Universally Composable Synchronous Computation

Jonathan Katz[1], Ueli Maurer[2], Björn Tackmann[2], and Vassilis Zikas[3*]

[1] Dept. of Computer Science, University of Maryland
jkatz@cs.umd.edu

[2] Dept. of Computer Science, ETH Zürich, Switzerland
{maurer,bjoernt}@inf.ethz.ch

[3] Dept. of Computer Science, UCLA
vzikas@cs.ucla.edu

Abstract. In synchronous networks, protocols can achieve security guarantees that are not possible in an asynchronous world: they can simultaneously achieve *input completeness* (all honest parties' inputs are included in the computation) and *guaranteed termination* (honest parties do not "hang" indefinitely). In practice truly synchronous networks rarely exist, but synchrony can be emulated if channels have (known) bounded latency and parties have loosely synchronized clocks.

The widely-used framework of universal composability (UC) is inherently asynchronous, but several approaches for adding synchrony to the framework have been proposed. However, we show that the existing proposals do *not* provide the expected guarantees. Given this, we propose a novel approach to defining synchrony in the UC framework by introducing functionalities exactly meant to model, respectively, bounded-delay networks and loosely synchronized clocks. We show that the expected guarantees of synchronous computation can be achieved given these functionalities, and that previous similar models can all be expressed within our new framework.

1 Introduction

In synchronous networks, protocols can achieve both *input completeness* (all honest parties' inputs are included in the computation) and *guaranteed termination* (honest parties do not "hang" indefinitely). In contrast, these properties cannot simultaneously be ensured in an asynchronous world [7,17].

The traditional model for synchronous computation assumes that protocols proceed in rounds: the current round is known to all parties, and messages sent in some round are delivered by the beginning of the next round. While this is a strong model that rarely corresponds to real-world networks, the model is still useful since it can be *emulated* under the relatively mild assumptions of a known bound on the network latency and loose synchronization of the (honest) parties'

* Work done while the author was at the University of Maryland.

clocks. In fact, it is fair to say that these two assumptions are exactly what is meant when speaking of "synchrony" in real-world networks.

The framework of *universal composability* (UC) [12] assumes, by default, completely asynchronous communication, where even eventual message delivery is not guaranteed. Protocol designers working in the UC setting are thus faced with two choices: either work in an asynchronous network and give up on input completeness [7] or guaranteed termination [29,15], or else modify the UC framework so as to incorporate synchronous communication somehow.

Several ideas for adding synchrony to the UC framework have been proposed. Canetti [10] introduced an ideal functionality $\mathcal{F}_{\text{SYN}}$ that was intended exactly to model synchronous communication in a general-purpose fashion. We prove in Section 5.1, however, that $\mathcal{F}_{\text{SYN}}$ does *not* provide the guarantees expected of a synchronous network. Nielsen [34] and Hofheinz and Müller-Quade [25] also propose ways of modeling synchrony with composition guarantees, but their approaches modify the foundations of the UC framework and are not sufficiently general to model, e.g., synchrony in an incomplete network, or the case when synchrony holds only in part of a network (say, because certain links do not have bounded delay while others do). It is fair to say that the proposed modifications to the UC framework are complex, and it is unclear whether they adequately capture the intuitive real-world notion of synchrony. The timing model considered in [20,23,26] extends the notion of interactive Turing machines by adding a "clock tape." It comes closer to capturing intuition, but (as we show in Section 5.2) this model also does not provide the guarantees expected from a synchronous network. A similar approach is taken in [4], which modifies the reactive-simulatability framework of [6] by adding an explicit "time port" to each automaton. Despite the different underlying framework, this work is most similar to the approach we follow here in that it also captures both guaranteed termination and incomplete networks. Their approach, however, inherently requires changing the underlying model and is based on restricting the class of adversaries (both of which we avoid). Such modifications result in (at least) a reformulation of the composition theorem and proof.

Our approach and results. We aim for an intuitively appealing model that faithfully embeds the actual real-world synchrony assumptions into the standard UC framework. The approach we take is to introduce functionalities specifically intended to (independently) model the two assumptions of bounded network delay and loose clock synchronization. An additional benefit of separating the assumptions in this way is that we can also study the case when only one of the assumptions holds.

We begin by formally defining a functionality corresponding to (authenticated) communication channels with *bounded delay.* Unfortunately, this alone is not sufficient for achieving guaranteed termination. (Throughout, we will always want input completeness to hold.) Intuitively, this is because bounded-delay channels alone—without any global clock—only provide the same "eventual message delivery" guarantee of classical asynchronous networks [7,9]. It thus becomes clear that what is missing when only bounded-delay channels are available is some notion of *time*. To rectify this, we further introduce a functionality $\mathcal{F}_{\text{CLOCK}}$ that directly corresponds to the presence of loosely synchronized clocks among the parties. We then show that $\mathcal{F}_{\text{CLOCK}}$ together with eventual-delivery channels is also not sufficient, but that standard protocols can indeed be used to securely realize any functionality with guaranteed termination in a hybrid world where both $\mathcal{F}_{\text{CLOCK}}$ and bounded-delay (instead of just eventual delivery) channels are available.

Overall, our results show that the two functionalities we propose—meant to model, independently, bounded-delay channels and loosely synchronized clocks—enable us to capture exactly the security guarantees provided by traditional synchronous networks. Moreover, this approach allows us to make use of the original UC framework and composition theorem.

Guaranteed termination. We pursue an approach inspired by constructive cryptography [31,32] to model guaranteed termination. We describe the termination guarantee as a property of functionalities; this bridges the gap between the theoretical model and the realistic scenario where the synchronized clocks of the parties ensure that the adversary *cannot* stall the computation *even if he tries to (time will advance).* Intuitively, such a functionality does not wait for the adversary *indefinitely*; rather, the environment—which represents (amongst others) the parties as well as higher level protocols—can provide the functionality with sufficiently many activations to make it proceed and eventually produce outputs, irrespective of the adversary's strategy. This design principle is applied to both the functionality that shall be realized and to the underlying functionalities formalizing the (bounded-delay) channels and the (loosely synchronized) clocks.

We then require from a protocol to realize a functionality with this guaranteed termination property, given as hybrids functionalities that have the same type of property. In more detail, following the real-world/ideal-world paradigm of the security definition, for any real-world adversary, there must be an ideal-world adversary (or simulator) such that whatever the adversary achieves in the real world can be mimicked by the simulator in the ideal world. As the functionality guarantees to terminate and produce output for any simulator, no (real-world) adversary can stall the execution of a secure protocol indefinitely.

The environment in the UC framework can, at any point in time, provide output and halt the entire protocol execution. Intuitively, however, this corresponds to the environment (which is the distinguisher) *ignoring the remainder of the random experiment*, not the adversary *stalling the protocol execution*. Any environment $\mathcal{Z}$ can be transformed into an environment $\mathcal{Z}'$ that completes the execution and achieves (at least) the same advantage as $\mathcal{Z}$.

A "polling"-based notion of time. The formalization of time we use in this work is different from previous approaches [20,23,26,33]; the necessity for the different approach stems from the inherently asynchronous scheduling scheme of the original UC model. In fact, the order in which protocols are activated in this model is determined by the communication; a party will only be activated during the execution whenever this party receives either an input or a message.

Given this model, we formalize a clock as an ideal functionality that is available to the parties running a protocol and provides a means of synchronization: the clock "waits" until all *honest* parties signal that they are finished with their tasks. This structure is justified by the following observation: the guarantees that are given to parties in synchronous models are that each party will be activated in every time interval, and will be able to perform its local actions fast enough to finish before the deadline (and then it might "sleep" until the next time interval begins). A party's confirmation that it is ready captures exactly this guarantee. As this model differentiates between honest and dishonest parties, we have to carefully design functionalities and protocols such that they do not allow *excessive* capabilities of detecting dishonest behavior. Still, the synchrony guarantee inherently *does* provide *some* form of such detections (e.g., usually by a time-out while waiting for messages, the synchrony of the clocks and the bounded delay of the channels guarantee that "honest" ones always arrive on time).

Our notion of time allows modeling both composition of protocols that run mutually asynchronously, by assuming that each protocol has its own independent clock, as well as mutually synchronous, e.g. lock-step, composition by assuming that all protocols use the same clock.

