A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer

Dennis Hofheinz and Jörn Müller-Quade

IAKS, Arbeitsgruppe Systemsicherheit,
Prof. Dr. Th. Beth,
Fakultät für Informatik, Universität Karlsruhe,
Am Fasanengarten 5, 76 131 Karlsruhe, Germany

Abstract. This work develops a composable notion of security in a synchronous communication network to analyze cryptographic primitives and protocols in a reliable network with guaranteed delivery. In such a synchronous model the abort of protocols must be handled explicitly. It is shown that a version of *global bit commitment* which allows to identify parties that did not give proper input cannot be securely realized with the primitives *oblivious transfer* and *broadcast*. This proves that the primitives oblivious transfer and broadcast are not complete in our synchronous model of security.
In the synchronous model presented ideal functionalities as well as parties can be equipped with a "shell" which can delay communication until the adversary allows delivery or the number of rounds since the shell received the message exceeds a specified threshold. This additionally allows asynchronous specification of ideal functionalities and allows to model a network where messages are not necessarily delivered in the right order. If these latency times are chosen to be infinite the network is no more reliable and becomes completely asynchronous. It is shown that secure protocols in the setting of [Can01] or [CLOS02] can be transformed to secure realizations in the new model if latency times are chosen to be infinite.

1 Introduction

In this contribution it is proven that in a communication network in which message delivery is guaranteed and participating parties are periodically activated, oblivious transfer together with a broadcast primitive are not complete for secure multi-party computations.

To show this separation between security in reliable networks and security in completely asynchronous networks a new synchronous model is developed. In addition to the properties of the synchronous models of [Can00] the new model allows very general composition of protocols along the line of the asynchronous settings [Can01,PW01]. The new model is a synchronous variation of [Can01] (for a relation, cf. Section 2.5). It differs from the synchronous variant sketched in [Can01], which was not suitable for our purpose as it does not guarantee activation of ideal functionalities in each round.

It might have been possible to formulate our main result in the frameworks of [PW01,PW00]. Yet since machine modeling and scheduling there differs substantially from that in [Can01,CLOS02], it would be difficult to compare our result to established realizability results in the latter settings: Our goal is to point out the importance of reliability assumptions on a network for deducing hierarchies of primitives, and to relate our result to results derived for the asynchronous modelings of [Can01,CLOS02].

Our new model shares with the aforementioned notions of security the concept of *simulatability*. Intuitively, this means that a given protocol is compared to an idealization of the protocol task in question and considered secure if no difference can be detected by any protocol environment, or, an arbitrary user. There is already a (positive and constructive) general realizability result for protocol tasks in a synchronous variant of the setting [Can01], cf. [Can01, Theorem 9 of full version]. Another realizability result was established in [CLOS02], again for a slight (asynchronous)

variation of the setting of [Can01]. Specifically, [CLOS02] present a protocol construction with which general reactive ideal functionalities (i. e., idealizations of protocol tasks) can be securely realized, given only a *common reference string*. (A common reference string, ideally drawn from a fixed distribution, can be considered an idealization of a public set-up information.)

However, the setting of [Can01] (and the mentioned variations) does not allow to formulate "timeouts" or functionalities which guarantee certain response times. In consequence, even secure protocols in that sense may "get stuck" or "hang" in face of corrupted parties, *even* if all protocol messages of the uncorrupted parties get delivered immediately.[1] Thus, we believe it is reasonable to investigate—in a simulatability-based setting—security properties of functionalities which *do* guarantee service. In Section 2, we therefore present a *synchronous* modeling of multi-party computation, along with a very general composition theorem and a result allowing to carry over realizability results established in the settings of [Can01,CLOS02] into our setting.[2] In the new model tools are provided to catch reliable or even asynchronous networks in our setting. In particular, we show that a protocol realizing a certain ideal functionality in the settings [Can01,CLOS02] realizes a similar functionality in our setting, *yet one in which it is made explicit that no response can be guaranteed*.

However, properties like guaranteed output and explicit abort can be especially important for real world applications—e. g., an electronic election or an electronic auction should not "hang", but should be robust to attacks like the one presented here.

If output is to be guaranteed, then aborting protocols must be handled explicitly. A synchronous, i. e., a completely reliable network allows to distinguish different kinds of abort—the most interesting of which is the abort with cheater identification. A commitment of one party to all parties to the same bit (a *global bit commitment*) becomes more challenging in a synchronous network as the ideal functionality aborts only if the committer refused to commit. Hence this protocol allows for cheater identification. To make this strong contrast to the asynchronous setting explicit we prove that it is not possible to securely implement a global bit commitment in our synchronous model given the cryptographic primitives of oblivious transfer and broadcast.

2 The modeling

2.1 Real and Hybrid Model

The real model is an abstraction of a *malicious* protocol environment as one would expect it in reality. Thus, a real-model adversary may read all messages sent between parties, or corrupt parties and then control their behavior. The hybrid model is a real model in which parties are additionally offered blackbox access to idealizations of (sub)protocols, henceforth called *ideal functionalities*.

All parties, adversaries and ideal functionalities are modelled as *interactive Turing machines (ITMs)* (cf. [Can01]). An ITM has read-only tapes for incoming communication and local input, write-once tapes for outgoing communication and local output, a work tape, a one-bit activation tape, a read-only random tape and read-only tapes containing machine identity and security parameter, respectively. Unless explicitly noted, any ITM mentioned in this work is assumed to be *polynomially bounded* in the sense that no matter with which tape contents activated, it terminates

[1] For the framework of [PW01], this problem was addressed in [BPSW02].

[2] Here we would like to stress that for the proofs of our Lemma 3, Theorems 4,5 and Proposition 7, we adopted ideas already appearing in the works [PW00,Can01,PW01,Bac03]; yet of course, we had to adapt these ideas to our specific setting.

this activation within $p(k)$ steps (i. e., transitions) for a fixed, ITM-specific polynomial p and the value k on the security parameter tape. To reflect polynomial total length of a protocol run, the ITMs $\mathcal{Z}$ and $\mathcal{A}$ described below are assumed to *halt* after a polynomial number of activations. An ITM which has halted terminates instantly—without switching at all—on all future activations.

Aside from parties P_i and an adversary $\mathcal{A}$, an *environment machine* $\mathcal{Z}$ (modelled as an ITM[3]) takes part in a protocol run. $\mathcal{Z}$ represents an arbitrary procotol environment in which the investigated protocol is run as a subprotocol. In particular, $\mathcal{Z}$ supplies parties with input, reads their output and may even communicate with the adversary. In the simulatability-based definition of security given below, $\mathcal{Z}$ takes a crucial role.

To protect the polynomially bounded adversary from being activated "too often" by the environment, we introduce the following special capability of the adversary: $\mathcal{A}$ may enter a special class of states to signal that further messages from $\mathcal{Z}$ are not to be delivered to $\mathcal{A}$ and thus, such messages do not cause activation of $\mathcal{A}$. When $\mathcal{A}$ enters such a state, we say that $\mathcal{A}$ *blocks*. This convention resembles the mechanism of *length functions* used in [BPSW02] for similar purposes. Without such a convention and polynomially bounded adversaries which may not depend on the distinguishing environment, the environment may simply "kill" the adversary by activating it sufficiently often right at the start of the protocol. This is especially crucial for simulators (see below).

The real model can be seen as a (trivial) special case of the hybrid model, and hence it suffices to give a description of a protocol run in the $\{\mathcal{F}_i\}$-hybrid model for a *finite* set $\{\mathcal{F}_i\}$ of ideal functionalities. But first, a notation: *Delivering* a message means *moving* it from the outgoing communication tape of the sending ITM to the incoming communication tape of the receiving ITM. Here we assume *authentication*: the sender identity is automatically added to the message at delivery. To prevent blocking incoming communication by sending huge or many messages "standing in the way", messages are delivered *interleaved by sender*[4]. After an ITM terminates its activation, its incoming communication tape is automatically cleared to ensure future message processing. All ITMs are assumed to initially have empty communication, local in-/output and work tapes. Each ITM is presumed to have a unique identity, an infinite sequence of independently uniformly chosen bits on its random tape, and a (common) security parameter k on the security parameter tape. All activation tapes contain 0, so no ITM is active.

All ITMs may, when active, of course access their own tapes; furthermore, $\mathcal{Z}$ may read the local output tapes of the P_i and write onto their local input tapes in a write-only manner. $\mathcal{A}$ may read all outgoing communication tapes of the P_i and may also *corrupt* one or more parties. Upon corruption of P_i, $\mathcal{A}$ instantly gets a message containing P_i's complete past history (including states, head positions and tapes). $\mathcal{A}$ may from then on write arbitrary messages on its outgoing communication tape in the name of P_i, and all messages addressed to P_i are delivered to $\mathcal{A}$. Moreover, a message stating that P_i was corrupted is automatically delivered both to $\mathcal{Z}$ and to all $\mathcal{F}_i$. Very briefly, the message transfer rules are: $\mathcal{Z}$ may talk to $\mathcal{A}$, $\mathcal{A}$ may talk to $\mathcal{Z}$, to the parties and the $\mathcal{F}_i$, the $\mathcal{F}_i$ may talk to $\mathcal{A}$ and to the parties, and the parties may talk to each other and to $\mathcal{A}$. A detailed description of a protocol run in the $\{\mathcal{F}_i\}$-hybrid model follows.

[3] In [Can01], $\mathcal{Z}$ is the only *non-uniform* ITM, i. e., $\mathcal{Z}$ gets as initial input the value of an arbitrary function of the security parameter. We adopt this, but stress that all results and proofs below hold also for *uniform* $\mathcal{Z}$, cf. also the discussion in [HMQS03].

[4] That means interleaved in blocks of constant length larger than the size needed to identify the sender of the respective message.

1. **Attack Phase:** Basically, this is a message-driven interaction between $\mathcal{Z}$, $\mathcal{A}$ and the $\mathcal{F}_i$, only $\mathcal{Z}$ and the $\mathcal{F}_i$ may not interact directly. First, $\mathcal{Z}$ is activated with local input "`round-start`". After $\mathcal{Z}$ has terminated its activation, all messages $\mathcal{Z}$ possibly wrote to $\mathcal{A}$ are delivered or, if $\mathcal{A}$ *blocked* messages from $\mathcal{Z}$, simply erased. If there was no such message, or if $\mathcal{Z}$ has halted, we consider the complete protocol run ended and the first cell on $\mathcal{Z}$'s local output tape is interpreted as $\mathcal{Z}$'s (binary) output. Otherwise, $\mathcal{A}$ is activated next or, if $\mathcal{A}$ blocked, $\mathcal{Z}$ is activated again. Once $\mathcal{A}$ has terminated its activation and wrote at least one message to $\mathcal{Z}$, all such messages are delivered and $\mathcal{Z}$ is activated again. However, if $\mathcal{A}$ wrote *no* message to $\mathcal{Z}$, the first $\mathcal{F}_i$ (in order of ITM identities) to which $\mathcal{A}$ wrote at least one message is activated with all messages addressed to it from $\mathcal{A}$ delivered. This includes messages sent from $\mathcal{A}$ to $\mathcal{F}_i$ in the name of a corrupted party. Once this $\mathcal{F}_i$ terminates its activation, all messages it possibly wrote to $\mathcal{A}$ or the parties are delivered and $\mathcal{A}$ gets activated again. If $\mathcal{A}$ wrote messages neither to $\mathcal{Z}$ nor to an $\mathcal{F}_i$, all messages $\mathcal{A}$ wrote to the P_i are delivered and we proceed to the next phase.
2. **Party Computation:** All messages from any party P_i to another party P_j are delivered. Then, all non-corrupted P_i are activated in parallel. When all P_i have terminated their activations, messages they have written to the $\mathcal{F}_i$ or to $\mathcal{A}$ are delivered.
3. **Ideal Functionality Computation:** All $\mathcal{F}_i$ are activated in parallel with local input "`computation`". After the $\mathcal{F}_i$ have terminated their activations, messages the $\mathcal{F}_i$ have written to $\mathcal{A}$ or to the parties are delivered. Then, we start over with the attack phase.

