} \oplus x_j^{\mathsf{it}}$.
   (d) Forward the input x_j to functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.
3. When functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ makes output
   (a) If $\rho < t < \frac{n}{2}$, receive an output flag o from the distinguisher, default to $o = 1$ if none is provided. Forward o to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ and, in case $o = 0$, halt.
   (b) Forward the output y from $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ to the distinguisher via the regular interfaces of the corrupted parties $i \in \mathcal{A}$.

We now argue that the simulator $\sigma_\mathcal{A}^{\mathsf{in}}$ indeed renders the ideal model $\sigma_\mathcal{A}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}])$ indistinguishable from the real model $\pi_\mathcal{H}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$.

When input is made by an honest party i, protocol machine π_i^{in} in $\pi_\mathcal{H}^{\mathsf{in}}(\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}'])$ first splits its input into $x_i = x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}}$ (where x_i^{it} is uniformly random) and computes the IT hiding commitment $(c_i, o_i) = \mathsf{commit}(x_i^{\mathsf{it}})$. Then, π_i^{in} provides (x_i^{co}, c_i) and (x_i^{it}, o_i) as input via the regular and the party-0-interface belonging to party i to $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$, respectively. In the current context, for $t < \frac{n}{2}$ in the IT setting, $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ forwards (x_i^{co}, c_i) to the adversary. $\sigma_\mathcal{A}^{\mathsf{in}}$ simulates this indistinguishably by providing random values (x_i^{co}, c_i) to the distinguisher. Here it is important to note that c_i is an IT hiding commitment, and as such independent of x_i^{it}.

Then, $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ computes a boolean complaint bit e, indicating whether the opening failed, and requests an output flag o from the distinguisher, defaulting to $o = 1$ if none is provided. In case $o = 0$, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ halts. In case $o = 1$, $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ outputs e to all parties. If $e = 1$, $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ allows party i to answer the complaint with a new x_i. Otherwise, for $e = 0$, it computes $x_i = x_i^{\mathsf{co}} \oplus x_i^{\mathsf{it}}$. Note that for $t < \frac{n}{2}$ no honest party will ever receive a complaint when trying to give input. $\sigma_\mathcal{A}^{\mathsf{in}}$ simulates this behavior identically to the distinguisher for input by both corrupted and honest parties. Finally, the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ processes x_i as input. In case of input by a corrupted party, the simulator $\sigma_\mathcal{A}^{\mathsf{in}}$ simulates this step indistinguishably by inputting the x_i to $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$.

When output is made, for $\rho < t < \frac{n}{2}$, the ideal functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ first requests an output flag o from the distinguisher, defaulting to $o = 1$ if none is provided. In case $o = 0$, functionality $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ halts. Otherwise $\mathcal{E}_\rho^{\mathsf{des,em}}[\mathcal{C}']$ delivers the output y to all parties. Outputs are simply forwarded by π_i^{in}. Functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ behaves identically, so the simulator $\sigma_{\mathcal{A}}^{\mathsf{in}}$ only needs to forward the messages in question.

Hence the protocol π^{in} IT securely implements the functionality $\mathcal{E}_\rho^{\mathsf{hyb}}[\mathcal{C}]$ for $t < \frac{n}{2}$.

7 Protocols Without Broadcast Channel

In this section, we discuss a protocol for hybrid secure MPC in a setting *without* BC channel. To be able to use our protocol π^ρ (which relies on a BC channel), we implement a BC channel from a complete network of synchronous secure channels. For this purpose, we use the construction from [FHHW03] that provides an IT secure BC channel $\mathsf{bc}_{\mathsf{extCons}}$ with extended consistency and validity detection. 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 $t_v < t \leq t_c$. The construction of $\mathsf{bc}_{\mathsf{extCons}}$ is based on 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 $t \leq t_c$).

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 choice of parameters 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 choice of parameters yields a protocol that extends the existing results and actually matches the result in Theorem 1 for $\rho = 0$ in the case where a BC channel is provided. This construction, where protocol π^ρ is run with $\rho = 0$ on the BC implementation above, is denote by π^0.