Organization of the paper. In Section 2, we include a brief description of the UC model [12] and introduce the necessary notation and terminology. In Section 3, we review the model of completely asynchronous networks, describe its limitations, and introduce a functionality modeling bounded-delay channels. In Section 4, we introduce a functionality $\mathcal{F}_{\text{CLOCK}}$ meant to model loose clock synchronization and explore the guarantees it provides. Further, we define *computation with guaranteed termination* within the UC framework, and show how to achieve it using $\mathcal{F}_{\text{CLOCK}}$ and bounded-delay channels. In Section 5, we revisit previous models for synchronous computation. Many details and, in particular, proofs have been omitted from this version but they can be found in the full version of this paper [27].

2 Preliminaries

Simulation-based security. Most general security frameworks are based on the real-world/ideal-world paradigm: In the real world, the parties execute the protocol using channels as defined by the model. In the ideal world, the parties securely access an ideal functionality $\mathcal{F}$ that obtains inputs from the parties, runs the program that specifies the task to be achieved by the protocol, and returns the resulting outputs to the parties. Intuitively, a protocol *securely realizes* the functionality $\mathcal{F}$ if, for any real-world adversary $\mathcal{A}$ attacking the protocol execution, there is an ideal-world adversary $\mathcal{S}$, also called the *simulator*, that emulates $\mathcal{A}$'s attack. The simulation is good if no distinguisher $\mathcal{Z}$—often called the *environment*—which interacts, in a well defined manner, with the parties and the adversary/simulator, can distinguish between the two worlds.

The advantage of such security definitions is that they satisfy strong composability properties. Let π_1 be a protocol that securely realizes a functionality $\mathcal{F}_1$. If a protocol π_2, using the functionality $\mathcal{F}_1$ as a subroutine, securely realizes a functionality $\mathcal{F}_2$, then the protocol $\pi_2^{\pi_1/\mathcal{F}_1}$, where the calls to $\mathcal{F}_1$ are replaced by invocations of π_1, securely realizes $\mathcal{F}_2$ (without calls to $\mathcal{F}_1$). Therefore, it suffices to analyze the security of the simpler protocol π_2 in the $\mathcal{F}_1$-*hybrid* model, where the parties run π_2 with access to the ideal functionality $\mathcal{F}_1$. A detailed treatment of protocol composition appears in, e.g., [6,11,12,18,32].

Model of computation. All security models discussed in this work are based on or inspired by the UC framework [12]. The definitions are based on the simulation paradigm, and the entities taking part in the execution (protocol machines, functionalities, adversary, and environment) are described as *interactive Turing machines* (ITMs). The execution is an interaction of *ITM instances* (ITIs) and is initiated by the environment that provides input to and obtains output from the protocol machines, and also communicates with the adversary. The adversary has access to the ideal functionalities in the hybrid models and also serves as a network among the protocol machines. During the execution, the ITIs are activated one-by-one, where the exact order of the activations depends on the considered model.

Notation, conventions, and specifics of UC '05. We consider protocols that are executed among a certain set of players $\mathcal{P}$, often referred to as the *player set*, where every $p_i \in \mathcal{P}$ formally denotes a unique party ID. A protocol execution involves the following types of ITMs: the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, the protocol machine π, and (possibly) ideal functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_m$. We say that a protocol π securely realizes $\mathcal{F}$ in the $\mathcal{F}'$-hybrid model if for each adversary $\mathcal{A}$ there exists a simulator $\mathcal{S}$ such that for all environments $\mathcal{Z}$, the contents of $\mathcal{Z}$'s output tape after an execution of π (using $\mathcal{F}'$) with $\mathcal{A}$ is indistinguishable from the contents of the tape after an execution of $\mathcal{F}$ with $\mathcal{S}$. For the details of the execution, we follow the description in [10].

As in [10], the statement "the functionality sends a *(private) delayed output* y to party i" describes the following process: the functionality requests the adversary's permission to output y to party i (without leaking the value y); as soon as the adversary agrees, the output y is delivered. The statement "the functionality sends a *public delayed output* y to party i" corresponds to the same process, where the permission request also includes the full message y.

All our functionalities $\mathcal{F}$ use *standard (adaptive) corruption* as defined in [10]. At any point in the execution, we denote by $\mathcal{H}$ the set of "honest" parties that have not (yet) been corrupted. Finally, all of our functionalities use a player set $\mathcal{P}$ that is fixed when the functionality is instantiated, and each functionality has a session ID which is of the form $sid = (\mathcal{P}, sid')$ with $sid' \in \{0,1\}^*$. We will usually omit the session ID from the description of our functionalities; different instances behave independently.

The functionalities in our model and their interpretation are specific to the model of [10] in that they exploit some of the mechanics introduced there, which we recall here. First, the order of activations is strictly defined by the model: whenever an ITI sends a message to some other ITI, the receiving ITI will immediately be activated and the sending ITI will halt. If some ITI halts without sending a message, the "master scheduler," the environment $\mathcal{Z}$, will become active. This scheme allows to model guaranteed termination since the adversary *cannot* prevent the invocation of protocol machines. Second, efficiency is defined as a "reactive" type of polynomial time: the number of steps that an ITI performs is bounded by a polynomial in the security parameter *and* (essentially) the length of the inputs obtained by this ITI. Consequently, the environment can continuously provide "run-time" to protocol machines to make them poll, e.g., at a bounded-delay or eventual-delivery channel. Our modeling of eventual delivery fundamentally relies on this fact.

3 Synchronous Protocols in an Asynchronous Network

Protocols in asynchronous networks cannot achieve input completeness and guaranteed termination simultaneously [7,17]. Intuitively, the reason is that honest parties cannot distinguish whether a message has been delayed—and to satisfy input completeness they should wait for this message—or whether the sender is corrupted and did not send the message—and for guaranteed termination they should proceed. In fact, there are two main network models for asynchronous protocols: on the one hand, there are fully asynchronous channels that do not at all guarantee delivery [10,15]; on the other hand, there are channels where delivery is guaranteed and the delay might be bounded by a publicly known constant or unknown [7]. In the following, we formalize the channels assumed in each of the two settings as functionalities in the UC framework and discuss how they can be used by round-based, i.e., synchronous, protocols. The results presented here formally confirm—in the UC framework—facts about synchrony assumptions that are known or folklore in the distributed computing literature.

3.1 Fully Asynchronous Network

The communication in a fully asynchronous network where messages are not guaranteed to be delivered is modeled by the functionality $\mathcal{F}_{\text{SMT}}$ from [10], which involves a sender, a receiver, and the adversary. Messages input by the sender p_s are immediately given to the adversary, and delivered to the receiver p_r only after the adversary's approval. Different privacy guarantees are formulated by a so-called leakage function $\ell(\cdot)$ that determines the information leaked during the transmission if both p_s and p_r are honest. In particular, the authenticated channel $\mathcal{F}_{\text{AUTH}}$ is modeled by $\mathcal{F}_{\text{SMT}}$ parametrized by the identity function $\ell(m) = m$, and the ideally secure channel $\mathcal{F}_{\text{SEC}}$ is modeled by $\mathcal{F}_{\text{SMT}}$ with the constant function $\ell(m) = \perp$. (For realistic channels obtained by encryption one typically resorts to the length function $\ell(m) = |m|$, see [14].)An important property of $\mathcal{F}_{\text{SMT}}$ is *adaptive message replacement*: the adversary can, depending on the leaked information, corrupt the sender and replace the sent message.

Canetti et al. [15] showed that, in this model and assuming a common reference string, any (well-formed) functionality can be realized, without guaranteed termination. Moreover, a combination of the results of Kushilevitz, Lindell, and Rabin [29] and Asharov and Lindell [1] show that appropriate modifications of the protocols from the seminal works of Ben-Or, Goldwasser, and Widgerson [8] and Chaum, Crépeau, and Damgård [16] (for unconditional security) or the work by Goldreich, Micali, and Widgerson [22] (for computational security)—all of which are designed for the synchronous setting—are sufficient to achieve general secure computation without termination in this asynchronous setting, under the same conditions on corruption-thresholds as stated in [8,16,22].

The following lemma formalizes the intuition that a fully asynchronous network is insufficient for terminating computation, i.e., computation which cannot be stalled by the adversary. For a functionality $\mathcal{F}$, denote by $[\mathcal{F}]^{\mathtt{NT}}$ the *non-terminating relaxation* of $\mathcal{F}$ defined as follows: $[\mathcal{F}]^{\mathtt{NT}}$ behaves as $\mathcal{F}$, but whenever $\mathcal{F}$ outputs a value to some honest party, $[\mathcal{F}]^{\mathtt{NT}}$ provides this output in a delayed manner (see Section 2). More formally, we show that there are functionalities $\mathcal{F}$ that are not realizable in the $\mathcal{F}_{\text{SMT}}$-hybrid model, but their delayed relaxations $[\mathcal{F}]^{\mathtt{NT}}$ are. This statement holds even for stand-alone security, i.e., for environments that do not interact with the adversary during the protocol execution. Additionally, the impossibility applies to all *non-trivial*, i.e., not locally computable, functionalities (see [28]) with guaranteed termination as defined in Section 4. While the lemma is implied by the more general Lemma 5, we describe the proof idea for this simpler case below.