Note that $\mathcal{A}$ cannot access communication of uncorrupted parties with ideal functionalities. However, our scheduling models a "rushing" adversary that may let corrupted parties send messages in dependance of the messages sent by honest parties *in the same round*. Since all ITMs terminate their current activation in polynomial time and both $\mathcal{Z}$ and $\mathcal{A}$ halt after a polynomial number of activations, a protocol run as described above ends after a polynomial number of steps. When we speak of a *protocol*, we mean a set of parties P_i running together as above. The output distribution of $\mathcal{Z}$ when run on security parameter k in the $\{\mathcal{F}_i\}$-hybrid model with protocol π and an adversary $\mathcal{A}$ is denoted by $\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k)$. Now we are ready to state the first part of our security definition, which relates two protocols.

Definition 1. *Let π be an n-party protocol formulated in the $\{\mathcal{F}_i\}$-hybrid model and let τ be an n-party protocol formulated in the $\{\mathcal{G}_j\}$-hybrid model. We say that* π securely realizes τ *(written $\pi \geq \tau$) iff for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ (called simulator) such that for every environment $\mathcal{Z}$ the function*

$$\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k) = 1) - \mathbf{P}(\mathcal{Z}(\{\mathcal{G}_j\}, \tau, \mathcal{S}, k) = 1)$$

is negligible[5] *in k. If this holds even with respect to $\mathcal{Z}$ which are* not *necessarily polynomially bounded (but still halt after polynomially many activations), we say that π securely realizes τ* unconditionally *(written $\pi \gg\geq \tau$).*

Note that for the unconditional case, we have chosen to allow an unbounded *environment*, but *not* an unbounded adversary. When considering unbounded adversaries, there is a practical need for an unbounded environment, as known proof techniques for composition don't seem to apply when only the adversary, but not the environment is unbounded. On the other hand, when

[5] $f : \mathbb{N} \to \mathbb{R}$ is called *negligible*, iff $\forall c \in \mathbb{N}\ \exists k_0 \in \mathbb{N}\ \forall k > k_0 : |f(k)| < k^{-c}$.

considering an unbounded environment, the security notion which allows for unbounded adversaries is *strictly weaker* than the same notion with polynomially bounded adversaries. Also, when allowing both unbounded adversaries and environments, the resulting security notion does *not* imply the seemingly weaker bounded security notion. (Our notion "$\geqq$", though, *does* imply "$\geq$" trivially.)

2.2 Ideal Model

In contrast to the real model, the ideal model reflects an idealization of a given protocol task. For simulation-based approaches, such an idealization is generally modelled as a single ideal functionality $\mathcal{F}$ which reads all input and secretly computes output accordingly, possibly in a reactive manner. This can be modelled in the $\{\mathcal{F}\}$-hybrid model with a set $D(\mathcal{F})$ of n identical *dummy parties*. (Here, the number n of parties is implicitly determined by the specification of $\mathcal{F}$.) Each dummy party relays its local input to $\mathcal{F}$ and locally outputs whatever it receives from $\mathcal{F}$. For polynomially boundedness, we assume that each dummy party terminates its activation after it has copied as much input to $\mathcal{F}$ as $\mathcal{F}$ can read in one activation (analogously for the output $\mathcal{F}$ may have written). Now we are ready to define the *$\mathcal{F}$-ideal model* as the $\{\mathcal{F}\}$-hybrid model with dummy parties $D(\mathcal{F})$ *and* an additional party computation step after the functionality computation step. This is to reflect immediate output generation and to pave the way for the composition theorems presented below. The output of an environment $\mathcal{Z}$ run on security parameter k in the $\mathcal{F}$-ideal model (as described above) and an adversary $\mathcal{A}$ will be denoted $\mathcal{Z}(\mathcal{F}, \mathcal{A}, k)$. The second part of our security definition allows us to specify when we consider a protocol a secure implementation of an ideal functionality.

Definition 2. *Let π be an n-party protocol formulated in the $\{\mathcal{F}_i\}$-hybrid model and let $\mathcal{F}$ be an n-party ideal functionality. We say that π* securely realizes *$\mathcal{F}$ (written $\pi \geq \mathcal{F}$) iff for every adversary $\mathcal{A}$ there exists an adversary $\mathcal{S}$ such that for any $\mathcal{Z}$, the function*

$$\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}, k) = 1) - \mathbf{P}(\mathcal{Z}(\mathcal{F}, \mathcal{S}, k) = 1)$$

is negligible in k. If this holds even with respect to $\mathcal{Z}$ which are not *necessarily polynomially bounded (but still halt after polynomially many activations), we say that π securely realizes $\mathcal{F}$* unconditionally *(written $\pi \gtrdot \mathcal{F}$).*

From the definitions, it is clear that the relations "$\geq$" and "$\gtrdot$" are transitive relations on n-party protocols. Furthermore, $\pi \geq \tau$ in conjunction with $\tau \geq \mathcal{F}$ implies $\pi \geq \mathcal{F}$, analogously for "$\gtrdot$". The following definition and the next lemma will be helpful for later proofs. We call a set $\mathfrak{A}$ of adversaries *complete* iff modified Definitions 1 and 2, in which only over all $\mathcal{A} \in \mathfrak{A}$ is quantified, are already equivalent to their respective non-modified counterparts (both for "$\geq$" and "$\gtrdot$"). Since adversaries are assumed to halt after a polynomial number of activations, there can be no *single* adversary which is complete. However, consider the set $\mathfrak{A} := \{\mathcal{A}_p : p \in \mathbb{Z}[x]\}$. Here, $\mathcal{A}_p$ is a fixed *dummy* adversary[6] which runs the following simple program: On each activation (with exceptions denoted below), it writes its complete view as a message to $\mathcal{Z}$ and terminates its activation. This view consists of all touched cells of $\mathcal{A}_p$'s own incoming communication tape and the outgoing communication tapes of uncorrupted parties (all messages are left interleaved by sender). Additionally, $\mathcal{A}_p$ accepts these *commands* (given as messages) from $\mathcal{Z}$: "`corrupt` P_i" makes $\mathcal{A}_p$

[6] A similar dummy adversary is also used in [Can01]; in fact, the proof of Lemma 3 is an adaption of the proof in [Can01].

corrupt P_i prior to writing the view. "`write` m" causes $\mathcal{A}$ to write m on its outgoing communication tape, and "`write` m `as` P_i" makes $\mathcal{A}$ write m with sender P_i *iff* P_i is corrupted. If in the first case, the recipient of the message is an $\mathcal{F}_i$, no view is written, so the message is delivered to that $\mathcal{F}_i$ immediately after $\mathcal{A}_p$ terminates its current activation. "`next round`" causes $\mathcal{A}_p$ to terminate its activation without writing *any* message, thus initiating the computation phases. Lastly, $\mathcal{A}_p$ runs internal counters so that it halts after $\max\{p(k), 1\}$ activations and makes in each activation at most $\max\{p(k), M\}$ steps, where M is the number of steps it takes to terminate immediately. Views to be written to $\mathcal{Z}$ "growing too large" in this sense are simply truncated. With these precautions, $\mathcal{A}_p$ is polynomially bounded and halts after a polynomial number of activations. Note that $\mathcal{A}$ never blocks.

Lemma 3. *The adversary set* $\mathfrak{A} := \{\mathcal{A}_p : p \in \mathbb{Z}[x]\}$ *just described is complete.*

Proof. This is proven in Appendix A. □

2.3 Composition

In the style of [Unr02], two composition theorems are presented. The first ensures that a (sub)protocol π can substitute an ideal functionality $\mathcal{F}$ once $\pi \geq \mathcal{F}$. The second shows that $\pi \geq \mathcal{F}$ implies $\pi \| p \geq \mathcal{F} \| p$ (similarly for "$\gtrsim$"), where $\pi \| p$ and $\mathcal{F} \| p$ are the respective multi-session extensions of π and $\mathcal{F}$ (cf. [CR03] for a similar result in the [Can01]-setting).

First, we introduce a notational tool capturing what it means to "substitute" an ideal functionality by a subprotocol. Let π be an n-party procotol formulated in the $\{\mathcal{F}_i\}$-hybrid model. Let τ be an n-party protocol formulated in the $\{\mathcal{G}_j\}$-hybrid model, so that $\tau \geq \mathcal{F}$ or $\tau \gtrsim \mathcal{F}$ for an $\mathcal{F} \in \{\mathcal{F}_i\}$ and $\{\mathcal{F}_i\} \cap \{\mathcal{G}_j\} = \emptyset$. Here, the intersection is to be understood with respect to the *identities* of ITMs. Thus, the last requirement can always be enforced by simply renaming ideal functionalities. Then π^τ denotes the protocol which is identical to π, but replaces all messages which the parties write to $\mathcal{F}$ by invocations of τ. That is, party P_i of protocol π^τ first runs the program of the respective P_i^π from π and then—after converting messages P_i^π sent to $\mathcal{F}$ to local input—the program of P_i^τ from τ. Local output of P_i^τ is converted back into $\mathcal{F}$-messages which P_i^π may then read in its next activation. Inter-party communication inside π and τ is separated by prefixing the respective messages with "π", resp. "τ". Hence, π^τ is an n-party protocol in the $\{\mathcal{G}_j\} \cup (\{\mathcal{F}_i\} \setminus \{\mathcal{F}\})$-hybrid model.

Theorem 4. *Let* π *be an* n*-party procotol* π *formulated in the* $\{\mathcal{F}_i\}$*-hybrid model. Let* τ *be an* n*-party protocol formulated in the* $\{\mathcal{G}_j\}$*-hybrid model, such that* $\tau \geq \mathcal{F}$ *(resp.,* $\tau \gtrsim \mathcal{F}$*) for an* $\mathcal{F} \in \{\mathcal{F}_i\}$ *and* $\{\mathcal{F}_i\} \cap \{\mathcal{G}_j\} = \emptyset$*. Then* $\pi^\tau \geq \pi$ *(resp.,* $\pi^\tau \gtrsim \pi$*) for the protocol* π^τ *conducted as above.*

Proof. This is proven in Appendix A. □

Often it is useful to have a (possibly unbounded) number of *instances* of an ideal functionality available. In [Can01], this is modelled directly in the $\mathcal{F}$-hybrid model, where access to different instances of $\mathcal{F}$ is distinguished via a *session identifier (SID)*. Such a SID uniquely identifies one instance of an ideal functionality, and in principle an unbounded number of instances may be accessed. However, the composition theorem given above holds only for a *constant* number of instances, which must be independent of the current security parameter.

To address a need for an unbounded (yet necessarily polynomial) number of instances, again we first fix some notation. For an ideal functionality $\mathcal{F}$ and a polynomial $p \in \mathbb{Z}[x]$, we denote by

$\mathcal{F}\|p$ the *p-parallelization* of $\mathcal{F}$. The functionality $\mathcal{F}\|p$ internally keeps $\max\{p(k),0\}$ simulations $\mathcal{F}^j$ (for $1 \leq j \leq \max\{p(k),0\}$) of $\mathcal{F}$ running and halts when all of them have halted. Every time $\mathcal{F}\|p$ is activated with local "`computation`" input, it activates all $\mathcal{F}^j$ with this input forwarded. Additionally, each time $\mathcal{F}\|p$ receives a message of the form (j,m) for $1 \leq j \leq \max\{p(k),0\}$, it forwards the "inner" message m to $\mathcal{F}^j$; messages not of this form are ignored. Similarly, all messages the $\mathcal{F}^j$ write are forwarded by $\mathcal{F}\|p$ after prepending them with the respective index j. Finally, corruption notifications sent to $\mathcal{F}\|p$ are automatically forwarded to all simulated $\mathcal{F}^j$ (yet without explicitly activating them).

Also for a protocol π, let $\pi\|p$ denote its p-parallelization described in the following. Each party P_i in $\pi\|p$ internally keeps $\max\{p(k),0\}$ simulations P_i^j of the respective party P_i in π and halts when all of them have halted. All P_i^j are activated whenever P_i gets activated. Moreover, local input and messages of the form (j,m) with $1 \leq j \leq \max\{p(k),m\}$ are forwarded to P_i^j after deleting the prefix j. If a simulated P_i^j generates output or writes a message, P_i relays this output or message after prefixing it with j. Consequently, if π is formulated in the $\{\mathcal{F}_i\}$-hybrid model, then $\pi\|p$ is formulated in the $\{\mathcal{F}_i\|p\}$-hybrid model. To achieve polynomially boundedness of both parallelized functionalities and parties, we restrict the relaying of local input and incoming messages to the respective maximum size the simulated functionality, resp. party, is able to read in one activation.