Theorem 2. *Given an arbitrary program $\mathcal{C}$, protocol π^0 UC securely implements the ideal functionality $\mathcal{E}_0^{\mathsf{hyb}}[\mathcal{C}]$ evaluating $\mathcal{C}$, from a complete and synchronous network of secure channels (*without *BC channel), and a common reference string, in the presence of a static and active adversary. Let t be the number of corrupted parties. Then π^0 evaluates $\mathcal{C}$ with IT fair security for $t < \frac{n}{2}$, and with CO abort security, for $t < n$ corrupted parties.*

Proof. As described above, protocol π^0 denotes protocol π^ρ with $\rho = 0$, run on the BC implementation of [FHHW03] with parameters $t_v = \rho = 0$ and $t_c = n$. The BC implementation of [FHHW03] is based on a detectable IT secure precomputation. If the precomputation fails, this is jointly detected by all honest parties, and the protocol execution is aborted. If the precomputation succeeds, [FHHW03] shows that the resulting BC protocol is IT secure and robust. Hence, [FHHW03] preserves all security properties except for robustness. Yet, for $\rho = 0$, protocol π^ρ has no robustness guarantee. Therefore, protocol π^0 is IT fair secure for $t < \frac{n}{2}$, and CO abort secure for $t < n$ corrupted parties.

8 Conclusions

We describe a hybrid secure MPC protocol 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 protocol 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,Kil00,Kat07,IKLP06]. We provide a security proof of the protocol in the UC setting based on synchronous secure channels, a broadcast channel, and 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 a BC is provided, indicating that a BC channel is required only for robustness.

9 Acknowledgements

We would like to thank Björn Tackmann and Stefano Tessaro for their valuable feedback and interesting discussions.

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.

[Blu81] Manuel Blum. Coin flipping by telephone. In *CRYPTO*, pages 11–15, 1981.

[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.

[Bra85] Gabriel Bracha. An $O(\lg n)$ expected rounds randomized byzantine generals protocol. In *STOC'85*, pages 316–326. ACM, 1985.

[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.

[DIK+08] Ivan Damgård, Yuval Ishai, Mikkel Krøigaard, Jesper Buus Nielsen, and Adam Smith. Scalable multiparty computation with nearly optimal work and resilience. In *CRYPTO'08*, pages 241–261. Springer, 2008.

[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.

[KMR09] 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.

[LRM10] Christoph Lucas, Dominik Raub, and Ueli Maurer. Hybrid-secure MPC: Trading information-theoretic robustness for computational privacy. In *PODC '10*, pages 219–228. ACM, 2010.

[Luc08] Christoph Lucas. Combining computational and information-theoretic security in multi-party computation. Master's thesis, ETH Zürich, 2008. Supervised by Ueli Maurer and Dominik Raub, see http://e-collection.ethbib.ethz.ch/view/eth:31131.

[PSL80] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[Rau10] Dominik Raub. *Exploring the Limits of Multi-Party Computation*. PhD thesis, ETH Zürich, 2010.

[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 Perfectly Hiding or Perfectly Binding UC Commitments

We describe a UC secure one-to-many commitment schemes implementing the one-to-many commitment functionality $\mathsf{F}_{\mathsf{Com,1:M}}$ 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].[15]

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

Functionality $\mathsf{F}_{\mathsf{ComH,1:M}}$ 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 $\mathsf{committed}$ 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 $\mathsf{F}_{\mathsf{ComB,1:M}}$ 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 $\mathsf{committed}$ 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 $\mathsf{F}_{\mathsf{ComH,1:M}}$ and $\mathsf{F}_{\mathsf{ComB,1:M}}$.

A.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].

[15] Note that, in this section, we describe the protocol as an n-party protocol. This variable n is not related to the variable n in the rest of the paper. It is only for the sake of simplicity that we use the same symbol. In fact, we use the protocol described here in the $(n+1)$-party protocol $\pi^{\mathsf{des},\rho}$, where n is the number of parties running protocol π^{ρ}.