Lemma 1. *There are functionalities $\mathcal{F}$ such that $[\mathcal{F}]^{\mathtt{NT}}$ can be realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, but $\mathcal{F}$ cannot be realized.*

Proof (idea). Consider the functionality $\mathcal{F}$ which behaves as $\mathcal{F}_{\text{SMT}}$, but with the following add-on: upon receiving a special "fetch" message from the receiver p_r, outputs y to p_r, where $y = m$ if the sender has input the message m, and $y = \perp$ (i.e., a default value), otherwise. $[\mathcal{F}]^{\mathtt{NT}}$ is realized from $\mathcal{F}_{\text{SMT}}$ channels by the dummy protocol, whereas realizing $\mathcal{F}$ is impossible. $\square$

3.2 Eventual-Delivery Channels

A stronger variant of asynchronous communication provides the guarantee that messages will be delivered *eventually*, independent of the adversary's strategy [7]. The functionality $\mathcal{F}_{\text{ED-SMT}}$ captures this guarantee, following the principle described in Section 1: The receiver can enforce delivery of the message using "fetch" requests to the channel. The potential delay of the channel is modeled by ignoring a certain number D of such requests before delivering the actual message to p_r; to model the fact that the delay might be arbitrary, we allow the adversary to repeatedly increase the value of D during the computation. Yet, the delay that $\mathcal{A}$ can impose is bounded by $\mathcal{A}$'s running time.[1] The fact that this models eventual delivery utilizes the "reactive" definition of efficiency in [10]:

[1] This is enforced by accepting the delay-number only when given in unary notation.

after the adversary determined the delay D for a certain message, the environment can still provide the protocol machines of the honest parties with sufficiently many activations to retrieve the message from the channel. The eventual delivery channel $\mathcal{F}_{\text{ED-SMT}}$ is, like $\mathcal{F}_{\text{SMT}}$, parametrized by a leakage function $\ell(\cdot)$.

Functionality $\mathcal{F}_{\text{ED-SMT}}(p_s, p_r, \ell(\cdot))$

Initialize $M := \bot$ and $D := 0$.

- Upon receiving a message m from p_s, set $D := 1$ and $M := m$ and send $\ell(M)$ to the adversary.
- Upon receiving a message (`fetch`) from p_r:
  1. Set $D := D - 1$.
  2. If $D = 0$ then send M to p_r (otherwise no message is sent and, as defined in [10], $\mathcal{Z}$ is activated).
- Upon receiving a message $(\texttt{delay}, T)$ from the adversary, if T encodes a natural number in unary notation, then set $D := D + T$; otherwise ignore the message.
- *(adaptive message replacement)*: Upon receiving $(\texttt{corrupt}, p_s, m', T')$ from $\mathcal{A}$: if $D > 0$ and T' is a valid delay, then set $D := T'$ and set $M := m'$.

Channels with eventual delivery are strictly stronger than fully asynchronous communication in the sense of Section 3.1. Indeed, the proof of Lemma 1 extends to the case where $\mathcal{F}$ is the eventual-delivery channel $\mathcal{F}_{\text{ED-SMT}}$: the simulator can delay the delivery of the message only by a polynomial number of steps, and the environment can issue sufficiently many queries at the receiver's interface.

As with fully asynchronous channels, one can use channels with eventual delivery to achieve secure computation without termination. Additionally, however, eventual-delivery channels allow for protocols which are guaranteed to (eventually) terminate, at the cost of violating input completeness. For instance, the protocol of Ben-Or, Canetti, and Goldreich [7] securely realizes any functionality where the inputs of up to $\frac{n}{4}$ parties might be ignored. Yet, the eventual-delivery channels, by themselves, do not allow to compute functionalities with strong termination guarantees. In fact, the result of Lemma 1 holds even if we replace $\mathcal{F}_{\text{SMT}}$ by $\mathcal{F}_{\text{ED-SMT}}$. This is stated in the following lemma, which again translates to both stand-alone security and to arbitrary functionalities that are not locally computable, and is again implied by Lemma 5.

Lemma 2. *There are functionalities $\mathcal{F}$ such that $[\mathcal{F}]^{\texttt{NT}}$ can be realized in the $\mathcal{F}_{\text{ED-SMT}}$-hybrid model, but $\mathcal{F}$ cannot be realized.*

3.3 Bounded-Delay Channels with a Known Upper Bound

Bounded-delay channels are described by a functionality $\mathcal{F}_{\text{BD-SMT}}$ that is similar to $\mathcal{F}_{\text{ED-SMT}}$ but parametrized by a (strictly) positive constant δ bounding the delay that the adversary can impose. In more detail, the functionality $\mathcal{F}_{\text{BD-SMT}}$

works as $\mathcal{F}_{\text{ED-SMT}}$, but queries of the adversary that lead to an accumulated delay of $T > \delta$ are ignored. Furthermore, the sender/receiver can query the functionality to learn the value δ. A formal specification of $\mathcal{F}_{\text{BD-SMT}}$ is given in the following:

Functionality $\mathcal{F}^{\delta}_{\text{BD-SMT}}(p_s, p_r, \ell(\cdot))$

Initialize $M := \perp$ and $D := 1$, and $D_t := 1$.

- Upon receiving a message m from p_s, set $D := 1$ and $M := m$ and send $\ell(M)$ to the adversary.
- Upon receiving a message (`LearnBound`) from p_s, p_r, or $\mathcal{A}$, reply with δ.
- Upon receiving a message (`fetch`) from p_r:
  1. Set $D := D - 1$.
  2. If $D = 0$, then send M to p_r.
- Upon receiving (`delay`, T) from the adversary, if $D_t + T \leq \delta$, then set $D := D + T$ and $D_t := D_t + T$; otherwise ignore the message.
- Upon receiving (`corrupt`, p_s, m', T') from $\mathcal{A}$: if $D > 0$ and T' is a valid delay, then set $D := T'$ and $M := m'$.

In reality, a channel with latency δ' is at least as useful as one with latency $\delta > \delta'$. Our formulation of bounded-delay channels is consistent with this intuition: for any $0 < \delta' < \delta$, $\mathcal{F}^{\delta}_{\text{BD-SMT}}$ can be UC-realized in the $\mathcal{F}^{\delta'}_{\text{BD-SMT}}$-hybrid model. Indeed, the simple $\mathcal{F}^{\delta'}_{\text{BD-SMT}}$-hybrid protocol that drops $\delta - \delta'$ (`fetch`)-queries realizes $\mathcal{F}^{\delta}_{\text{BD-SMT}}$; the simulator also increases the delay appropriately. The converse is not true in general: channels with smaller upper bound on the delay are *strictly* stronger when termination is required. This is formalized in the following lemma, which again extends to both stand-alone security and to non-trivial functionalities with guaranteed termination as in Section 4.

Lemma 3. *For any $0 < \delta' < \delta$, the functionality $[\mathcal{F}^{\delta'}_{\text{BD-SMT}}]^{\texttt{NT}}$ can be realized in the $\mathcal{F}^{\delta}_{\text{BD-SMT}}$-hybrid model, but $\mathcal{F}^{\delta'}_{\text{BD-SMT}}$ cannot be realized.*

The proof of Lemma 3 follows the same idea as Lemma 2 and can be found in the full version of the paper. (The proof of Lemma 2 does not use the fact that no upper bound on the network latency is known.) The technique used in the proof already suggests that bounded-delay channels, without additional assumptions such as synchronized clocks, are not sufficient for terminating computation. While Lemma 3 only handles the case where the assumed channel has a strictly positive upper-bound on the delay, the (more general) impossibility in Lemma 5 holds even for *instant-delivery* channels, i.e., bounded-delay channels which become ready to deliver as soon as they get input from the sender.