It should be remarked that such parallelization techniques bear a great resemblance to the "joint state functionalities" described in [CR03]. There, a "joint state functionality" $\hat{\mathcal{F}}$ built from an arbitrary ideal functionality $\mathcal{F}$ allows for an a priori *unbounded* number of parallel executions of $\mathcal{F}$; similarly for parallelized protocols $\pi^{[\hat{\rho}]}$ in that sense. (A priori unbounded means here that, e. g., the machine $\hat{\mathcal{F}}$ itself does not restrict the number of invoked $\mathcal{F}$-instances; yet a polynomial restriction follows from the polynomiality of $\mathcal{Z}$ and $\mathcal{A}$.)

Theorem 5. *Let π and τ be n-party protocols formulated in the $\{\mathcal{F}_i\}$, resp. $\{\mathcal{G}_j\}$-hybrid model. Then, for an arbitrary polynomial $p \in \mathbb{Z}[x]$, from $\pi \geq \tau$ (resp., $\pi \gg\!\geq \tau$) it follows $\pi\|p \geq \tau\|p$ (resp., $\pi\|p \gg\!\geq \tau\|p$). Specifically, $\pi \geq \mathcal{F}$ implies $\pi\|p \geq \mathcal{F}\|p$ and $\pi \gg\!\geq \mathcal{F}$ implies $\pi\|p \gg\!\geq \mathcal{F}\|p$.*

Proof. This is proven in Appendix A. □

For practical purposes, the following corollary might be handy. It combines Theorem 4, Theorem 5 and the transitivity of "$\geq$", resp. "$\gg\!\geq$".

Corollary 6. *Let $p \in \mathbb{Z}[x]$. Let π and τ be n-party protocols formulated in the $\{\mathcal{F}_i\}$, resp. $\{\mathcal{G}_j\}$-hybrid model such that $\{\mathcal{F}_i\} \cap \{\mathcal{G}_j\|p\} = \emptyset$. Now if $\tau \geq \mathcal{F}$ and $\mathcal{F}\|p \in \{\mathcal{F}_i\}$, then $\pi^{\tau\|p}$ (using functionalities $\{\mathcal{G}_j\|p\} \cup (\{\mathcal{F}_i \setminus \{\mathcal{F}\|p\}\})$ securely realizes π. If additionally π securely realizes an ideal functionality $\mathcal{I}$, then so does $\pi^{\tau\|p}$. All this also holds unconditionally.*

2.4 Shell Constructs

In contrast to [Can01,CLOS02], our model does not allow the adversary to block messages, not even those sent from or to an ideal functionality. This allows formulating functionalities in a very specific way, but often it might be necessary for simulation to leave delivery—to a certain degree—up to the simulator. Therefore, we adapt the idea of [Bac03] to equip a machine with a "coat", or "shell", which manages message delivery to and from it. (In [Bac03], an "asynchronous coat" was used to investigate synchronously formulated machines in an asynchronous setting.)

Namely, for an ITM M (one should have in mind a party or an ideal functionality here), we define M's *asynchronization* $[M](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ (often denoted $[M]$ when the context is clear), with $p_{\mathsf{recv}}, p_{\mathsf{send}} \in \mathbb{Z}[x] \cup \{\infty\}$ and $\mathsf{clk} \in \{\mathsf{async}, \mathsf{sync}\}$. Internally, $[M]$ keeps a simulation of M and relays local in- and output as well as communication of M with $\mathcal{A}$ directly, with some exceptions explicitly noted below. Upon an incoming message m from a sender $S \neq \mathcal{A}$, $[M]$ writes a message "`request receive` j `from` S" to $\mathcal{A}$; here, j simply denotes a running number assigned by $[M]$. If $[M]$ receives a message "`allow receive` j" from $\mathcal{A}$, where j has been assigned before, $[M]$ relays the corresponding message m to the simulated M. Also, any message is automatically relayed to M after $p_{\mathsf{recv}}(k)$ activations of $[M]$—or, if M is an ideal functionality, after $p_{\mathsf{recv}}(k)$ local "`computation`" inputs. (There is *no* automatic message delivery if $p_{\mathsf{recv}} = \infty$.) Similarly, if M wants to send a message m to a recipient $R \neq \mathcal{A}$, $[M]$ first generates a "`request send` j `to` R" message to $\mathcal{A}$ and actually sends m to R upon an "`allow send` j" message from $\mathcal{A}$ or—whatever happens first—after $p_{\mathsf{send}}(k)$ rounds (i. e., $[M]$-activations, resp. local "`computation`" inputs). M is activated exactly once in *every* $[M]$-activation if $\mathsf{clk} = \mathsf{sync}$. Otherwise, M is activated only if one or more messages or local input are relayed to it in the respective $[M]$-activation; in that case, M is activated once for local input other than "`computation`", and each incoming message. The order is: local input first, then incoming messages ordered by sender identity. (Formally, we assume M only to process *interleaved* messages, as guaranteed by the delivery process in our modeling.)

$[M]$ halts when M has halted and all messages *from* M have actually been sent. Clearly, $[M]$ halts after a polynomial number of activations (resp., rounds in the case of ideal functionalities) if and only if M does so, $\mathsf{clk} = \mathsf{sync}$ and $p_{\mathsf{send}} \neq \infty$. To make $[M]$ polynomially bounded in each activation, we first mandate that $[M]$ reads in each activation only *one* local input and *one* message per sender $S \neq \mathcal{A}$, truncated to the maximum size which M is able to process in one activation. (We assume that by the time of construction of the "shell", the number n and the identities of parties and adversary are already fixed.) Additionally, at most one "`allow receive`" message from $\mathcal{A}$ per sender and at most one "`allow send`" message per recipient is processed; also, at most one message from $\mathcal{A}$ to M is read and truncated if "too long" for M. Processing of $\mathcal{A}$'s messages stops as soon as "too long" or "too many" messages are encountered. Clearly, these restrictions limit the generality of $[M]$, yet in many cases—as, e. g., the case $M = \mathcal{F}_{\mathrm{SFT}}$ of an ideal functionality for secure function evaluation—this might be considered condonable.

By adding shells to the parties of a protocol, one can catch the notion of reliable or even asynchronous networks, the former which deliver messages after a polynomial number of steps. Furthermore, ideal functionalities may be formulated asynchronously in the first place, and later a shell may be added to leave message delivery factually up to the adversary (while it is possible to fix certain maximum latency times for messages sent to and from the functionality).

There is another important use of (different) shell constructs, which we sketch only briefly: One can implicitly restrict the class of adversaries by a suitable shell. Such a shell hands its internal state (including the state of $\mathcal{F}$) and all future inputs to the adversary when being notified of a corruption which does not conform to a certain corruption structure. Furthermore, from then on the adversary supplies $[\mathcal{F}]$ with all future outputs. In such a case, (perfect) simulation of a "real" protocol becomes trivial, and only attacks which conform to the corruption structure stay relevant for a distinction. The advantage of such a modeling of, say, static adversaries (i. e., adversaries which corrupt parties only in the Attack Phase of the first round, i. e., before any party is activated) is that general theorems need not be re-proven for the static case. (Of course, it is crucial that when actually *used* in a hybrid model, such a shell must be exposed only to—in that case—static adversaries.)

Anyway, inspecting the reductions of adversaries in the respective proofs shows that Lemma 3 and Theorems 4 and 5 still hold when restricting to static adversaries or adversaries that corrupt no more than a fixed number of parties; of course, here the dummy adversaries have to be modified in the obvious way.

2.5 Relation to Other Models

In [Bac03], the synchronous framework of [PSW00] is embedded in a very general way into the asynchronous setting of [PW01]. We would like to be able to conversely investigate [Can01]-functionalities and -protocols in our setting (then equipped with suitable shells, as sketched above). Yet two details which make things harder for us are that (a) due to the message-driven nature of [Can01], their machines are invoked with at most *one* incoming message at a time, and (b) in contrast to [Can01], we have chosen to activate the *adversary* (instead the environment) if an ideal functionality does not send a message in an activation. So for the time being, we can only state one relation to the frameworks of [Can01] and [CLOS02]. However, this relation can be useful since it allows for carrying over realizability results established in [Can01] or [CLOS02], thereby showing that our security notion is not "too strict".

Before we state this result, it should be noted that it is not made explicit which computational limitations the adversaries in [Can01] and [CLOS02] underlie. In [Can01, Section 3 of full version], ITMs are introduced as generally polynomially bounded in both the number of activations and running time per activation. However, the dummy adversary $\tilde{\mathcal{A}}$ (where we implicitly set the corresponding adversary class $\mathcal{C}$ to include *all* adversaries) introduced in [Can01, Section 4.4 of full version] is not polynomially bounded *a priori* in this sense. One might be tempted to reason that polynomial bounds follow implicitly from the role of $\tilde{\mathcal{A}}$ taken in a run with polynomial-time $\mathcal{Z}$ and P_i; then again, it is not straightforward how to interpret the quantification over adversaries, simulators and environments in Definition 3 of [Can01] (the actual security definition) in this case. Anyway, we believe it is reasonable to restrict only to [Can01]/[CLOS02]-adversaries, -simulators and -environments which are *a priori* polynomially bounded in both the number of activations and running time per activation (just like our equivalents).

Say that an ideal functionality $\mathcal{F}$ in the sense of [Can01] or [CLOS02] *does not communicate with the adversary* if it (a) never sends messages to the adversary, and (b) ignores all messages from the adversary (i. e., terminates its activation immediately when activated by a message from the adversary). Furthermore, for a protocol π formulated in the real model of [Can01], let $[\pi](p_{\mathsf{recv}}, \mathsf{clk})$ denote the modification of π, in which each party P_i has been substituted by $[P_i](p_{\mathsf{recv}}, 0, \mathsf{clk})$.

Ideal functionalities in the sense of [Can01] are not notified upon corruptions. We therefore denote by $[\mathcal{F}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk}, \mathsf{no\text{-}not})$ the functionality $[\mathcal{F}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$, only that the shell does *not* forward corruption notifications to $\mathcal{F}$. In contrast to [Can01], the setting of [CLOS02] allows an adversary $\mathcal{S}$ to also delay a message m sent *to* a functionality $\mathcal{F}$; however, $\mathcal{S}$ is provided with the "public headers" of m as soon as m is sent. Implicitly assuming the header information to be polynomial-time computable *in the security parameter*, we can catch this in our setting by a dedicated shell construction $[\mathcal{F}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk}, f_{\mathsf{hdr}})$. Here, upon an incoming message m, the public headers $f_{\mathsf{hdr}}(m)$ are included in the "`request receive`" message sent to $\mathcal{S}$. For [CLOS02]-functionalities $\mathcal{F}$, we assume f_{hdr} to be automatically derived from $\mathcal{F}$'s specification (cf. [CLOS02, full version]).

Proposition 7. *Let $\mathcal{F}$ be an ideal functionality which does not communicate with the adversary and let π be an n-party protocol. If π securely realizes $\mathcal{F}$ in the plain model[7] of [Can01], then $[\pi](\infty, \mathsf{async})$ securely realizes $[\mathcal{F}](0, \infty, \mathsf{async}, \mathsf{no\text{-}not})$ in our setting. Furthermore, if π securely realizes $\mathcal{F}$ in the plain model of [CLOS02], then $[\pi](\infty, \mathsf{async})$ securely realizes $[\mathcal{F}](\infty, \infty, \mathsf{async}, f_{\mathsf{hdr}})$ in our setting.*

Proof. This is proven in Appendix A. □

At this point, several questions arise. A natural question is, whether or not there are relations to the frameworks [Can00,PSW00,PW01]. The synchronous framework of [Can00] focuses on secure function evaluation and thus considers an environment machine which is *not* allowed to actually *interfere* a protocol run. In contrast, the environment machine of [Can00] only delivers party input and reads party output. We believe that it is an interesting open question whether or not the modeling of [Can00] can be related to our setting. The works [PSW00,PW01] consider a synchronous (resp., asynchronous) network, yet with a communication model and scheduling which is very different from that of [Can00,Can01,CLOS02] and ours. However, especially since the synchronous model of [PSW00] could already be embedded in a very general way into the asynchronous model of [PW01] (cf. [Bac03]), we believe that it is worthwhile to investigate a possible connection to our setting. Another interesting question seems to be to what extent the converse statements to the last proposition holds, possibly for different "functionality coats". Also it is reasonable to ask whether there can be analogous results for hybrid models in the sense of [Can01] or [CLOS02]; this is in particular interesting, as the hybrid model in these works is slightly different from ours. Namely, as mentioned above, it allows for an a priori unbounded number of instances of a single functionality, each instance identified via a unique session identifier. (For example, a straightforward analogue of Proposition 7 seems to hold for the important *common-reference-string*-hybrid model of [Can01], resp. [CLOS02]; at least as long as the protocol in question uses no more than *one* instance of the common reference string functionality $\mathcal{F}_{\mathrm{CRS}}$ investigated in [CLOS02].)

