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

Dennis Hofheinz[†] and Jörn Müller-Quade[†]

May 7, 2007

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. In the full version [HMQ04] of this paper, it is shown that a large class of protocols which are secure in the asynchronous settings [Can01, CLOS02] can be transformed into secure realizations in the new model by choosing infinite latency times.

Keywords: Security protocols, oblivious transfer, protocol composition.

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, BPW04]. 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, BPW04, 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].

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

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 which allows for universal composition, and a result allowing to carry over realizability results established in the settings of [Can01, CLOS02] into our setting.

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)*, just as in [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 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 $\mathcal{Z}$

[1]For the framework of [PW01, BPW04], this problem was addressed in [BPSW02].

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[2]) 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. First some terminology: *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. After an ITM terminates its activation, its incoming communication tape is automatically cleared to ensure future message processing.

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 $\mathcal{F}_i$.[3] 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, to the $\mathcal{F}_i$ and to $\mathcal{A}$. A detailed description of a protocol run in the $\{\mathcal{F}_i\}$-hybrid model follows.

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 written 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

[2]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 below hold also for *uniform* $\mathcal{Z}$, cf. also the discussion in [HMQS03].

[3]This is in analogy to [Can01, CLOS02] and ensures that $\mathcal{Z}$ can be used to compare two protocols using knowledge about the corruptions. Furthermore, it allows formulating strong functionalities $\mathcal{F}_i$.

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[4] 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 $\pi \gtrsim \tau$).*

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 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 "$\gtrsim$", 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. 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.

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

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 $\pi \geqq \mathcal{F}$).*

From the definitions, it is clear that the relations "$\geq$" and "$\geqq$" 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 "$\geqq$".

2.3 Composition

Due to space limitations, here we only note our security notion for protocols behaves well under universal composition. For details, cf. the full version [HMQ04].

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 modelling.)

$[M]$ 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 $M = \mathcal{F}_{\mathrm{SFE}}$ 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).

2.5 Relation to Other Models

Due to space limitations, we only note that a large class of protocols secure in the sense of [Can01] or [CLOS02] can be sensibly embedded into our model. This embedding is security-preserving. For details, cf. [HMQ04].

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 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.[5] In [MQ02],

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

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 A 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.

Figure 1: Functionality $\mathcal{F}_{\mathrm{GCOM}}$

Theorem 3 *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. A proof can be found in the full version [HMQ04]. We provide a very rough sketch: A party P_1 commits to P_2, P_3 using a protocol π assumed to realise $[\mathcal{F}_{\mathrm{GCOM}}]$. But either P_1 or P_2 blocks all point-to-point communication between P_1 and P_2, especially the oblivious transfer channel. As P_3 can from its view not decide who is cheating, the protocol π must produce a valid commitment without using any point-to-point communication between P_1 and P_2. A variant of the attack of [CF01] on universally composable bit commitment can be mounted in this situation. Thus, π must be insecure. $\square$

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 3 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 3 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. Our model allows for universal composability.

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 a large class of 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. The functionalities oblivious transfer and broadcast do not suffice to securely realize this global bit commitment in our synchronous model. This especially implies 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.

[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.

[BPW04] Michael Backes, Birgit Pfitzmann, and Michael Waidner. Secure Asynchronous Reactive Systems. IACR ePrint Archive, March 2004.

[Can00] Ran Canetti. Security and Composition of Multi-party Cryptographic Protocols. *Journal of Cryptology*, 3(1):143–202, 2000.

[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.

[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.

[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.

[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.

[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.

[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.

[HMQ04] Dennis Hofheinz and Jörn Müller-Quade. A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer. IACR ePrint Archive, January 2004.

[HMQS03] Dennis Hofheinz, Jörn Müller-Quade, and Rainer Steinwandt. On Modeling IND-CCA Security in Cryptographic Protocols. IACR ePrint Archive, February 2003.

[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.

[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.

[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.

[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.

A 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 $P_1, \dots, P_n$ 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 $1 \le l \le \ell(k)$) from a party P_i, send m to all parties $P_1, \dots, P_n$ and to $\mathcal{S}$. Process at most one message per party.

Figure 2: The broadcast functionality $\mathcal{F}_{\mathrm{BC}}^{\ell(k)}$

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

$\mathcal{F}_{\mathrm{OT}}$ proceeds as follows, running with parties $P_1, \dots, P_n$ 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 tuples "`(sender,`P_i`,`P_j`,`x_1`,`x_2`)`" *and* "`(receiver,`P_i`,`P_j`,`m`)`" for some P_i, P_j recorded, send the message "`(transferred,`P_i`,`P_j`,`x_m`)`" to P_j and the message "`(transferred,`P_i`,`P_j`)`" to $\mathcal{S}$.

Figure 3: The oblivious transfer functionality $\mathcal{F}_{\mathrm{OT}}$