In the remainder of this paper we use instant-delivery channels, i.e., $\mathcal{F}^{\delta}_{\text{BD-SMT}}$ with $\delta = 1$; however, our results easily extend to arbitrary values of δ. To simplify notation, we completely omit the delay parameter, i.e., we write $\mathcal{F}_{\text{BD-SMT}}$ instead of $\mathcal{F}^{1}_{\text{BD-SMT}}$. Furthermore, we use $\mathcal{F}_{\text{BD-SEC}}$ and $\mathcal{F}_{\text{BD-AUTH}}$ to denote the corresponding authenticated and secure bounded-delay channel with $\delta = 1$, respectively.

4 Computation with Guaranteed Termination

Assuming bounded-delay channels is not, by itself, sufficient for achieving both input completeness and termination. In this section, we introduce the functionality $\mathcal{F}_{\text{CLOCK}}$ that, together with the bounded-delay channels $\mathcal{F}^{\delta}_{\text{BD-SMT}}$, allows synchronous protocols to satisfy both properties simultaneously. In particular, we define what it means for a protocol to UC-realize a given multi-party function *with guaranteed termination*, and show how $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}^{\delta}_{\text{BD-SMT}}\}$-protocols can satisfy this definition.

4.1 The Synchronization Functionality

To motivate the functionality $\mathcal{F}_{\text{CLOCK}}$, we examine how synchronous protocols in reality use the assumptions of bounded-delay (with a known upper bound) channels and synchronized clocks to satisfy the input-completeness and the termination properties simultaneously: they assign to each round a time-slot that is long enough to incorporate the time for computing and sending all next-round messages, plus the network delay. The fact that their clocks are (loosely) synchronized allows the parties to decide (without explicit communication) whether or not all honest parties have finished all their operations for some round. Note that it is sufficient, at the cost of having longer rounds, to assume that the clocks are not advancing in a fully synchronized manner but there is an known upper bound on the maximum clock-drift [23,26,33].

The purpose of $\mathcal{F}_{\text{CLOCK}}$ is to provide the described functionality to UC protocols. But as $\mathcal{F}_{\text{CLOCK}}$ is an ordinary UC functionality, it has no means of knowing whether or not a party has finished its intended operations for a certain round. This problem is resolved by having the parties signal their round status (i.e, whether or not they are "done" with the current round) to $\mathcal{F}_{\text{CLOCK}}$. In particular, $\mathcal{F}_{\text{CLOCK}}$ keeps track of the parties' status in a vector $(d_1, \ldots, d_n)$ of indicator bits, where $d_i = 1$ if p_i has signaled that it has finished all its actions for the current round and $d_i = 0$, otherwise. As soon as $d_i = 1$ for all $p_i \in \mathcal{H}$, $\mathcal{F}_{\text{CLOCK}}$ resets $d_i = 0$ for all $p_i \in \mathcal{P}$.[2] In addition to the notifications, any party p_i can send a synchronization request to $\mathcal{F}_{\text{CLOCK}}$, which is answered with d_i. A party p_i that observes that d_i has switched can conclude that all honest parties have completed their respective duties.[3] As $\mathcal{F}_{\text{CLOCK}}$ does *not* wait for signals from corrupted parties, $\mathcal{F}_{\text{CLOCK}}$ cannot be realized based on well-formed functionalities. Nevertheless, as discussed above, in reality time *does* offer this functionality to synchronous protocols.

[2] Whenever some party is corrupted, $\mathcal{F}_{\text{CLOCK}}$ is notified and updates $\mathcal{H}$ accordingly. This is consistent with models such as [10,34] (and, formally, requires a small change to the UC control function).

[3] For arbitrary protocols, the functionality offers too strong guarantees. Hence, we restrict ourselves to considering protocols that are either of the type described here or do not use the clock at all.

Functionality $\mathcal{F}_{\text{CLOCK}}(\mathcal{P})$

Initialize for each $p_i \in \mathcal{P}$ a bit $d_i := 0$.

- Upon receiving message (`RoundOK`) from party p_i set $d_i := 1$. If for all $p_j \in \mathcal{H} : d_j = 1$, then reset $d_j := 0$ for all $p_j \in \mathcal{P}$. In any case, send $(\texttt{switch}, i)$ to $\mathcal{A}$.[a]
- Upon receiving (`RequestRound`) from p_i, send d_i to p_i.

[a] The adversary is notified in each such call to allow attacks at any point in time.

Synchronous protocols as $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-SMT}}\}$-hybrid protocols. The code of every party is a sequence of "send," "receive," and "compute" operations, where each operation is annotated by the index of the round in which it is to be executed. In each round r, each party first receives its messages from round $r-1$, then computes and sends its messages for round r. The functionalities $\mathcal{F}_{\text{CLOCK}}$ and $\mathcal{F}_{\text{BD-SMT}}$ are used in the straightforward manner: At the onset of the protocol execution, each p_i sets its local round index to 1; whenever p_i receives a message from some entity other than $\mathcal{F}_{\text{CLOCK}}$ (i.e., from $\mathcal{Z}$, $\mathcal{A}$, or some other functionality), if a (`RoundOK`) messages has not yet been sent for the current round (i.e., the computation for the current round is not finished) the party proceeds with the computation of the current round (the last action of each round is sending (`RoundOK`) to $\mathcal{F}_{\text{CLOCK}}$); otherwise (i.e., if (`RoundOK`) has been sent for the current round), the party sends a (`RequestRound`) message to $\mathcal{F}_{\text{CLOCK}}$, which replies with the indicator bit d_i. The party p_i uses this bit d_i to detect whether or not every party is done with the current round and proceeds to the next round or waits for further activations accordingly.

In an immediate application of the above described protocol template, the resulting protocol would not necessarily be secure. Indeed, some party might start sending its round $r+1$ messages before some other party has even received its round r messages, potentially sacrificing security. (Some models in the literature, e.g. [34], allow such an ordering, while others, e.g. [25], don't.) The slackness can be overcome by introducing a "re-synchronization" round between every two rounds, where all parties send empty messages.

Perfect vs. imperfect clock synchronization. $\mathcal{F}_{\text{CLOCK}}$ models that once a single party observes that a round is completed, every party will immediately (upon activation) agree with this view. As a "real world" assumption, this means that all parties perform the round switch at exactly the same time, which means that the parties' clocks must be in perfect synchronization. A "relaxed" functionality that models more realistic synchrony assumptions, i.e., imperfectly synchronized clocks, can be obtained by incorporating "delays" as for the bounded-delay channel $\mathcal{F}_{\text{BD-SMT}}$. The high-level idea for this "relaxed" clock $\mathcal{F}^{-}_{\text{CLOCK}}$ is the following: for each party p_i, $\mathcal{F}^{-}_{\text{CLOCK}}$ maintains a value t_i that corresponds to the number of queries needed by p_i before learning that the round has switched. The adversary is allowed to choose (at the beginning of each round), for each party p_i a delay t_i

up to some upper bound $\delta > 0$. A detailed description of the functionality $\mathcal{F}_{\text{CLOCK}}^{-}$ can be found in the full version [27].

4.2 Defining Guaranteed Termination

In formalizing what it means to UC-securely compute some specification *with guaranteed termination*, we follow the principle described in Section 1. For simplicity, we restrict ourselves to non-reactive functionalities (secure function evaluation, or SFE), but our treatment can be easily extended to reactive multi-party computation. We refer to the full version [27] for details on this extension.

Let $f : (\{0,1\}^*)^n \times R \longrightarrow (\{0,1\}^*)^n$ denote an n-party (randomized) function, where the i-th component of f's input (or output) corresponds to the input (or output) of p_i, and the $(n+1)$-th input $r \in R$ corresponds to the randomness used by f. In simulation-based frameworks like [10], the secure evaluation of such a function f is generally captured by an ideal functionality parametrized by f. For instance, the functionality $\mathcal{F}_{\text{SFE}}^{f}$ described in [10] works as follows: Any honest party can either submit input to $\mathcal{F}_{\text{SFE}}^{f}$ or request output. Upon input x_i from some party p_i, $\mathcal{F}_{\text{SFE}}^{f}$ records x_i and notifies $\mathcal{A}$. When some party requests its output, $\mathcal{F}_{\text{SFE}}^{f}$ checks if all honest parties have submitted inputs; if so, $\mathcal{F}_{\text{SFE}}^{f}$ evaluates f on the received inputs (missing inputs of corrupted parties are replaced by default values), stops accepting further inputs, and outputs to p_i its output of the evaluation. We refer to [10,27] for a more detailed description of $\mathcal{F}_{\text{SFE}}^{f}$.