3 Global Bit Commitment is Impossible

In [BG90,GL91,CvdGT95], different protocols for realizing general secure function evaluation based only on oblivious transfer and broadcast were given—yet the notion of security used in these contributions is not simulatability-based; furthermore, these protocols can be aborted by a single party.

In this section we will show that in a reliable network with a broadcast functionality with guaranteed delivery, the primitive oblivious transfer (together with a broadcast primitive) is not complete as soon as three or more parties are involved. Namely, oblivious transfer and a broadcast channel will be proven not to be sufficient to implement a version of global bit commitment for which the output of the uncorrupted parties upon abort allows to identify who did not cooperate. Cryptographic primitives which upon abort allow to **identify** a corrupted party (which deviated from the protocol) are of special interest as they could be used to expell "disruptors" and replace their input by some default value until the protocol terminates successfully.

The constructions of [CLOS02] do not build up protocols from given primitives like oblivious transfer or broadcast, but allow to translate protocols which are secure with respect to passive adversaries into protocols which can tolerate an actively corrupted majority. The compiler in [CLOS02] is

[7] The "plain model" of [Can01] and [CLOS02] assumes authenticated message transfer, but no hybrid functionalities available.

designed for an asynchronous model and no party or functionality can know if a message is missing due to deviation of a corrupted party from the protocol, or if this message is simply not delivered by the asynchronous network.

In contrast to that, the synchronous model of communication developed in this work reflects the properties of a completely reliable network. Intuitively, this makes it impossible for the adversary to let his actions appear as network problems. Hence, in a setting with authenticated links an uncorrupted party or a functionality is always able to unambigiously identify the sender of a faulty message or a party who refuses to send a message as required in the protocol. This allows to define multi-party primitives which cannot be implemented with the primitives oblivious transfer and broadcast. The functionality introduced here is a version of the primitive *global bit commitment*, which allows to identify parties giving no proper input.

It is an interesting problem if a synchronous version of the compiler used in [CLOS02] allows to securely implement general functionalities with cheater identification in a reliable network.

Settings in which oblivious transfer or secure channels are not complete were considered in the literature before. In [FGMO01] a complete three party primitive (oblivious two cast) was presented which can implement all secure function evaluations in presence of a corrupted minority without using a broadcast channel. However, oblivious two cast cannot implement oblivious transfer if one drops the assumption of an uncorrupted majority.[8] In [MQ02], a quantum cryptographic protocol, which implements an unconditionally secure signature scheme along the line of [PW92] was presented. In the protocol of [MQ02], uncorrupted parties can decide from their view if the signer refused to sign a document or if some other party aborted the computation. As shown there, this is impossible when using only classical secure channels and a broadcast channel. The unpublished draft [MQI00] which inspired part of this work informally sketches a multi-party primitive *anonymous oblivious transfer* which is claimed to be more powerful than oblivious transfer.

Next we will describe a (single use per party) functionality $\mathcal{F}_{\mathrm{GCOM}}$, intended to formalize *global bit commitment*. Here, one party can be committed to all parties to the same bit. Moreover, using the delivery guarantee of our synchronous model, a party is either committed to all honest parties or all honest parties can deduce that P_i did not use the functionality $\mathcal{F}_{\mathrm{GCOM}}$. We will show that this functionality, which intuitively allows to detect misuse, cannot be securely realized in the $\{[\mathcal{F}_{\mathrm{OT}} \| p_{\mathrm{OT}}], [\mathcal{F}_{\mathrm{BC}}^{\ell(k)} \| p_{\mathrm{BC}}]\}$-hybrid model (cf. Appendix B for a description of the broadcast functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$ and the oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$).

Functionality $\mathcal{F}_{\mathrm{GCOM}}$

$\mathcal{F}_{\mathrm{GCOM}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. Messages not covered here are simply ignored.

- **Commit**: When receiving "`(commit,`b`)`" from a party P_i with $b \in \{0,1\}$, store the tuple (P_i, b) and send "`(receipt,`P_i`)`" to all parties and to the adversary. Ignore any future "`commit`" messages from this party P_i as well as all messages not of the form "`(commit,`P_i`,`b`)`".
- **Reveal**: When receiving a message "`(reveal)`" from a party P_i: If a tuple (P_i, b) was previously recorded, then send the message "`(reveal,`P_i`,`b`)`" to all parties and to $\mathcal{S}$. Otherwise, ignore.

Fig. 1. Functionality $\mathcal{F}_{\mathrm{GCOM}}$

[8] Then, a collusion of all parties but the sender of an oblivious cast can reconstruct everything that was sent.

Theorem 8. *In the* $\{[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}], [\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]\}$*-hybrid model (for arbitrary but fixed choices of shell parameters and polynomials* $p_{\mathrm{OT}}(k), p_{\mathrm{BC}}(k), \ell(k)$*), there is* no *functionality* $[\mathcal{F}_{\mathrm{GCOM}}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ *(with* $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ *and* $\mathsf{clk} \in \{\mathsf{async}, \mathsf{sync}\}$*) which can be securely realized for* $n \geq 3$ *parties.*

Proof. This is proven in Appendix A. □

Here a remark is in place: functionalities like $\mathcal{F}_{\mathrm{GCOM}}$ and its shell-equipped variant $[\mathcal{F}_{\mathrm{GCOM}}]$ considered above may be hard to realize for trivial reasons, since real and ideal model must be indistinguishable even if both respective adversaries have already halted. Also in some cases, it might be more suitable to restrict to functionalities which halt after a polynomial number of *rounds*. However, the result of Theorem 8 remains true when restricting to explicitly "round-bounded" functionalities $[\mathcal{F}_{\mathrm{GCOM}}]$, $[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]$, which halt after a polynomial number of *rounds*. Namely, the number of rounds each of the environments constructed in the proof of Theorem 8 runs depends *only* on the choices of the shell parameters p_{recv} and p_{send} of $[\mathcal{F}_{\mathrm{GCOM}}]$, but *not* on π. In fact, the proof holds literally for "round-bounded" functionalities $[\mathcal{F}_{\mathrm{GCOM}}]$ which halt after $2 \cdot (p_{\mathsf{recv}} + p_{\mathsf{send}} + 1)$ rounds. Moreover, any protocol π realizing a functionality $[\mathcal{F}_{\mathrm{GCOM}}]$ in a hybrid model with round-bounded $[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]$ implies a protocol π' which does so in a hybrid model with unbounded (regarding the number of rounds) $[\mathcal{F}_{\mathrm{OT}}\|p_{\mathrm{OT}}]$ and $[\mathcal{F}_{\mathrm{BC}}^{\ell(k)}\|p_{\mathrm{BC}}]$. Summarizing, the theorem holds also when restricting to round-bounded ideal functionalities.

If one allows computational assumptions as well as use of a common reference string (in form of an ideal functionality with guaranteed delivery), the functionality $\mathcal{F}_{\mathrm{GCOM}}$ *may* become realizable even in our synchronous network by a synchronous version of a protocol of [CLOS02] using a broadcast channel with guaranteed delivery. For this, one could use a non-interactive bit commitment and broadcast the commitment to all parties. As the broadcast functionality guarantees delivery this might realize a guaranteed-delivery version of $[\mathcal{F}_{\mathrm{GCOM}}]$. This *would* in particular imply that such a common reference string functionality cannot be realized by oblivious transfer and broadcast functionalities alone.

4 Conclusions and Open Questions

In this contribution a synchronous model of security was developed as an abstraction of a reliable network with guaranteed delivery. For this synchronous model a composition theorem was proven.

In the synchronous model a shell concept $[M](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ was introduced for ideal functionalities as well as for parties. A shell allows to delay incoming messages to M up to p_{recv} rounds and outgoing messages from M up to p_{send}. The parameter clk can be set to sync to have the machine M in the shell activated in each round even if no new message is to be received. For $\mathsf{clk} = \mathsf{async}$ the machine M in the shell is only activated if a message is delivered to it.

Also, shell constructs can be used for asynchronous specification of ideal functionalities. In particular, completely asynchronous executions of protocols can be modelled. It was proven that secure realizations in the setting of [Can01,CLOS02] can be transferred to secure realizations in our model if the shell parameters are set accordingly.

This work showed that security in our synchronous model with $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ is not completely covered by the the asynchronous definitions of [Can01,CLOS02]. A variant of global bit commitment was given as an example of a functionality which allows the identification of a party who did not give input to the protocol. It was shown that the functionalities oblivious transfer and broadcast do not suffice to securely realize this global bit commitment in our synchronous model. This especially

proves that the primitives oblivious transfer and broadcast are not complete in the synchronous model presented here.

We also raised questions with respect to security in reliable networks like the one considered here. Does a synchronous version of the compiler of [CLOS02] yield general realizations of ideal functionalities which allow cheater identification? Which ideal functionalities are complete for the synchronous model presented here?

References

[Bac03] Michael Backes. Unifying simulatability definitions in cryptographic systems under different timing assumptions. In Roberto Amadio and Denis Lugiez, editors, *Concurrency Theory, Proceedings of CONCUR 2003*, number 2761 in Lecture Notes in Computer Science, pages 350–365. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2003/114.ps`.

[BG90] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority. In Gilles Brassard, editor, *Advances in Cryptology, Proceedings of CRYPTO '89*, number 435 in Lecture Notes in Computer Science, pages 589–590. Springer-Verlag, 1990.

[BPSW02] Michael Backes, Birgit Pfitzmann, Michael Steiner, and Michael Waidner. Polynomial fairness and liveness. In *15th IEEE Computer Security Foundations Workshop, Proceedings of CSFW 2002*, pages 160–174. IEEE Computer Society, 2002. Online available at `http://www.zurich.ibm.com/~mbc/papers/BPSW_02Liveness.ps`.

[Can00] Ran Canetti. Security and composition of multi-party cryptographic protocols. *Journal of Cryptology*, 3(1):143–202, 2000. Full version online available at `http://eprint.iacr.org/1998/018.ps`.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42th Annual Symposium on Foundations of Computer Science, Proceedings of FOCS 2001*, pages 136–145. IEEE Computer Society, 2001. Full version online available at `http://www.eccc.uni-trier.de/eccc-reports/2001/TR01-016/revisn01.ps`.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 19–40. Springer-Verlag, 2001. Full version online available at `http://eprint.iacr.org/2001/055.ps`.

[CLOS02] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th Annual ACM Symposium on Theory of Computing, Proceedings of STOC 2002*, pages 494–503. ACM Press, 2002. Extended abstract, full version online available at `http://eprint.iacr.org/2002/140.ps`.

[CR03] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *Advances in Cryptology, Proceedings of CRYPTO 2003*, number 2729 in Lecture Notes in Computer Science, pages 265–281. Springer-Verlag, 2003. Full version online available at `http://eprint.iacr.org/2002/047.ps`.

[CvdGT95] Claude Crépeau, Jeroen van de Graaf, and Alain Tapp. Committed oblivious transfer and private multi-party computation. In Don Coppersmith, editor, *Advances in Cryptology, Proceedings of CRYPTO '95*, number 963 in Lecture Notes in Computer Science, pages 110–123. Springer-Verlag, 1995. Online available at `http://www.cs.mcgill.ca/~crepeau/PS/CGT95.ps`.

[FGMO01] Matthias Fitzi, Juan A. Garay, Ueli Maurer, and Rafail Ostrovsky. Minimal complete primitives for secure multi-party computation. In Joe Kilian, editor, *Advances in Cryptology, Proceedings of CRYPTO 2001*, number 2139 in Lecture Notes in Computer Science, pages 80–100. Springer-Verlag, 2001. Online available at `ftp://ftp.inf.ethz.ch/pub/crypto/publications/FGMO01.ps`.