A.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 $\mathsf{crs}^n = (N, K_X, K_E, \bar{K}_1, \ldots, \bar{K}_n, \mathsf{crs}^n{}')$. The first part of the crs^n, encompassing the $n+3$ keys $N, K_X, K_E, \bar{K}_1, \ldots, \bar{K}_n$, 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 $K_E, \bar{K}_1, \ldots, \bar{K}_n \in \mathcal{K}_E$ are random E-keys. The second part of the crs^n, i.e. $\mathsf{crs}^n{}'$, 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.

A.3 The UC Commitment Protocol

Without loss of generality, let $C = P_1$ be the committer and the remaining parties be the receivers $R_i = P_i$ $(i \in 2, \ldots, n)$. 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 $(c_i', o_i') = \mathsf{commit}'(s_i, \mathsf{crs}^n{}')$, 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 $\mathsf{committed}$

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.

A.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 σ_R^{it} 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, σ_R^{it} produces a regular crs^n with E-key K_E and E-trapdoor t_E. During the commit phase, σ_R^{it} 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, σ_R^{it} receives the correct $m^\star$ from $\mathsf{F}^h_{\mathsf{com}}$. Now, σ_R^{it} opens the K_E commitment c_3 to $m_3' = m^\star \oplus m_2$ using t_E.

Finally, simulator σ_C^{co} is used in case 2 where C and any number of receivers are CO corrupted. First, σ_C^{co} produces a fake $\widetilde{\mathsf{crs}^n}$ with interchanged keys: On one hand, in $\widetilde{\mathsf{crs}^n}$, K_X is an *equivocable* key taken from $\mathcal{K}_E$, together with trapdoor t_{K_X} for equivocability. On the other hand, in $\widetilde{\mathsf{crs}^n}$, 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 $\widetilde{\mathsf{crs}^n}$ is indistinguishable from a real crs^n. Furthermore, σ_C^{co} internally runs the protocol of honest R_i which can be perfectly simulated since they do not require any input. During the simulation, σ_C^{co} 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, σ_C^{co} 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, σ_C^{co} 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 σ_C^{it}. Our simulator σ_C^{it} first produces a regular crs^n. Then, σ_C^{it} 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, σ_C^{it} 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 σ_C^{it} is used in case 1 where C and any number of receivers are IT or CO corrupted. First, σ_C^{it} produces a regular crs^n with X-key K_X and X-trapdoor t_N. Furthermore, σ_C^{it} internally runs the protocol of honest R_i which can be perfectly simulated since they do not require any input. During the simulation, σ_C^{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, σ_C^{it} extracts the message m from c_3 using the trapdoor t_N, and enters it into $\mathsf{F}^b_{\mathsf{com}}$. In the opening phase, σ_C^{it} sends an open message to $\mathsf{F}^b_{\mathsf{com}}$ if and only if C provides correct opening information for m.

Simulator σ_R^{co} is used in case 2 where C is honest, but any number of receivers R_i is CO corrupted. First, σ_R^{co} produces a fake $\widetilde{\mathsf{crs}^n}$ with interchanged keys: On one hand, in $\widetilde{\mathsf{crs}^n}$, K_X is an *equivocable* key taken from $\mathcal{K}_E$, together with trapdoor t_{K_X} for equivocability. On the other hand, in $\widetilde{\mathsf{crs}^n}$, 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 $\widetilde{\mathsf{crs}^n}$ is indistinguishable from a real crs^n. In the commit phase in step C.1, σ_R^{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, σ_R^{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, σ_R^{co} receives the correct m from $\mathsf{F}_{\mathsf{com}}^h$. Then, σ_R^{co} opens the commitment c_3 and the commitment c_2 to $m_3' = m_2' = m$. By choice of the $\widetilde{\mathsf{crs}^n}$, 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, σ_R^{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 σ_R^{it} that exploits this. First, σ_R^{it} produces a regular crs^n. 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.