As described in Section 1, an ideal functionality for evaluating a function f captures *guaranteed termination* if the honest parties (or higher level protocols, which are all encompassed by the environment in the UC framework) are able to make the functionality proceed and (eventually) produce outputs, irrespective of the adversary's strategy. (Technically, we allow the respective parties to "poll" for their outputs.) The functionality $\mathcal{F}_{\text{SFE}}^{f}$ from [10] has this "terminating" property; yet, for most choices of the function f, there exists no synchronous protocol realizing $\mathcal{F}_{\text{SFE}}^{f}$ from any "reasonable" network functionality. More precisely, we say that a network-functionality $\mathcal{F}_{\text{NET}}$ provides *separable rounds* if for any synchronous $\mathcal{F}_{\text{NET}}$-hybrid protocol which communicates exclusively through $\mathcal{F}_{\text{NET}}$, $\mathcal{F}_{\text{NET}}$ activates the adversary at least once in every round.[4] The following lemma then shows that for any function f which requires more than one synchronous round to be evaluated, $\mathcal{F}_{\text{SFE}}^{f}$ cannot be securely realized by any synchronous protocol in the $\mathcal{F}_{\text{NET}}$-hybrid model. Note that this includes many interesting functionalities such as broadcast, coin-tossing, etc.

Lemma 4. *For any function f and any network functionality $\mathcal{F}_{\text{NET}}$ with separable rounds, every $\mathcal{F}_{\text{NET}}$-hybrid protocol π that securely realizes $\mathcal{F}_{\text{SFE}}^{f}$ computes its output in a single round.*

[4] In [10], this is not necessarily the case. A priori, if some ITI sends a message to some other ITI, the receiving ITI will be activated next. Only if an ITI halts without sending a message, the "master scheduler"—the environment—will be activated.

Proof (sketch). Assume, towards a contradiction, that π is a two-round protocol securely computing $\mathcal{F}^f_{\text{SFE}}$. Consider the environment $\mathcal{Z}$ that provides input to all parties and immediately requests the output from some honest party. As $\mathcal{F}_{\text{NET}}$ provides separable rounds, after all inputs have been submitted, the adversary will be activated at least twice before the protocols first generate outputs. This is not the case for the simulator in the ideal evaluation of $\mathcal{F}^f_{\text{SFE}}$. Hence, the dummy adversary cannot be simulated, which contradicts the security of π. □

To obtain an SFE functionality that matches the intuition of guaranteed termination, we need to circumvent the above impossibility by making the functionality activate the simulator during the computation. We parametrize $\mathcal{F}_{\text{SFE}}$ with a function $Rnd(k)$ of the security parameter which corresponds to the number of rounds required for evaluating f; one can easily verify that for any (polynomial) round-function $Rnd(\cdot)$ the functionality $\mathcal{F}^{f,Rnd}_{\text{SFE}}$ will terminate (if there are sufficiently many queries at the honest parties' interfaces) independently of the simulator's strategy. In each round, the functionality gives the simulator $|\mathcal{P}|+1$ activations which will allow him to simulate the activations that the parties need for exchanging their protocol messages and notifying the clock $\mathcal{F}_{\text{CLOCK}}$.

Functionality $\mathcal{F}^{f,Rnd}_{\text{SFE}}(\mathcal{P})$

$\mathcal{F}^{f,Rnd}_{\text{SFE}}$ proceeds as follows, given a function $f : (\{0,1\}^* \cup \{\perp\})^n \times R \to (\{0,1\}^*)^n$, a round function Rnd, and a player set $\mathcal{P}$. For each $p_i \in \mathcal{P}$, initialize variables x_i and y_i to a default value $\perp$ and a current delay $t_i := |\mathcal{P}|+1$. Moreover, initialize a global round counter $\ell := 1$.

- Upon receiving input $(\texttt{input}, v)$ from some party $p_i \in \mathcal{P}$, set $x_i := v$ and send a message $(\texttt{input}, i)$ to the adversary.
- Upon receiving input $(\texttt{output})$ from some party $p_i \in \mathcal{P}$, if $p_i \in \mathcal{H}$ and x_i has not yet been set then ignore p_i's message, else do:
  - If $t_i > 1$, then set $t_i := t_i - 1$. If (now) $t_j = 1$ for all $p_j \in \mathcal{H}$, then set $\ell := \ell + 1$ and $t_j := |\mathcal{P}|+1$ for all $p_j \in \mathcal{P}$. Send $(\texttt{activated}, i)$ to the adversary.
  - Else, if $t_i = 1$ but $\ell < Rnd$, then send $(\texttt{early})$ to p_i.
  - Else,
    - ∗ if x_j has been set for all $p_j \in \mathcal{H}$, and $y_1, \ldots, y_n$ have not yet been set, then choose $r \overset{R}{\leftarrow} R$ and set $(y_1, \ldots, y_n) := f(x_1, \ldots, x_n, r)$.
    - ∗ Output y_i to p_i.

Definition 1 (Guaranteed Termination). *A protocol π UC-securely evaluates a function f* with guaranteed termination *if it UC-realizes a functionality $\mathcal{F}^{f,Rnd}_{\text{SFE}}$ for some round function $Rnd(\cdot)$.*

Remark 1 (Lower Bounds). The above formulation offers a language for making UC-statements about (lower bounds on) the round complexity of certain problems in the synchronous setting. In particular, the question whether $\mathcal{F}^{f,Rnd}_{\text{SFE}}$ can be realized by a synchronous protocol corresponds to the question: "Does there exist a synchronous protocol π which securely evaluates f in $Rnd(k)$ rounds?", where k is the security parameter. As an example, the statement: "A function f needs at least r rounds to be evaluated." is (informally) translated to "There exists no synchronous protocol which UC securely realizes the functionality $\mathcal{F}_{\text{SFE}}^{f,r'}$, where $r' < r$."

The following theorem allows us to translate known results on feasibility of secure computation, e.g., [8,16,22,35], into our setting of UC with termination. (This follows from the theorem and the fact that these protocols are secure with respect to an efficient straight-line black-box simulator.) The only modification is that the protocols start with a void synchronization round where no honest party sends or receives any message. For a synchronous protocol ρ, we denote by $\hat{\rho}$ the protocol which is obtained by extending ρ with such a start-synchronization round. The proof is based on ideas from [29] and is included in the full version [27].

Theorem 1. *Let f be a function and let ρ be a protocol that, according to the notion of [11], realizes f with computational (or statistical or perfect) security in the stand-alone model, with an efficient straight-line black-box simulator. Then $\hat{\rho}$ UC-realizes f with computational (or statistical or perfect) security and guaranteed termination in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-SEC}}\}$-hybrid model with a static adversary.*

4.3 The Need for Both Synchronization and Bounded-Delay

In this section, we formalize the intuition that each one of the two "standard" synchrony assumptions, i.e., bounded-delay channels and synchronized clocks, is *alone* not sufficient for computation with guaranteed termination. We first show in Lemma 5 that bounded-delay channels (even with instant delivery) are, by themselves, not sufficient; subsequently, we show in Lemma 6 that (even perfectly) synchronized clocks are also not sufficient, even in combination with eventual-delivery channels (with no known bound on the delay).

Lemma 5. *There are functions f such that for any (efficient) round-function Rnd and any $\delta > 0$: $[\mathcal{F}_{\text{SFE}}^{f,Rnd}]^{\texttt{NT}}$ can be realized in the $\mathcal{F}_{\text{BD-SMT}}^{\delta}$-hybrid model, but $\mathcal{F}_{\text{SFE}}^{f,Rnd}$ cannot.*

Proof (idea). Consider the two-party function f which, on input a bit $x_1 \in \{0,1\}$ from party p_1 (and nothing from p_2), outputs x_1 to p_2 (and nothing to p_1). The functionality $\mathcal{F}_{\text{SFE}}^{f,Rnd}$ guarantees that an honest p_1 will be able to provide input, independently of the adversary's behavior. On the other hand, a corrupted p_1 will not keep p_2 from advancing (potentially with a default input for p_1).[5] However, in the real world, the behavior of the bounded-delay channel in the above two cases is identical.

[5] This capability of distinguishing "honest" from "dishonest" behavior is key in synchronous models: as honest parties are guaranteed that they can send their messages on time, dishonest parties will blow their cover by not adhering to the deadline.