[GL91] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A. Vanstone, editors, *Advances in Cryptology, Proceedings of CRYPTO '90*, number 537 in Lecture Notes in Computer Science, pages 77–93. Springer-Verlag, 1991.

[HMQS03] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On modeling IND-CCA security in cryptographic protocols. IACR ePrint Archive, February 2003. Online available at `http://eprint.iacr.org/2003/024.ps`.

[MQ02] Jörn Müller-Quade. Quantum pseudosignatures. *Journal of Modern Optics*, 49(8):1269–1276, July 2002.

[MQI00] Jörn Müller-Quade and Hideki Imai. Anonymous oblivious transfer. lanl.arXiv.org ePrint Archive, December 2000. Online available at `http://xxx.lanl.gov/ps/cs.CR/0011004`.

[PSW00] Birgit Pfitzmann, Matthias Schunter, and Michael Waidner. Secure reactive systems. Technical Report RZ 3206, IBM Zurich Research Laboratory, 2000. Online available at `http://www.semper.org/sirene/publ/PfSW1_00ReactSimulIBM.ps.gz`.

[PW92] Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In Alain Finkel and Matthias Jantzen, editors, *9th Annual Symposium on Theoretical Aspects of Computer Science, Proceedings of STACS 92*, number 577 in Lecture Notes in Computer Science, pages 339–350. Springer-Verlag, 1992. Extended abstract, online available at `http://www.semper.org/sirene/publ/PfWa_92BA1-1.ps.gz`.

[PW00] Birgit Pfitzmann and Michael Waidner. Composition and integrity preservation of secure reactive systems. In *7th ACM Conference on Computer and Communications Security, Proceedings of CCS 2000*, pages 245–254. ACM Press, 2000. Extended version online available at `http://www.semper.org/sirene/publ/PfWa_00CompInt.ps.gz`.

[PW01] Birgit Pfitzmann and Michael Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *IEEE Symposium on Security and Privacy, Proceedings of SSP 2001*, pages 184–200. IEEE Computer Society, 2001. Full version online available at `http://eprint.iacr.org/2000/066.ps`.

[Unr02] Dominique Unruh. Formale Sicherheit in der Quantenkryptologie. Institut für Algorithmen und Kognitive Systeme, Universität Karlsruhe, 2002. Student research project, online available at `http://www.unruh.de/DniQ/publications/quantum_security.ps.gz`.

A Proofs

Proof of Lemma 3. Let π, τ be protocols as in Definition 1. Assume $\pi \geq \tau$ with respect to $\mathfrak{A}$ holds. We show that then, also $\pi \geq \tau$ holds. Let therefore $\bar{\mathcal{A}}$ be an arbitrary adversary and let $p \in \mathbb{Z}[x]$ be a polynomial bounding (a) $\bar{\mathcal{A}}$'s total running time and (b) the number of steps it takes a dummy adversary to write a view sufficiently large for $\bar{\mathcal{A}}$ and additionally execute a "`corrupt`" command or write out a message as large as $\bar{\mathcal{A}}$ could have written one. (Since $\bar{\mathcal{A}}$ itself is polynomially bounded, there must be such p.) As $\pi \geq \tau$ with respect to $\mathfrak{A}$, there is a simulator $\mathcal{S}_p$ attacking τ and mimicking attacks carried out by the dummy adversary $\mathcal{A}_p$ on π. From $\mathcal{S}_p$, we build a simulator $\bar{\mathcal{S}}$ mimicking $\bar{\mathcal{A}}$.

Internally, $\bar{\mathcal{S}}$ runs simulations of both $\bar{\mathcal{A}}$ and $\mathcal{S}_p$. $\bar{\mathcal{S}}$ blocks when $\bar{\mathcal{A}}$ blocks or halts, and $\bar{\mathcal{S}}$ halts when both $\bar{\mathcal{A}}$ and $\mathcal{S}_p$ have halted. In each activation, $\bar{\mathcal{S}}$ first activates $\mathcal{S}_p$ with all incoming messages from the $\mathcal{G}_j$ relayed (up to a size and number $\mathcal{S}_p$ can handle) and—as necessary—with a faked empty message from $\mathcal{Z}$. Then, if $\mathcal{S}_p$ supplied a view yet, $\bar{\mathcal{A}}$ is activated with this view and all messages which $\bar{\mathcal{S}}$ possibly got from the environment $\bar{\mathcal{Z}}$ relayed (as many and large as $\bar{\mathcal{A}}$ can read). Corruptions by $\bar{\mathcal{A}}$ and communication of $\bar{\mathcal{A}}$ with the parties or the $\mathcal{F}_i$ are forwarded to $\mathcal{S}_p$ in form of appropriate "`corrupt`" or "`write`" commands. Corruptions issued by $\mathcal{S}_p$ are executed by $\bar{\mathcal{S}}$; moreover, $\bar{\mathcal{S}}$ relays (up to a size $\mathcal{S}_p$ can handle) $\mathcal{S}_p$'s complete communication with the $\mathcal{G}_j$ and supplies $\mathcal{S}_p$ with $\bar{\mathcal{S}}$'s own view of the parties' outgoing communication tapes. Finally, messages $\bar{\mathcal{A}}$ sends to $\bar{\mathcal{Z}}$ are forwarded; if there is no such message, $\mathcal{S}_p$ is activated with a "`next round`" command. Clearly, $\bar{\mathcal{S}}$ is polynomially bounded and halts after a polynomial number of activations; also, by construction of p, we may assume $\mathcal{S}_p$ not to halt or block in our simulation before $\bar{\mathcal{A}}$ does.

Now let $\bar{\mathcal{Z}}$ be an arbitrary environment and let $\mathcal{Z}$ be an environment which internally runs simulations of both $\bar{\mathcal{Z}}$ and $\bar{\mathcal{A}}$. Interaction of $\bar{\mathcal{Z}}$ with the parties is relayed by $\mathcal{Z}$. Views for $\bar{\mathcal{A}}$ are extracted by $\mathcal{Z}$ from views it expects the adversary $\mathcal{A}$ it interacts with to deliver; furthermore, corruptions and communication of $\bar{\mathcal{A}}$ are forwarded to $\mathcal{A}$. Each time $\bar{\mathcal{A}}$ terminates its activation in $\mathcal{Z}$'s simulation without writing messages (and thus the message delivery phase should begin), $\mathcal{Z}$ issues a "`next round`" command to $\mathcal{A}$. Lastly, $\mathcal{Z}$ halts when $\bar{\mathcal{Z}}$ halts and then forwards $\bar{\mathcal{Z}}$'s local output. So $\mathcal{Z}$ is polynomially bounded if $\bar{\mathcal{Z}}$ is and halts after polynomially many activations. By

assumption about p, the distributions $\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{A}_p, k)$ and $\bar{\mathcal{Z}}(\{\mathcal{F}_i\}, \pi, \bar{\mathcal{A}}, k)$ are identical. Similarly, $\mathcal{Z}(\{\mathcal{G}_j\}, \tau, \mathcal{S}_p, k)$ and $\bar{\mathcal{Z}}(\{\mathcal{G}_j\}, \tau, \bar{\mathcal{S}}, k)$ match. As $\bar{\mathcal{A}}$ and $\bar{\mathcal{Z}}$ were arbitrary and $\pi \geq \tau$ with respect to $\mathfrak{A}$, this proves $\pi \geq \tau$. The proof holds also for the "$\geqq$" and the "$\pi \geq \mathcal{F}$" (resp., "$\pi \geqq \mathcal{F}$") cases. □

Proof of Theorem 4. By Lemma 3, it suffices to describe for any *dummy* adversary $\mathcal{A}_p$ a corresponding simulator $\mathcal{S}$ such that for every environment $\mathcal{Z}$, the function $\mathbf{P}(\mathcal{Z}(\{\mathcal{G}_j\} \cup (\{\mathcal{F}_i\} \setminus \{\mathcal{F}\}), \pi^\tau, \mathcal{A}_p, k) = 1) - \mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i\}, \pi, \mathcal{S}, k) = 1)$ is negligible in k. In our description, we silently assume $\mathcal{S}$ to halt when $\mathcal{A}_p$ would have halted and to automatically truncate its views to reflect the p-bounding of $\mathcal{A}_p$. Assume $\tau \geq \mathcal{F}$; then there is a simulator $\bar{\mathcal{S}}$ with $\mathbf{P}(\bar{\mathcal{Z}}(\{\mathcal{G}_j\}, \tau, \bar{\mathcal{A}}_p, k) = 1) - \mathbf{P}(\bar{\mathcal{Z}}(\mathcal{F}, \bar{\mathcal{S}}, k) = 1)$ negligible in k for any $\bar{\mathcal{Z}}$. (Here, adversary $\bar{\mathcal{A}}_p$ is the dummy adversary $\mathcal{A}_p$, and we make a distinction in notation only to clarify which protocols are compared.) Our simulator $\mathcal{S}$ internally runs a simulation of $\bar{\mathcal{S}}$ and acts like $\mathcal{A}_p$, with the following exceptions. First of all, $\mathcal{S}$ blocks as soon as $\mathcal{A}_p$ would have halted. In each $\mathcal{S}$-activation, $\bar{\mathcal{S}}$ is activated exactly once, possibly with forwarded commands from $\mathcal{Z}$ (see below). Any message from $\mathcal{F}$ is forwarded to $\bar{\mathcal{S}}$, analogously for messages $\bar{\mathcal{S}}$ writes to $\mathcal{F}$. When obtaining a valid view from $\bar{\mathcal{S}}$, $\mathcal{S}$ integrates messages $\bar{\mathcal{S}}$ reports to be (a) sent between parties and (b) on $\bar{\mathcal{S}}$'s incoming communication tape into its own view after prefixing such messages with "τ". However, when $\mathcal{S}$ gets a "`corrupt` P_i" command from $\mathcal{Z}$, it first corrupts P_i, reconstructs from P_i's state when and with which input subprotocol τ was invoked and generates the history of a dummy party accordingly. (In fact, only a history as large as $\bar{\mathcal{S}}$ can read in one activation needs to be generated.) It then activates $\bar{\mathcal{S}}$ with a "`corrupt` P_i" command, supplies $\bar{\mathcal{S}}$ with the generated history as necessary, and finally merges the view which $\bar{\mathcal{S}}$ presents it into the state of P_i. (By construction of π^τ, this is canonically possible.)

If $\mathcal{S}$ is instructed to write a message to a $\mathcal{G}_j$ or to send a message prefixed with "τ" between two parties, it relays this command to $\bar{\mathcal{S}}$ after stripping the "τ" prefix. Requests by $\bar{\mathcal{S}}$ to actually *write*—possibly in the name of a corrupted party—a message to a party are buffered by $\mathcal{S}$ and included only in the party history passed to $\bar{\mathcal{S}}$ upon a corruption (and, of course, in the view of outgoing communication tapes of dummy parties $\mathcal{S}$ provides $\bar{\mathcal{S}}$ with). "`next round`" commands are forwarded to $\bar{\mathcal{S}}$, too. Finally, if $\mathcal{S}$ did *not* get a "`next round`" or a "`write`" command, it sends its view (merged as described with a view returned by $\bar{\mathcal{S}}$) to $\mathcal{Z}$. Clearly, $\mathcal{S}$ is polynomially bounded and halts after a polynomial number of activations. Furthermore, for $\bar{\mathcal{S}}$ successfully mimics $\bar{\mathcal{A}}_p$ by assumption, we may assume the simulated $\bar{\mathcal{S}}$ not to block or halt before $\mathcal{S}$ does.