On the other hand, the functionality $[\mathcal{F}_{\text{SFE}}^{f,Rnd}]^{\texttt{NT}}$ can be realized from $\mathcal{F}_{\text{BD-SMT}}$: p_1 simply has to send the input to p_2 via the $\mathcal{F}_{\text{BD-SMT}}$-channel. The simulator makes sure that the output in the ideal model is delivered to the p_2 only after $\mathcal{Z}$ acknowledges the delivery. A detailed proof can be found in [27]. $\square$

In reality, synchronous clocks alone are not sufficient for synchronous computation if there is no *known* upper bound on the delay of the channels (even with guaranteed *eventual* delivery); this statement is formalized using the clock functionality $\mathcal{F}_{\text{CLOCK}}$ and the channels $\mathcal{F}_{\text{ED-SMT}}$ in the following lemma. The proof is similar to the proof of Lemma 1 and can be found in [27].

Lemma 6. *There are functions f such that for any (efficient) round-function Rnd: $[\mathcal{F}_{\text{SFE}}^{f,Rnd}]^{\texttt{NT}}$ can be realized in the $\{\mathcal{F}_{\text{ED-SMT}}, \mathcal{F}_{\text{CLOCK}}\}$-hybrid model, but $\mathcal{F}_{\text{SFE}}^{f,Rnd}$ cannot.*

4.4 Atomicity of Send/Receive Operations and Rushing

Hirt and Zikas [24] pointed out that the standard formulation of a "rushing" adversary [11] in the synchronous setting puts a restriction on the order of the send/receive operations within a synchronous round. The modularity of our framework allows to pinpoint this restriction by showing that the rushing assumption corresponds to a "simultaneous multi-send" functionality which cannot even be realized using $\mathcal{F}_{\text{CLOCK}}$ and $\mathcal{F}_{\text{BD-SMT}}$.

Intuitively, a rushing adversary [11] cannot preempt a party while this party is sending its messages of some round. This is explicitly stated in [11], where the notion of "synchronous computation with *rushing*" is defined (cf. [11, Page 30]). In reality, it is arguable whether we can obtain the above guarantee by just assuming bilateral bounded-delay channels and synchronized clocks. Indeed, sending multiple messages is typically not an atomic operation, as the messages are buffered on the network interface of the computer and sent one-by-one. Hence, to achieve the simultaneity, one has to assume that the total time it takes for the sender to put all the messages on the network minus the *minimum* latency of the network is not sufficient for a party to become corrupted.

The "simultaneous multi-send" guarantee is captured in the following UC-functionality, which is referred to as the *simultaneous multi-send* channel, and denoted by $\mathcal{F}_{\text{MS}}$. On a high level, $\mathcal{F}_{\text{MS}}$ can be described as a channel allowing a sender p_i to send a vector of messages $(x_1, \ldots, x_n)$ to the respective receivers $p_1, \ldots, p_n$ as an atomic operation. The formal description of $\mathcal{F}_{\text{MS}}$ is similar to $\mathcal{F}_{\text{BD-SMT}}$ with the following modifications: First, instead of a single receiver, there is a set $\mathcal{P}$ of receivers, and instead of a single message, the sender inputs a vector of $|\mathcal{P}|$ messages, one for each party in $\mathcal{P}$. As soon as some party receives its message, the adversary cannot replace any of the remaining messages that correspond to honest receivers, not even by corrupting the sender. As in the case of bounded-delay channels, we denote by $\mathcal{F}_{\text{MS-AUTH}}$ the multi-send channel which leaks the transmitted vector to the adversary. The following lemma states that the delayed relaxation of $\mathcal{F}_{\text{MS-AUTH}}$ cannot be realized from $\mathcal{F}_{\text{BD-SEC}}$ and $\mathcal{F}_{\text{CLOCK}}$

when arbitrary many parties can be corrupted. This implies that $\mathcal{F}_{\text{MS-AUTH}}$ can also not be realized from $\mathcal{F}_{\text{BD-SEC}}$ and $\mathcal{F}_{\text{CLOCK}}$.

Functionality $\mathcal{F}_{\text{MS}}(\ell, i, \mathcal{P})$

- Upon receiving a vector of messages $\boldsymbol{m} = (m_1, \ldots, m_n)$ from p_i, record $\boldsymbol{m}$ and send a message $(\texttt{sent}, \ell(\boldsymbol{m}))$ to the adversary.
- Upon receiving $(\texttt{fetch})$ from $p_j \in \mathcal{P}$, output m_j to p_j ($m_j = \perp$ if $\boldsymbol{m}$ has not been recorded).
- *(restricted response to* `replace`*)* Upon receiving a $(\texttt{replace}, \boldsymbol{m}')$ request from the adversary for replacing p_i's input (after issuing a request for corrupting p_i), if no (honest or corrupted) p_j received m_j *before* p_i got corrupted, then replace $\boldsymbol{m}$ by $\boldsymbol{m}'$.

Lemma 7. *Let $\mathcal{P}$ be a player set with $|\mathcal{P}| > 3$. Then there exists no protocol which UC-realizes $[\mathcal{F}_{\text{MS-AUTH}}]^{\texttt{NT}}$ in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-AUTH}}\}$-hybrid model and tolerates a corrupted majority.*

Proof (sketch). Garay et al. [21] showed that if atomic multi-send (along with a setup for digital signatures) is assumed, then the broadcast protocol from Dolev and Strong [19] UC-realizes broadcast (without guaranteed termination) in the presence of an adaptive adversary who corrupts any number of parties. Hence, if there exist a protocol for realizing $[\mathcal{F}_{\text{MS-AUTH}}]^{\texttt{NT}}$ in the synchronous model, i.e., in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-AUTH}}\}$-hybrid world, with corrupted majority and adaptive adversary, then one could also realize broadcast in this model, contradicting the impossibility result from [24]. $\square$

The above lemma implies that the traditional notion of "synchronous computation with *rushing*" cannot be, in general, achieved in the UC model unless some non-trivial property is assumed on the communication channel. Yet, $[\mathcal{F}_{\text{MS-AUTH}}]^{\texttt{NT}}$ *can* be UC-realized from $\{[\mathcal{F}_{\text{BD-AUTH}}]^{\texttt{NT}}, [\mathcal{F}_{\text{COM}}]^{\texttt{NT}}\}$, where $\mathcal{F}_{\text{COM}}$ denotes the standard UC-commitment functionality [13]. The idea is the following: In order to simultaneously multi-send a vector $(x_1, \ldots, x_n)$ to the parties $p_1, \ldots, p_n$, the sender sends an independent commitment on x_i to every recipient p_i, who acknowledges the receipt (using the channel $[\mathcal{F}_{\text{BD-SEC}}]^{\texttt{NT}}$). After receiving all such acknowledgments, the sender, in a second round, opens all commitments. The functionality $\mathcal{F}_{\text{COM}}$ ensures that the adversary (unless the sender is corrupted in the first round) learns the committed messages x_i only after every party has received the respective commitment; but, from that point on, $\mathcal{A}$ can no longer change the committed message. For completeness we state the above in the following lemma.

Lemma 8. *There is a synchronous $\{[\mathcal{F}_{\text{BD-AUTH}}]^{\texttt{NT}}, [\mathcal{F}_{\text{COM}}]^{\texttt{NT}}\}$-hybrid protocol that UC-realizes $[\mathcal{F}_{\text{MS-AUTH}}]^{\texttt{NT}}$.*

Using Lemmas 7 and 8, and the fact that the delayed relaxation of any $\mathcal{F}$ can be realized in the $\mathcal{F}$-hybrid model, we can extend the result of [13] on impossibility of UC commitments to our synchronous setting.

Corollary 1. *There exists no protocol which UC-realizes the commitment functionality $\mathcal{F}_{\text{COM}}$ in the $\{\mathcal{F}_{\text{BD-AUTH}}, \mathcal{F}_{\text{CLOCK}}\}$-hybrid model.*

5 Existing Synchronous Models as Special Cases

In this section, we revisit existing models for synchronous computation. We show that the $\mathcal{F}_{\text{SYN}}$-hybrid model as specified in [10] and the Timing Model from [26] are sufficient only for non-terminating computation (which can be achieved even in a fully asynchronous environment). We also show that the models of [34] and [25] can be expressed as special cases in our model. Many details are omitted; we refer to the full version of this paper [27] for a complete treatment.

5.1 The $\mathcal{F}_{\text{SYN}}$-Hybrid Model

In [10], a model for synchronous computation is specified by a *synchronous network* functionality $\mathcal{F}_{\text{SYN}}$. On a high-level, $\mathcal{F}_{\text{SYN}}$ corresponds to an authenticated network with storage, which proceeds in a round-based fashion; in each round r, every party associated to $\mathcal{F}_{\text{SYN}}$ inputs a vector of messages, where it is guaranteed that (1) the adversary cannot change the message sent by an honest party without corrupting this party, and (2) the round index is only increased after every honest party as well as the adversary have submitted their messages for that round. Furthermore, $\mathcal{F}_{\text{SYN}}$ allows the parties to query the current value of r along with the messages of that round r.