Let $\mathcal{Z}$ be an environment trying to distinguish protocol π^τ and $\mathcal{A}_p$ from π and $\mathcal{S}$. From $\mathcal{Z}$, we build an environment $\bar{\mathcal{Z}}$ distinguishing τ and $\bar{\mathcal{A}}_p$ from $\mathcal{F}$ and $\bar{\mathcal{S}}$ *with the same success*. For this, $\bar{\mathcal{Z}}$ internally runs a simulation of $\mathcal{Z}$ and $\mathcal{A}_p$, functionalities $\{\mathcal{F}_i\} \setminus \{\mathcal{F}\}$ and parties running π. Here we label the non-simulated machines $\bar{\mathcal{Z}}$ interacts with $\bar{\mathcal{A}}$ and $\bar{P}_i$, respectively. The scheduling of the simulation inside $\bar{\mathcal{Z}}$ is as follows: Initially, $\mathcal{Z}$ is activated with local input "`round-start`" and the scheduling follows the one in the hybrid model; only communication of $\mathcal{A}_p$ with $\mathcal{F}$ is forwarded to $\bar{\mathcal{A}}$. Right after the simulated P_i have terminated their computation phase, messages they have possibly written to $\mathcal{F}$ are converted into input for the $\bar{P}_i$ and the (non-simulated!) computation phases are started by $\bar{\mathcal{Z}}$ sending a "`next round`" command to $\bar{\mathcal{A}}$. By its next activation, $\bar{\mathcal{Z}}$ reads output of the $\bar{P}_i$ and translates it into $\mathcal{F}$-messages in the simulation, which is then continued with the functionality computation phase. Each time the simulated $\mathcal{A}_p$ sends a view to $\mathcal{Z}$, $\bar{\mathcal{Z}}$ first queries $\bar{\mathcal{A}}$ and embeds the returned $\bar{\mathcal{A}}$-view into the $\mathcal{A}_p$-view (just like $\mathcal{S}$ embeds views from $\bar{\mathcal{S}}$). Also, corruption commands in the simulation are additionally forwarded to $\bar{\mathcal{A}}$; similarly, commands to $\mathcal{A}_p$ to write "τ"-prefixed messages are forwarded to $\bar{\mathcal{A}}$. In both cases, the view returned by $\bar{\mathcal{A}}$ is

embedded into an $\mathcal{A}_p$-view and delivered to the simulated $\mathcal{Z}$. Finally, $\bar{\mathcal{Z}}$ halts whenever $\mathcal{Z}$ halts and, in that case, forwards $\mathcal{Z}$'s output. $\bar{\mathcal{Z}}$ is polynomially bounded if $\mathcal{Z}$ is and halts after polynomially many activations.

That way, when interacting with protocol τ and adversary $\bar{\mathcal{A}} = \bar{\mathcal{A}}_p$, the simulated $\mathcal{Z}$ gets exactly the same view as when running "live" with $\mathcal{A}_p$ and protocol π^τ. On the other hand, when running in the $\mathcal{F}$-ideal model with adversary $\bar{\mathcal{A}} = \bar{\mathcal{S}}$, by construction $\mathcal{Z}$ gets the same view it would have gotten when running with $\mathcal{S}$ and π. As $\bar{\mathcal{Z}}$ simply forwards $\mathcal{Z}$'s output and $\mathcal{A}_p$ was arbitrary, it follows that $\pi^\tau \geq \pi$. (Otherwise, there was an immediate contradiction to $\tau \geq \mathcal{F}$.) Again, the proof holds also for the unconditional (the "$\gg$") case. □

Proof of Theorem 5. Fix a polynomial $p \in \mathbb{Z}[x]$ and assume $\pi \geq \tau$. Then for any dummy adversary $\bar{\mathcal{A}}_q$, there is a simulator $\bar{\mathcal{S}}_q$ such that for any $\bar{\mathcal{Z}}$, the function $\mathbf{P}(\bar{\mathcal{Z}}(\{\mathcal{F}_i\}, \pi, \bar{\mathcal{A}}_q, k) = 1) - \mathbf{P}(\bar{\mathcal{Z}}(\{\mathcal{G}_j\}, \tau, \bar{\mathcal{S}}_q, k) = 1)$ is negligible in k. Let $\mathcal{A}_q$ be an arbitrary dummy adversary attacking $\pi \| p$. We construct a simulator $\mathcal{S}$ attacking $\tau \| p$ and mimicking $\mathcal{A}_q$. Again, we silently assume $\mathcal{S}$ to honor the q-bounding of $\mathcal{A}_q$ just as in the proof of Theorem 4. Also, we assume $\mathcal{S}$ to block as soon as $\mathcal{A}_q$ would have halted. Internally, $\mathcal{S}$ keeps $\max\{p(k), 1\}$ simulations $\bar{\mathcal{S}}^j$ of $\bar{\mathcal{S}}_q$, and halts as soon as all of them have halted. Each $\bar{\mathcal{S}}^j$ inherits $\mathcal{S}$'s view of the parties' outgoing communication tapes *restricted* to (a) a size $\bar{\mathcal{S}}^j$ can handle, and (b) messages m, where (j, m) is actually sent. Messages to (resp. from) an $\bar{\mathcal{S}}^j$ are relayed while deleting (resp. prepending) a j-prefix, *except* for messages to and from the environment $\mathcal{Z}$. Upon a "`corrupt` P_i" command from $\mathcal{Z}$, $\mathcal{S}$ first corrupts P_i and then invokes all $\bar{\mathcal{S}}^j$ with a "`corrupt` P_i" command. When $\bar{\mathcal{S}}^j$ wishes to actually corrupt P_i, it is supplied with the state of the simulation P_i^j inside P_i—without loss of generality, we may assume that P_i is already corrupted. Commands to write messages of the form (j, m) with $1 \leq j \leq \max\{p(k), 0\}$ are forwarded to $\bar{\mathcal{S}}^j$, commands to write messages of a different form are only buffered by $\mathcal{S}$. (Such messages have no effect in τ, but must be reported upon later corruption of the recipient.) When a $\bar{\mathcal{S}}^j$ actually wants to write a message m in the name of a corrupted party, $\mathcal{S}$ writes (j, m) in the name of that party. "`next round`" commands are forwarded to all $\bar{\mathcal{S}}^j$. Finally, if $\mathcal{S}$ did not receive a "`write`" or "`next round`" command, it queries all $\bar{\mathcal{S}}^j$—sequentially, and allowing each of them to talk to the $\mathcal{G}_j \| p$—for a view. From these views, $\mathcal{S}$ then constructs a view eventually passed to $\mathcal{Z}$.

Now consider a $\mathcal{Z}$ with $\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i \| p\}, \pi \| p, \mathcal{A}_q, k) = 1) - \mathbf{P}(\mathcal{Z}(\{\mathcal{G}_j \| p\}, \tau \| p, \mathcal{S}, k) = 1)$ non-negligible. For $\ell \in \mathbb{N}$, let H_ℓ denote the following protocol formulated in the $(\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\})$-hybrid model (we do *not* mandate $\{\mathcal{F}_i \| p\} \cap \{\mathcal{G}_j \| p\} = \emptyset$ here). The party P_i internally keeps $\min\{\ell, p(k)\}$ simulations of the party P_i from protocol π, and $\max\{p(k) - \ell, 0\}$ simulations of the party P_i of protocol τ. Message handling and internal scheduling is just as with a p-parallelization of a protocol. Let $\mathcal{H}_\ell$ be the following adversary attacking H_ℓ. $\mathcal{H}_\ell$ is identical to $\mathcal{S}$, but simulating $\min\{\ell, p(k)\}$ copies of a dummy adversary $\bar{\mathcal{A}}_q$ and only $\max\{p(k) - \ell, 0\}$ copies of $\bar{\mathcal{S}}_q$. Message and corruption handling is just like with $\mathcal{S}$, only each of the $\min\{\ell, p(k)\}$ simulated $\bar{\mathcal{A}}_q$ gets access to one of the $\min\{\ell, p(k)\}$ respective "protocol threads" of π inside H_ℓ. Analogously, the $\bar{\mathcal{S}}_q$ get access to the respective τ-threads in H_ℓ. By construction, the distributions $\mathcal{Z}(\{\mathcal{F}_i \| p\}, \pi \| p, \mathcal{A}_q, k)$ and $\mathcal{Z}(\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\}, H_{p(k)}, \mathcal{H}_{p(k)}, k)$ match, as well as $\mathcal{Z}(\{\mathcal{G}_j \| p\}, \tau \| p, \mathcal{S}, k)$ equals $\mathcal{Z}(\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\}, H_0, \mathcal{H}_0, k)$. Consequently, for each k, there must be $m = m(k)$ with $1 \leq m \leq p(k)$ and $\mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\}, H_{m-1}, \mathcal{H}_{m-1}, k) = 1) - \mathbf{P}(\mathcal{Z}(\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\}, H_m, \mathcal{H}_m, k) = 1)$ non-negligible in k.

Let $\bar{\mathcal{Z}}$ be an environment which picks $\ell \in \{1, \ldots, p(k)\}$ uniformly and then internally simulates a complete protocol run of H_ℓ together with $\mathcal{Z}$, functionalities $\{\mathcal{F}_i \| p\} \cup \{\mathcal{G}_j \| p\}$ and adversary

$\mathcal{H}_\ell$. Rounds within this simulation are synchronized with "outside rounds", and the scheduling is straightforward. All ℓ-prefixed input $\mathcal{Z}$ gives the simulated parties is—after removing this prefix—forwarded by $\bar{\mathcal{Z}}$ to the "outside" parties $\bar{P}_i$ with which $\bar{\mathcal{Z}}$ interacts. Also, $\bar{\mathcal{Z}}$ forwards output of the $\bar{P}_i$ into its simulation after prefixing it with "ℓ". Similarly, communication of $\mathcal{H}_\ell$ with its ℓ-th simulated adversary $\bar{\mathcal{A}}_q$ is relayed by $\bar{\mathcal{Z}}$ to the adversary it interacts with. Finally, $\bar{\mathcal{Z}}$ halts when $\mathcal{Z}$ halts and then relays $\mathcal{Z}$'s output. So $\bar{\mathcal{Z}}$ is polynomially bounded if $\mathcal{Z}$ is and halts after a polynomial number of activations. Furthermore, the distributions $\mathcal{Z}(\{\mathcal{F}_i\|p\}\cup\{\mathcal{G}_j\|p\}, H_{\ell-1}, \mathcal{H}_{\ell-1}, k)$ and $\bar{\mathcal{Z}}(\{\mathcal{G}_j\|p\}, \tau, \mathcal{S}_q, k)$ are identical, and $\mathcal{Z}(\{\mathcal{F}_i\|p\}\cup\{\mathcal{G}_j\|p\}, H_\ell, \mathcal{H}_\ell, k)$ matches $\bar{\mathcal{Z}}(\{\mathcal{F}_i\|p\}, \pi, \mathcal{A}_q, k)$. As $\bar{\mathcal{Z}}$ chooses $\ell = m(k)$ with probability $1/p(k)$, this contradicts $\pi \geq \tau$. Since q and $\mathcal{Z}$ were arbitrary, $\pi\|p \geq \tau\|p$ follows. Inspection of $\bar{\mathcal{Z}}$'s running time shows that the above proof works also for the "$\gtrsim$" case.

For the second statement, the proof carries over with a minor change and an explaining remark. Namely, each party P_i in protocol H_ℓ keeps $\min\{\ell, p(k)\}$ simulations of the party P_i from protocol π, and $\max\{p(k)-\ell, 0\}$ simulations of a dummy party from protocol $D(\mathcal{F})$. The scheduling for protocol H_ℓ is identical to the ideal-model scheduling, so that each party is activated twice in each round. We therefore assume a party P_i from H_ℓ to activate its simulated π-parties only in the respective first of these activations. With these modifications, the proof given above shows that $\pi \geq \mathcal{F}$ implies $\pi\|p \geq \mathcal{F}\|p$ (similar for "$\gtrsim$"). Note here that although in the $\mathcal{F}\|p$-ideal model, the dummy parties have *not* been "parallelized", $\mathcal{F}\|p$ itself ignores inputs not of the form (j, m) for $1 \leq j \leq p(k)$. □

Proof of Proposition 7. Assume that π securely realizes $\mathcal{F}$ in the plain model of [Can01]. In the following, $\tilde{\mathcal{A}}_p$ denotes the p-restriction of the dummy adversary $\tilde{\mathcal{A}}$ of [Can01]; $\tilde{\mathcal{A}}_p$ simulates $\tilde{\mathcal{A}}$ but halts after $\max\{p(k), 1\}$ activations and quits every single activation if it runs longer than $p(k)$ steps. By assumption, for every $q \in \mathbb{Z}[x]$, there is a simulator $\tilde{\mathcal{S}}_q$ such that for any environment $\tilde{\mathcal{Z}}$, we have—in the setting and notation of [Can01]—that $\text{IDEAL}_{\mathcal{F},\tilde{\mathcal{S}}_q,\tilde{\mathcal{Z}}} \approx \text{REAL}_{\pi,\tilde{\mathcal{A}}_q,\tilde{\mathcal{Z}}}$. By assumption about $\mathcal{F}$, we may—without loss of generality—assume $\tilde{\mathcal{S}}_q$ to never send any message to $\mathcal{F}$. Let $p \in \mathbb{Z}[x]$ be arbitrary and let $\mathcal{A}_p$ be the p-bounded dummy adversary in our setting. We describe a simulator $\mathcal{S}$ mimicking $\mathcal{A}_p$; once again, we silently assume $\mathcal{S}$ to respect the p-bounding of $\mathcal{A}_p$. $\mathcal{S}$ internally keeps a simulation of $\tilde{\mathcal{S}}_q$ (for $q \in \mathbb{Z}[x]$ sufficiently large as described below) and follows these rules:

- If $\tilde{\mathcal{S}}_q$ asks to see the destinations of messages sent from $\mathcal{F}$ to the parties, $\mathcal{S}$ reports them according to "`request send`" messages received from $[\mathcal{F}]$. If $\tilde{\mathcal{S}}_q$ wants to deliver one of these messages, $\mathcal{S}$ immediately writes the respective "`allow send`" message to $[\mathcal{F}]$.
- "`write`" commands are buffered by $\mathcal{S}$. If the sender is a corrupted party and the receiver is an uncorrupted party P_i, in the respective next round, a "`request receive`" message from P_i with recipient $\mathcal{A}$ is simulated. As soon as $\mathcal{S}$ gets a "`next round`" command, all valid buffered "`allow receive`" messages and buffered messages with sender $\mathcal{A}$ are translated into requests to *deliver* the respective message and forwarded to $\tilde{\mathcal{S}}_q$.
- "`corrupt`" commands are forwarded to $\tilde{\mathcal{S}}_q$. When $\tilde{\mathcal{S}}_q$ actually wants to corrupt a party P_i, $\mathcal{S}$ first corrupts the dummy party P_i and supplies $\tilde{\mathcal{S}}_q$ with P_i's history. The party history which $\tilde{\mathcal{S}}$ then returns is later—after preparing it as follows—embedded into the view sent to $\mathcal{Z}$ (see also the next bullet). Namely, a "shell" $[P_i]$ is added and all "`request send`" and "`request receive`" messages reported earlier are included in $[P_i]$'s history. Similarly, "`allow receive`" messages $\mathcal{S}$ was advised to write are included.

– Unless invoked with a "`next round`" command, $\mathcal{S}$ eventually asks $\tilde{\mathcal{S}}_q$ to see all messages sent by the parties. To this information, all buffered messages are added. Messages not addressed to the adversary are reported as "`request send`" and—in the next round—as "`request receive`" messages from the sending, resp. the receiving party. Finally, $\mathcal{S}$ reports all these messages and the view of corrupted parties to $\mathcal{Z}$ and terminates its activation.

Let q be chosen such that $\tilde{\mathcal{S}}_q$ (a) does not halt when simulated inside $\mathcal{S}$ as above, and (b) is able to supply views at least as large as $\mathcal{S}$ does. (For fixed $\mathcal{F}$ and p, there must be such q as $\mathcal{F}$ is polynomially bounded, and $\mathcal{S}$ obeys the p-bounding of $\mathcal{A}_p$.) Then $\mathcal{S}$ is polynomially bounded and halts after polynomially many activations. Let $\mathcal{Z}$ be an arbitrary (yet bounded) environment in our setting. Let $\tilde{\mathcal{Z}}$ be a [Can01]-environment which internally simulates $\mathcal{Z}$ and halts whenever $\mathcal{Z}$ halts while relaying $\mathcal{Z}$'s output. The idea of $\tilde{\mathcal{Z}}$ is to give the simulated $\mathcal{Z}$ the view of a *synchronous* protocol run of $[\pi]$ while actually an *asynchronous* protocol run with π takes place. This means:

– At the first activation of $\tilde{\mathcal{Z}}$, $\mathcal{Z}$ gets activated with local input "`round-start`" and possible *non-uniform* local input of $\tilde{\mathcal{Z}}$ relayed (as much as $\mathcal{Z}$ can read). $\mathcal{Z}$ also gets activated each time a message from the adversary is generated for it by $\tilde{\mathcal{Z}}$ and after a "`next round`" command (see below).
– Input $\mathcal{Z}$ gives the parties is buffered until $\mathcal{Z}$ wants to send the adversary a "`next round`" command. Then, $\tilde{\mathcal{Z}}$ activates—in the order of party identities—all parties which $\mathcal{Z}$ gave input with the respective input. Then, $\mathcal{Z}$ is activated with local input "`round-start`" and all local output the parties possibly generated are displayed immediately to it. (Local output generated later by the parties are not displayed to $\mathcal{Z}$ until the next "`next round`" command.)
– If $\mathcal{Z}$ writes a message other than "`next round`" to the adversary, this message is treated by $\tilde{\mathcal{Z}}$ just like a message $\mathcal{S}$ receives from an environment. Namely, a reply to $\mathcal{Z}$ is generated by asking the adversary to report new messages and then translating this report as described above for $\mathcal{S}$. "`write`" commands are buffered and—at the next "`next round`" command from $\mathcal{Z}$ and after translating "`allow receive`" messages into delivery commands as above—forwarded to the adversary. "`corrupt` P_i" commands are forwarded to the adversary; P_i's state, which the adversary returns is then added—in form of a corresponding "shell" $[P_i]$ constructed as above—to the view $\mathcal{Z}$ is eventually supplied with. Corruption notifications $\tilde{\mathcal{Z}}$ receives are forwarded to $\mathcal{Z}$.

Clearly, $\tilde{\mathcal{Z}}$ halts after a polynomial number of activations and is polynomially bounded as $\mathcal{Z}$ is. By buffering all party in- and output and message deliveries until a "`next round`" command is issued, $\tilde{\mathcal{Z}}$ establishes for $\mathcal{Z}$ a synchronous scheduling exactly like the one in our setting. (Here it is crucial that no messages are sent between $\mathcal{F}$ and $\tilde{\mathcal{S}}_q$. Otherwise, activations of $\mathcal{F}$ could happen while $\mathcal{Z}$ and $\tilde{\mathcal{S}}_q$ are communicating; also $\tilde{\mathcal{S}}_q$ could be activated in the established "computation phase" due to an $\mathcal{F}$-message.) Moreover, in the ideal model, $\tilde{\mathcal{S}}_q$ gets the same view as when running simulated inside $\mathcal{S}$. As p and $\mathcal{Z}$ were arbitrary, this implies that the distribution ensembles $\text{IDEAL}_{\mathcal{F},\tilde{\mathcal{S}}_q,\tilde{\mathcal{Z}}}$ and $\{\mathcal{Z}(\mathcal{F},\mathcal{S},k)\}_k$, resp. the ensembles $\text{REAL}_{\pi,\tilde{\mathcal{A}}_q,\tilde{\mathcal{Z}}}$ and $\{\mathcal{Z}(\emptyset,[\pi],\mathcal{A}_p,k)\}_k$ are identical. Hence $[\pi] \geq [\mathcal{F}]$.

The proof for the second statement is analogous to the one for the first, and we describe only the differences. Namely, $\mathcal{S}$ translates "`request receive`" messages from $[\mathcal{F}]$ into messages reported to $\tilde{\mathcal{S}}_q$ as sent from a party to $\mathcal{F}$. Possible "public headers" are supplied as included in the "`request receive`" message from $[\mathcal{F}]$. Consequently, if $\tilde{\mathcal{S}}_q$ wishes to deliver a message to $\mathcal{F}$, an "`allow receive`" message is immediately written to $[\mathcal{F}]$. The rest of the proof doesn't need be to changed. □

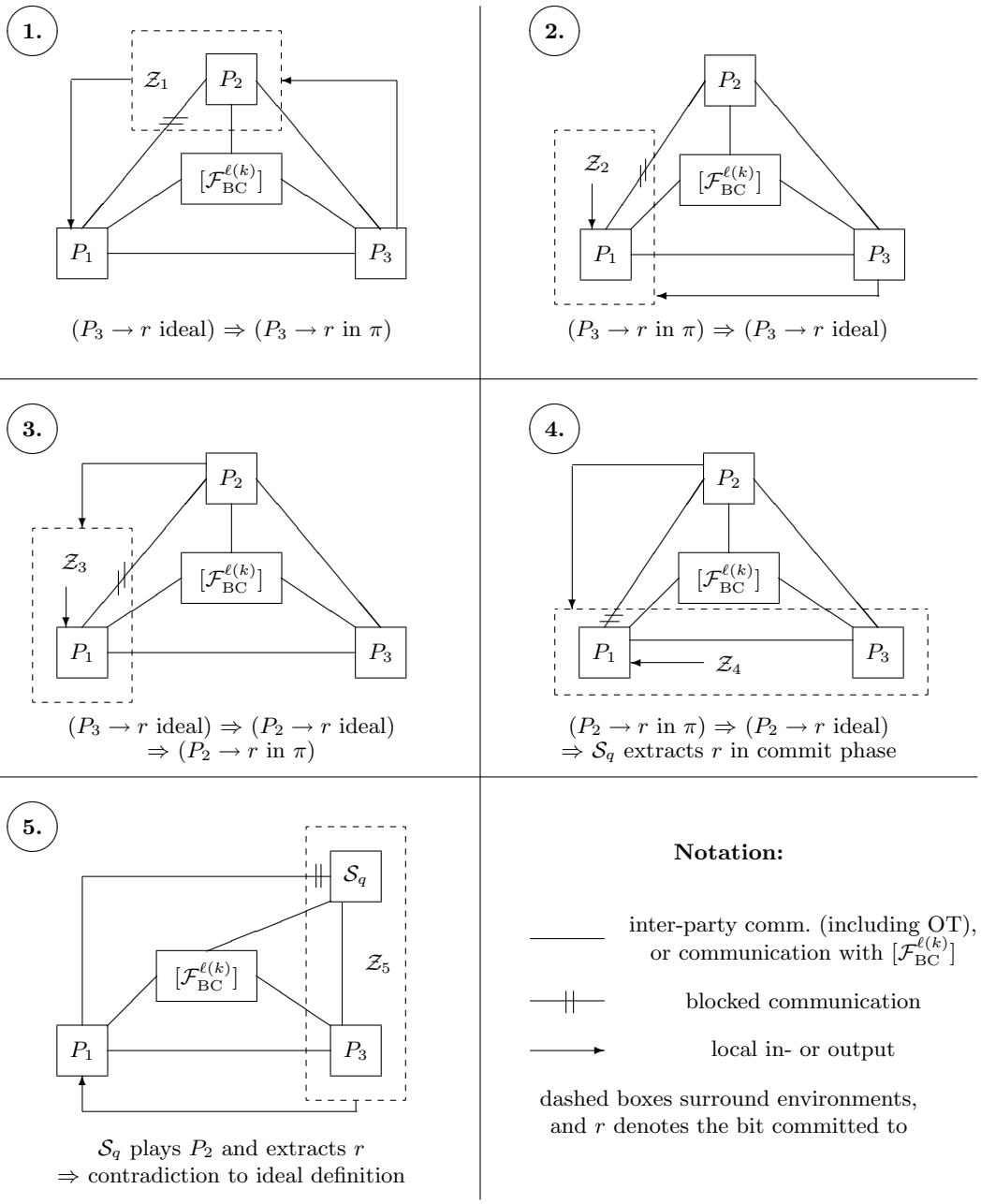

Fig. 2. Illustration of proof of Proposition 8

Proof of Theorem 8. Assume that π securely realizes a three-party (the three of which will be denoted P_1, P_2 and P_3) functionality $[\mathcal{F}_{\text{GCOM}}](p_{\mathsf{recv}}, p_{\mathsf{send}}, \mathsf{clk})$ for certain p_{recv}, $p_{\mathsf{send}} \neq \infty$ in a $\{[\mathcal{F}_{\text{OT}}\|p_{\text{OT}}], [\mathcal{F}_{\text{BC}}^{\ell(k)}\|p_{\text{BC}}]\}$-hybrid model (also for certain shell parameters and polynomials $p_{\text{OT}}(k)$, $p_{\text{BC}}(k)$, $\ell(k)$). To derive a contradiction from this, we present environments $\mathcal{Z}_i$, $i = 1, \ldots, 5$, one of which *must* be successful in distinguishing π and a dummy adversary $\mathcal{A}_q$ (for sufficiently large q) from $[\mathcal{F}_{\text{GCOM}}]$ and the corresponding simulator $\mathcal{S}_q$ (whose existence would be guaranteed by $\pi \geq [\mathcal{F}_{\text{GCOM}}]$). In our description, we assume each $\mathcal{Z}_i$ to silently truncate relayed messages as well as in- and outputs to and from parties or internal simulations of parties up to a size the sending and/or receiving party can handle. Hence q can be chosen so to enable $\mathcal{A}_q$ to execute all commands issued by any $\mathcal{Z}_i$ and supply only non-truncated views. Anticipating here, this makes each $\mathcal{Z}_i$ polynomially bounded and halt after polynomially many activations. To get a quick idea of how the argumentation below works, the proof idea has been illustrated in Figure 2. A description of $\mathcal{Z}_1$ follows.

Initially, $\mathcal{Z}_1$ lets the adversary corrupt P_2. Then $\mathcal{Z}_1$ runs a simulation $\bar{P}_2$ of party P_2 from π, initialized with a fresh random tape. $\bar{P}_2$ is activated every time $\mathcal{Z}_1$ gets activated with local input "`round-start`". All communication of $\bar{P}_2$ with ideal functionalities or the other parties is relayed via commands to the adversary, with one exception: $\bar{P}_2$'s non-broadcasted communication with P_1 is blocked. That is, all "direct" messages to and from P_1 as well as communication of $\bar{P}_2$ with $[\mathcal{F}_{\text{OT}}\|p_{\text{OT}}]$ with "sender" or "receiver" P_1 are *not* relayed.

$\mathcal{Z}_1$ itself initially gives P_1 local input "`(commit,`r`)`" for uniformly selected $r \in \{0, 1\}$. Then $\mathcal{Z}_1$ waits $p_{\mathsf{recv}}(k) + p_{\mathsf{send}}(k)$ rounds and checks for possible "`receipt`" output of P_3. If there was such output, $\mathcal{Z}_1$ gives local input "`(reveal)`" to P_1 and waits again $p_{\mathsf{recv}}(k) + p_{\mathsf{send}}(k)$ rounds. If P_3 generated local output "`(reveal,`P_1`,`$\bar{r}$`)`", $\mathcal{Z}_1$ halts with local output $r \oplus \bar{r}$. On the other hand, if P_3 generated no "`receipt`" or "`reveal`" output, $\mathcal{Z}_1$ halts with local output 1. By definition of $\mathcal{F}_{\text{GCOM}}$ and the restriction $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ on the shell parameters, $\mathcal{Z}_1$ *always* outputs 0 in the ideal model. As $\pi \geq [\mathcal{F}_{\text{GCOM}}]$ by assumption, we can hence conclude that P_3 of protocol π generates output "`(reveal,`P_1`,`r`)`" with overwhelming[9] probability when run with $\mathcal{Z}_1$.

Let $\mathcal{Z}_2$ be identical to $\mathcal{Z}_1$, except that instead of P_2, party P_1 is initially corrupted and simulated (as $\bar{P}_1$ and with a fresh random tape) by $\mathcal{Z}_2$. Message relaying is completely analogous; in particular, non-broadcasted communication of $\bar{P}_1$ with P_2 is blocked. Consequently, local "`commit`" input is given to the *simulation* $\bar{P}_1$. When being run with π and $\mathcal{A}_q$, the output of $\mathcal{Z}_2$ must be identically distributed to that of $\mathcal{Z}_1$: here, the views of $\bar{P}_1$ and P_1 are identical, as well as the views of P_2 and $\bar{P}_2$. Thus, view and local output of P_3 cannot depend on whether a run with $\mathcal{Z}_1$ or $\mathcal{Z}_2$ takes place. The definition of $\mathcal{Z}_2$ now demands that $\mathcal{S}_q$ guarantees eventual output "`(reveal,`P_1`,`r`)`" of P_3 also in the ideal model. By definition of the ideal functionality, then also the uncorrupted dummy party P_2 generates output in the ideal model. With an environment $\mathcal{Z}_3$ defined in the obvious way (i. e., identical to $\mathcal{Z}_2$, but making its own output dependent on the local output of P_2), it follows that also P_2 of π generates such local output with overwhelming probability.

Consider an environment $\mathcal{Z}_4$ identical to $\mathcal{Z}_3$, except that, in addition to P_1, also P_3 is initially corrupted and simulated (as $\bar{P}_3$ and also with a fresh random tape). This corruption is completely passive in the sense that *all* communication of $\bar{P}_3$ is relayed—here, all non-broadcast communication of the two simulations $\bar{P}_1$ and $\bar{P}_3$ is handled internally by $\mathcal{Z}_4$, that is, by simulating an instance of $[\mathcal{F}_{\text{OT}}\|p_{\text{OT}}]$ as necessary. When run with π, the output of $\mathcal{Z}_4$ is henceforth distributed identically to that of $\mathcal{Z}_3$ and thus the uncorrupted party P_2 generates output. The definition of $\mathcal{Z}_4$ forces $\mathcal{S}_q$

[9] "Overwhelming probability" means that the difference of the probability from 1 is negligible in k.

to make P_2 generate such output also in the ideal model. For doing so, $\mathcal{S}_q$ *must* send a `(commit,`r`)` message to $[\mathcal{F}_{\text{GCOM}}]$ *before* P_1 is advised to reveal the commitment to r. In other words, $\mathcal{S}_q$ must *extract* the bit committed to; this will lead to a contradiction.

Namely, consider the following, more complex environment $\mathcal{Z}_5$. Initially, $\mathcal{Z}_5$ advises the adversary to corrupt P_2 and P_3. Internally, $\mathcal{Z}_5$ keeps simulations $\bar{P}_3$ and $\bar{\mathcal{S}}_q$ of P_3 and $\mathcal{S}_q$, respectively. The *simulation* $\bar{\mathcal{S}}_q$ is initially instructed to corrupt the parties P_1 and P_3 and supplied with the initial state of dummy parties P_1, resp. P_3 when $\bar{\mathcal{S}}_q$ actually wants to corrupt these parties. The idea here is to give $\bar{\mathcal{S}}_q$ a view just as if it was interacting with $\mathcal{Z}_4$ in the ideal model, and then to use $\bar{\mathcal{S}}_q$ to extract the bit committed to. Hence,

- non-broadcasted communication of $\bar{P}_3$ with P_1 (i. e., possibly via $[\mathcal{F}_{\text{OT}}\|p_{\text{OT}}]$) is relayed (via the adversary $\mathcal{A}$ with which $\mathcal{Z}_5$ interacts) to and from the non-corrupted party P_1 —note that $\bar{\mathcal{S}}_q$ does not have access to such communication when running with $\mathcal{Z}_4$,
- non-broadcasted communication of $\bar{P}_3$ with P_2 is relayed to and from $\bar{\mathcal{S}}_q$ in form of appropriate "`write`" commands and by translating views from $\bar{\mathcal{S}}_q$—as indicated above, $\bar{\mathcal{S}}_q$ takes the role of P_2 in our setting,
- non-broadcasted communication of P_1 with the corrupted P_2 is *not* relayed—this is actually the crucial point in the proof, as both $\bar{\mathcal{S}}_q$ and $\mathcal{A}$ would expect to be able to control the oblivious transfer channel between P_1 and P_2; the blocking reflects that such communication is blocked by all $\mathcal{Z}_i$, $i = 1, \ldots, 4$,
- $\bar{\mathcal{S}}_q$ is advised to immediately deliver any message sent from P_2 to $[\mathcal{F}_{\text{BC}}^{\ell(k)}\|p_{\text{BC}}]$; "delivering" here means sending the appropriate "`allow receive`" message to the ideal functionality. The definition of $\mathcal{F}_{\text{BC}}^{\ell(k)}\|p_{\text{BC}}$ implies that the adversary—and thus $\mathcal{Z}_5$—learns the broadcasted message then. Messages broadcasted by P_2 and learned like this by $\mathcal{Z}_5$ are immediately forwarded to $\mathcal{A}$, *and* $\mathcal{A}$ is instructed to deliver that message to $\mathcal{F}_{\text{BC}}^{\ell(k)}\|p_{\text{BC}}$. Similarly, messages broadcasted by P_1 and reported by $\mathcal{A}$ are forwarded to $\bar{\mathcal{S}}_q$ and $\bar{P}_3$. Finally, messages broadcasted by $\bar{P}_3$ are forwarded to both $\mathcal{A}$ and $\bar{\mathcal{S}}_q$.

$\mathcal{Z}_5$ itself gives the party P_1 local "`(commit,`r`)`" input just as the previous $\mathcal{Z}_i$ do and waits for $\bar{\mathcal{S}}_q$ to write a "`(commit,`$\bar{r}$`)`" message in the name of P_1 to $[\mathcal{F}_{\text{GCOM}}]$. If $\bar{\mathcal{S}}_q$ writes no such message even after $p_{\mathsf{recv}}(k) + p_{\mathsf{send}}(k)$ rounds, $\mathcal{Z}_5$ outputs r, otherwise $r \oplus \bar{r}$.

The communication rules just given ensure that the view of $\bar{P}_3$ and thus that of $\bar{\mathcal{S}}_q$ when run with $\mathcal{Z}_5$ in the *hybrid* model (i. e., with π and $\mathcal{A}_q$) is identical to that of $\bar{P}_3$, resp. $\mathcal{S}_q$ when run with $\mathcal{Z}_4$ in the ideal model. By the above discussion of $\mathcal{Z}_4$, the output of $\mathcal{Z}_5$ in the hybrid model is hence 0 with overwhelming probability. On the other hand, $\mathcal{Z}_5$'s output in the ideal model *must* be a random bit, as r and a possible $\bar{r}$ are statistically independent by definition of $\mathcal{F}_{\text{GCOM}}$. Summarizing, the environment $\mathcal{Z}_5$ successfully distinguishes π from $[\mathcal{F}_{\text{GCOM}}]$ and thus we have the desired contradiction.

For the n-party case (with $n > 3$), the same argument applies if we only consider the party P_3 above to stand for the *set* $\{P_i \mid i \geq 3\}$ of parties. It should be noted here that the "guaranteed delivery" property of $[\mathcal{F}_{\text{GCOM}}]$ for shell parameters $p_{\mathsf{recv}}, p_{\mathsf{send}} \neq \infty$ is heavily used already in the first step of the proof: without such a property, we could not be sure that suppressing communication between P_1 and P_2 doesn't keep P_3 from generating output. Also should be mentioned that the last step in our proof is actually an adaption of the argument in [CF01] which shows a two-party bit commitment functionality non-realizable *without* helper functionalities such as oblivious transfer.

□

B Functionalities

Here we describe the oblivious transfer and broadcast functionalities used in Section 3.

Functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$

$\mathcal{F}_{\mathrm{BC}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. $\mathcal{F}_{\mathrm{BC}}$ is parameterized with a polynomial $\ell(k) \in \mathbb{Z}[k]$ bounding the length of broadcasted messages. All messages not covered here are ignored.

- When receiving $m \in \{0,1\}^l$ (with $1 \leq l \leq \ell(k)$) from a party P_i, send m to all parties $P_1, \ldots, P_n$ and to $\mathcal{S}$. Process at most one message per party.

Fig. 3. The broadcast functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$

Functionality $\mathcal{F}_{\mathrm{OT}}$

$\mathcal{F}_{\mathrm{OT}}$ proceeds as follows, running with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$. All messages not covered here are ignored.

- When receiving "(`sender`,P_i,P_j,x_1,x_2)" from P_i, such that $x_1, x_2 \in \{0,1\}$ and there has been no such message before for this value of (P_i, P_j), record this tuple and send "(`sender`,P_i,P_j)" to $\mathcal{S}$.
- When receiving "(`receiver`,P_i,P_j,m)" from P_j, such that $m \in \{1,2\}$ and there has been no such message before for this value of (P_i, P_j), record this tuple and send "(`receiver`,P_i,P_j)" to $\mathcal{S}$.
- If at any time, there are for some P_i, P_j tuples "(`sender`,P_i,P_j,x_1,x_2)" *and* "(`receiver`,P_i,P_j,m)" recorded, send "(`transferred`,P_i,P_j,x_m)" to P_j and "(`transferred`,P_i,P_j)" to $\mathcal{S}$.

Fig. 4. The oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$