$\mathcal{F}_{\text{SYN}}$ requires the adversary to explicitly initiate the round switch; this allows the adversary to stall the protocol execution (by not switching rounds). Hence, $\mathcal{F}_{\text{SYN}}$ cannot be used for evaluating a non-trivial[6] function f with guaranteed termination: because we require termination, for every protocol which securely realizes $\mathcal{F}_{\text{SFE}}$ and for every adversary, the environment $\mathcal{Z}$ which gives inputs to all honest parties and issues sufficiently many `fetch` requests has to be able to make π generate its output from the evaluation. This must, in particular, hold when the adversary never commands $\mathcal{F}_{\text{SYN}}$ to switch rounds, which leads to a contradiction.

Lemma 9. *For every non-trivial n-party function f and for every round function Rnd there exists no $\mathcal{F}_{\text{SYN}}$-hybrid protocol which securely realizes $\mathcal{F}_{\text{SFE}}^{f,Rnd}$ with guaranteed termination.*

The only advantage that $\mathcal{F}_{\text{SYN}}$ offers on top of what can be achieved from (asynchronous) bounded-delay channels is that $\mathcal{F}_{\text{SYN}}$ defines a point in the computation (chosen by $\mathcal{A}$) in which the round index advances for all parties *simultaneously.* More precisely, denote by $\mathcal{F}_{\text{SYN}}^{-}$ the functionality that behaves as $\mathcal{F}_{\text{SYN}}$, except for a small modification upon receiving the (`Advance-Round`)-message: $\mathcal{F}_{\text{SYN}}^{-}$ advances the round, but, for each party, allows the adversary to further delay the output (initially, the (`receive`)-requests of p_i are still answered with

[6] Recall that a non-trivial function is one that cannot be computed locally (cf. [28]).

the previous round messages). This is only a mild relaxation of the functionality: the adversary can delay the delivery of the new messages, but the important property that the messages are "committed" by the time of the round switch is preserved.[7]

The functionality $\mathcal{F}_{\text{SYN}}^{-}$ can be realized by a protocol using asynchronous channels with eventual delivery. This protocol follows the ideas of [2,26]: In each round r, each party p_i sends a message to all other parties. After receiving messages from all p_j, p_i sends an acknowledgment to all p_j. Once all such acknowledgments have been received, p_i prepares the messages received in that round for local output (upon request) and starts the next round (as soon as messages have been provided as local input). This proves the following lemma.

Lemma 10. *There exist a protocol that UC-realizes the functionality $\mathcal{F}_{\text{SYN}}^{-}$ in the $\mathcal{F}_{\text{BD-SMT}}$-hybrid model.*

5.2 The Timing Model

The "Timing model" [23,26] integrates a notion of time into the protocol execution by extending the model of computation. Each party, in addition to its communication and computation tapes, has a *clock tape* that is writable for the adversary in a monotone and "bounded-drift"-preserving manner: The adversary can only increase the value of the clocks, and, for any two parties, the distance ϵ of their clocks' speed (drift) at any point in time is bounded by a known constant. The value of a party's clock-tape defines the local time of this party. Depending on this time, protocols delay sending messages or time-out if a message has not arrived as expected. We formalize this modeling of time in UC in the following straightforward manner: We introduce a functionality $\mathcal{F}_{\text{TIME}}$ that maintains a clock value for each party and allows the adversary to advance this clock in the same monotone and "bounded-drift"-preserving way. Instead of reading the local clock tape, $\mathcal{F}_{\text{TIME}}$-hybrid protocols obtain their local time value by querying $\mathcal{F}_{\text{TIME}}$.

Lemma 11 shows that $\mathcal{F}_{\text{TIME}}$ can be realized from fully asynchronous authenticated communication. The idea of the proof is the following: The protocol τ that realizes the functionality $\mathcal{F}_{\text{TIME}}$ from pairwise authenticated channels maintains, for each party p_i, a local integer variable t_i that corresponds to p_i's local time. Using the authenticated network, the parties ensure that the local time values increase with bounded drift. Together, Lemmas 11 and 12 demonstrate that "timed" protocols cannot UC-realize more functionalities than non-"timed" protocols, which is consistent with [26, Theorem 2] and implies that the Timing Model does not allow for computation with guaranteed termination.

Lemma 11. *Let $\mathcal{P}$ be a player set and $\epsilon \geq 1$. The functionality $\mathcal{F}_{\text{TIME}}(\mathcal{P}, \epsilon)$ can be UC-realized from pairwise authenticated channels $\mathcal{F}_{\text{AUTH}}$.*

[7] The delay can only be detected if the parties have access to a channel which delivers faster than the specified delay. The parties overcome this slackness by issuing (`receive`)-queries until they obtain the desired output.

In contrast to $\mathcal{F}_{\text{SYN}}$, which "lives" in the UC framework, security statements in the Timing Model cannot be automatically transferred to the UC setting. Indeed, there is a "type-mismatch" between functionalities/protocols in the two frameworks, which we resolve by the following idea inspired by [3,30,32]: A functionality $\mathcal{F}$ in the timing model is compiled to a functionality in UC which behaves exactly as $\mathcal{F}$ but ensures that the interfaces are compatible with the UC model of computation. On a high-level, the functionality compiler $T_T(\cdot)$ works as follows: $T_T(\mathcal{F})$ behaves as $\mathcal{F}$, but on any input from an honest party it notifies the adversary (without leaking the contents). Whenever $\mathcal{F}$ outputs a value y to some party, $T_T(\mathcal{F})$ issues a (private) delayed output y instead.

Lemma 12 then shows that any security statement about a functionality $\mathcal{F}$ in the Timing Model can be translated into a statement about $T_T(\mathcal{F})$ in the $\{\mathcal{F}_{\text{TIME}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model (in UC). The translation is both constructive and uniform, i.e., we describe a protocol compiler $C_T(\cdot)$ that translates a protocol in the Timing Model into a corresponding one in the $\{\mathcal{F}_{\text{TIME}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model.

Lemma 12. *For an arbitrary functionality $\mathcal{F}$ and a protocol π in the Timing Model, π securely realizes $\mathcal{F}$ (in the Timing Model) if and only if the compiled protocol $C_T(\pi)$ UC-realizes $T_T(\mathcal{F})$ in the $\{\mathcal{F}_{\text{TIME}}, \mathcal{F}_{\text{AUTH}}\}$-hybrid model in the presence of a static adversary.*

5.3 Models with Explicit Round-Structure

Nielsen's framework [34]. The framework described in [34] is an adaptation of the asynchronous framework of [12] to authenticated synchronous networks. While the general structure of the security definition is adopted, the definition of protocols and their executions differs considerably. For instance, the "subroutine" composition of two protocols is defined in a "lock-step" way: the round switches occur at the same time. Similarly to our bounded-delay channels, messages in transfer can be replaced if the sender becomes corrupted. Lemma 13 allows to translate, along the lines of Section 5.2, any security statement in the model of [34] into a security statement about a synchronous protocol in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-AUTH}}\}$-hybrid model. As in the previous section, the translation is done by a functionality compiler $T_N(\cdot)$ that resolves the type-mismatch between the functionalities in UC and in [34], and a corresponding protocol compiler $C_N(\cdot)$. We emphasize that the converse statement of Lemma 13 does *not* hold, i.e., there are UC statements about synchronous protocols that cannot be modeled in the [34] framework. For instance, our synchronous UC model allows protocols to use further functionalities that run mutually asynchronously with the synchronous network, which cannot be modeled in [34].

Lemma 13. *For an arbitrary functionality $\mathcal{F}$ and a protocol π in [34], π securely realizes $\mathcal{F}$ (in [34]) if and only if the compiled protocol $C_N(\pi)$ UC-realizes $T_N(\mathcal{F})$ in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{BD-AUTH}}\}$-hybrid model.*

Hofheinz and Müller-Quade's framework [25]. The framework of [25] also models authenticated synchronous networks based on the framework of [12], but the rules of the protocol execution differ considerably: The computation proceeds in rounds, and each round is split into three phases. In each phase, only a subset of the involved ITIs are activated, and the order of the activations follows a specific scheme. The adversary has a relaxed *rushing* property: while being the last to specify the messages for a round, he cannot corrupt parties within a round. This corresponds to a network with guarantees that are stronger than simultaneous multi-send: once the first message of an honest party is provided to the adversary, all messages of honest parties are guaranteed to be delivered correctly.[8] We model this relaxed rushing property in UC by the functionality $\mathcal{F}_{\text{MS+}}$ (cf. [27]), which is a modified version of $\mathcal{F}_{\text{MS}}$ and exactly captures this guarantee. As before, we translate the security statements of [25] to our model (where $\mathcal{F}_{\text{MS+}}$ is used instead of $\mathcal{F}_{\text{AUTH}}$) through a pair of compilers $(T_H(\cdot), C_H(\cdot))$.

Lemma 14. *For an arbitrary functionality $\mathcal{F}$ and a protocol π in [25], π securely realizes $\mathcal{F}$ (in [25]) if and only if the compiled protocol $C_H(\pi)$ UC-realizes $T_H(\mathcal{F})$ in the $\{\mathcal{F}_{\text{CLOCK}}, \mathcal{F}_{\text{MS+}}\}$-hybrid model.*

6 Conclusion

We described a modular security model for synchronous computation within the (otherwise inherently asynchronous) UC framework by specifying the real-world synchrony assumptions of bounded-delay channels and loosely synchronized clocks as functionalities. The design principle that underlies these functionalities allows us to treat guaranteed termination; previous approaches for synchronous computation within UC either required fundamental modifications of the framework (which also required re-proving fundamental statements) or did not allow to make such statements altogether. Given this model, we revisited basic results from the literature on synchronous protocols, formalizing and proving them within the UC framework. Finally, we showed that previous specialized frameworks can be cast as special cases of our model by introducing network functionalities that provide the guarantees formalized in those models.

Acknowledgments. We thank Ran Canetti for interesting discussions and comments on the comparison of our model with the $\mathcal{F}_{\text{SYN}}$-hybrid formulation. The first author was supported by NSF awards #0447075, #1111599, and #1223623. The second and third authors were supported by the Swiss National Science Foundation (SNF), project no. 200020-132794. The fourth author was partially supported by a fellowship from the SNF, project no. PBEZP2-134445.

[8] In the context of [25], this is an advantage as it strengthens the impossibility result.

References

1. Asharov, G., Lindell, Y., Rabin, T.: Perfectly-Secure Multiplication for Any $t < n/3$. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 240–258. Springer, Heidelberg (2011)
2. Awerbuch, B.: Complexity of Network Synchronization. Journal of the ACM 32, pp. 804–823 (1985)
3. Backes, M.: Unifying Simulatability Definitions in Cryptographic Systems under Different Timing Assumptions. In: Amadio, R., Lugiez, D. (eds.) CONCUR 2003. LNCS, vol. 2761, pp. 350–365. Springer, Heidelberg (2003)
4. Backes, M., Hofheinz, D., Müller-Quade, J., Unruh, D.: On Fairness in Simulatability-based Cryptographic Systems. In: Proceedings of FMSE, pp. 13–22. ACM (2005)
5. Backes, M., Pfitzmann, B., Steiner, M., Waidner, M.: Polynomial Fairness and Liveness. In: Proceedings of the 15th Annual IEEE Computer Security Foundations Workshop, pp. 160–174. IEEE (2002)
6. Backes, M., Pfitzmann, B., Waidner, M.: The Reactive Simulatability (RSIM) Framework for Asynchronous Systems. Information and Computation 205, pp. 1685–1720 (2007)
7. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous Secure Computation. In: Proceedings of the 25th Annual ACM Symposium on Theory of Computing, pp. 52–61. ACM (1993)
8. Ben-Or, M., Goldwasser, S., Widgerson, A.: Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 1–10. ACM (1988)
9. Canetti, R.: Studies in Secure Multiparty Computation and Applications. PhD thesis, The Weizmann Institute of Science (1996)
10. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: Cryptology ePrint Archive, Report 2000/067 (2005)
11. Canetti, R.: Security and Composition of Multiparty Cryptographic Protocols. In: Journal of Cryptology 13, pp. 143–202 (2000)
12. Canetti, R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, pp. 136–145. IEEE (2001)
13. Canetti, R., Fischlin, M.: Universally Composable Commitments. In: Kilian, J. (ed.) CRYPTO 2001, LNCS, vol. 2139, pp. 19–40. Springer, Heidelberg (2001)
14. Canetti, R., Krawczyk, H.: Universally Composable Notions of Key Exchange and Secure Channels. In: Knudsen, L. R. (ed.) EUROCRYPT 2002, LNCS, vol. 3027, pp. 337–351. Springer, Heidelberg (2002)
15. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally Composable Two-Party and Multi-Party Secure Computation. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 494–503. ACM (2002)
16. Chaum, D., Crépeau, C., Damgård, I.: Multiparty Unconditionally Secure Protocols. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing, pp. 11–19. ACM (1988)
17. Chor, B., Moscovici, L.: Solvability in Asynchronous Environments. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, pp. 422–427. IEEE (1989)
18. Dodis, Y., Micali, S.: Parallel Reducibility for Information-Theoretically Secure Computation In: Bellare, M. (ed.) CRYPTO 2000. LNCS, vol. 1880, pp. 74–92. Springer, Heidelberg (2000)

19. Dolev, D., Strong, H. R.: Polynomial Algorithms for Multiple Processor Agreement. In: Proceedings of the 14th Annual ACM Symposium on Theory of Computing, pp. 401–407. ACM (1982)
20. Dwork, C., Naor, M., Sahai, A.: Concurrent Zero-Knowledge. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 409–418. ACM (1998)
21. Garay, J. A., Katz, J., Kumersan, R., Zhou, H.-S.: Adaptively Secure Broadcast, Revisited. In: Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, pp. 179–186. ACM (2011)
22. Goldreich, O., Micali, S., Widgerson, A.: How to Play any Mental Game—A Completeness Theorem for Protocols with Honest Majority. In: Proceedings of the 19th Annual ACM Symposium on Theory of Computing, pp. 218–229. ACM (1987)
23. Goldreich, O.: Concurrent Zero-Knowledge with Timing, Revisited. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, pp. 332–340. ACM (2002)
24. Hirt, M., Zikas, V.: Adaptively Secure Broadcast. In: Gilbert, H. (ed.) EUROCRYPT 2010, LNCS, vol. 6110, pp. 466–485. Springer, Heidelberg (2010)
25. Hofheinz, D., Müller-Quade, J.: A Synchronous Model for Multi-Party Computation and the Incompleteness of Oblivious Transfer. In: Proceedings of Foundations of Computer Security — FCS'04, pp. 117–130. (2004)
26. Kalai, Y. T., Lindell, Y., Prabhakaran, M.: Concurrent General Composition of Secure Protocols in the Timing Model. In: Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pp. 644–653. ACM (2005)
27. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally Composable Synchronous Computation. In: Cryptology ePrint Archive, Report 2011/310 (2012)
28. Künzler, R., Müller-Quade, J., Raub, D.: Secure Computability of Functions in the IT Setting with Dishonest Majority and Applications to Long-Term Security. In: Reingold, O. (ed.) Theory of Cryptography, LNCS, vol. 5444, pp. 238–255. Springer, Heidelberg (2009)
29. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically Secure Protocols and Security under Composition. In: Proceedings of the 38th Annual ACM Symposium on Theory of Computing, pp. 109–118. ACM (2006)
30. Maji, H., Prabhakaran, M., Rosulek, M.: Cryptographic Complexity Classes and Computational Intractability Assumptions. In: Innovations in Computer Science. Tsinghua University Press (2010)
31. Maurer, U.: Constructive Cryptography: A New Paradigm for Security Definitions and Proofs. In: Mödersheim, S., Palamidessi, C. (eds.) TOSCA, LNCS, vol. 6993, pp. 33–56. Springer, Heidelberg (2011)
32. Maurer, U., Renner, R.: Abstract Cryptography. In: Innovations in Computer Science. Tsinghua University Press (2011)
33. Maurer, U., Tackmann, B.: Synchrony Amplification. In: International Symposium on Information Theory Proceedings, pp. 1583–1587. IEEE (2012)
34. Nielsen, J. B.: On Protocol Security in the Cryptographic Model. PhD thesis, University of Aarhus (2003)
35. Rabin, T., Ben-Or, M.: Verifiable Secret Sharing and Multiparty Protocols with Honest Majority. In: Proceedings of the 21st Annual ACM Symposium on Theory of Computing, pp. 73–85. ACM (1989)