# Security in Asynchronous Interactive Systems[⋆] (Invited paper)

Ivan Geffner[1] [0000−0001−6900−2109] and Joseph Y. Halpern[1] [0000−0002−9229−1663]

Cornell University, Ithaca NY 14850, USA

**Abstract.** Secure function computation has been thoroughly studied and optimized in the past decades. We extend techniques used for secure computation to simulate arbitrary protocols involving a mediator. The key feature of our notion of simulation is that it is bidirectional: not only does the simulation produce only outputs that could happen in the original protocol, but the simulation produces all such outputs. In asynchronous systems there are also new subtleties that arise because the scheduler can influence the output. Thus, these requirements cannot be achieved by the standard notion of secure computation. We provide a construction that is secure if $n > 4t$, where $t$ is the number of malicious agents, which is provably the best possible. We also show that our construction is secure in the *universal composability* model and that it satisfies additional security properties even if $3t < n \leq 4t$.

## 1 Introduction

In a distributed system, agents often want to be able to carry out a computation without revealing any private information. There has been a great deal of work showing how and to what extent this can be done. We briefly review the most relevant work here.

Ben-Or, Goldwasser and Widgerson [3] (BGW from now on) and Chaum, Crépeau, and Damgard [8] showed that, if $n > 3t$, then every function $f$ of $n$ inputs can be $t$-securely computed by $n$ agents in a synchronous system with private communication channels, where "$t$-securely computed" means that no coalition of at most $t$ malicious agents can either (a) prevent the honest agents from correctly computing the output of $f$ given their inputs (assuming some fixed inputs for malicious agents who do not provide inputs) or (b) learn anything about the inputs of the honest agents (beyond what can be concluded from the output of $f$). The notion of an agent "not learning anything" is formalized by comparing what happens in the actual computation to what could have happened had there been a trusted third party (which we here call a *mediator*) who will calculate $f(x_1, \ldots, x_n)$ after being given the input $x_i$ by agent $i$, for $i = 1, \ldots, n$. Then, roughly speaking, the malicious agents do not learn

⋆ Supported in part by NSF grants IIS-1703846 and IIS-0911036, ARO grant W911NF-17-1-0592, MURI grant W911NF-19-1-0217 from the ARO, and a grant from Open Philanthropy.

anything if the distribution of outputs in the actual computation could have also resulted in the computation with a mediator if the malicious agents had given the appropriate input to the mediator.

Ben-Or, Canetti and Goldreich [2] (BCG from now on) proved analogous results in the asynchronous case. Asynchrony raises new subtleties. For example, agent $i$ cannot tell if the fact that he has received no messages from another agent $j$ (which means that $i$ cannot use $j$'s input in computing $f$) is due to the fact that $j$ is malicious or that its messages have not yet arrived. Roughly speaking, when defining secure function computation in an asynchronous setting, BCG require that for every scheduler $\sigma_e$ and set $T$ of malicious agents, no matter what the agents in $T$ do, the resulting distribution over outputs could have also resulted in the computation with a mediator if the malicious agents had given the appropriate input to the mediator.

BCG show that, in asynchronous systems, if $n > 4t$, the malicious agents cannot prevent the honest agents from correctly computing the output of $f$ given their inputs, nor can the malicious agents learn anything about the inputs of the honest agents. Ben-Or, Kelmer, and Rabin [4] (BKR from now on) then showed if we are willing to tolerate a small probability $\epsilon > 0$ that the agents do not correctly compute $f$ or that the malicious agents learn something, then we can achieve this if $n > 3t$. BCG and BKR also prove matching lower bounds for their results, showing that we really need to have $n > 4t$ (resp., $n > 3t$).

We can view secure function computation as a one-round interaction with a trusted mediator: each agent sends its input to the mediator, the mediator waits until it receives enough inputs, applies $f$ to these inputs (again, replacing missing inputs with a default value), and sends the output back to the agents, who then output it. We generalize BCG and BKR's results for function computation to a more general setting. Specifically, we want to simulate arbitrary interactions with a mediator, not just function computation. Also, unlike previous approaches, we want the simulation to be "bidirectional": the set of possible output distributions that arise with the mediator must be the same as those that arise without the mediator, even in the presence of malicious parties. More precisely, we show that, given a protocol profile $\boldsymbol{\pi}$ for $n$ agents[1] and a protocol $\pi_d$ for a mediator, we can construct a protocol profile $\boldsymbol{\pi}'$ such that for all sets $T$ of fewer than $n/4$ malicious agents, the following properties hold:

(a) For all protocols $\boldsymbol{\tau}'_T$ for the malicious agents and all schedulers $\sigma'_e$ in the setting without the mediator, there exists a protocol $\boldsymbol{\tau}_T$ for the agents in $T$ and a scheduler $\sigma_e$ in the setting with the mediator such that, for all input profiles $\boldsymbol{x}$, the output distribution in the computation with $\boldsymbol{\pi}'$, $\boldsymbol{\tau}'$, and $\sigma'_e$ with input $\boldsymbol{x}$ is the same as the output distribution with $\boldsymbol{\pi} + \pi_d$, $\boldsymbol{\tau}$, and $\sigma_e$ with input $\boldsymbol{x}$.

[1] In the economics literature, the term "profile" is used to denote a tuple, so, for example, a protocol profile is a tuple of protocols, one for each agent. In this paper, we refer to protocol profiles as just "protocols", as is standard in the distributed computing literature.

(b) For all protocols $\boldsymbol{\tau}_T$ for the malicious agents and all schedulers $\sigma_e$ in the setting with the mediator, there exists a protocol $\boldsymbol{\tau}'_T$ for the agents in $T$ and a scheduler $\sigma'_e$ in the setting without a mediator such that, for all input profiles $\boldsymbol{x}$, the output distribution in the computation with $\boldsymbol{\pi}'$, $\boldsymbol{\tau}'$, and $\sigma'_e$ with input $\boldsymbol{x}$ is the same as the output distribution with $\boldsymbol{\pi} + \pi_d$, $\boldsymbol{\tau}$, and $\sigma_e$ with input $\boldsymbol{x}$.

We use the notation $\boldsymbol{\pi} + \pi_d$ to indicate that the agents use protocol $\boldsymbol{\pi}$ and the mediator uses protocol $\pi_d$ (we use the subscript $d$ to denote the mediator); we view the mediator as just another agent here. This result implies that arbitrary distributed protocols that work in the presence of a trusted mediator can be compiled to protocols that work without a mediator, as long as there are less than $n/4$ malicious agents. And, just as BKR, if we allow a probability $\epsilon$ of error, we can get this result while tolerating up to $n/3$ malicious agents. BCG proved the analogue of (a) for secure function computation, which is enough for security purposes: if there is any bad behavior in the protocol without the mediator, this bad behavior must already exist in the protocol with the mediator. However, (b) also seems like a natural requirement; if a protocol satisfies this property, then all behaviors in the protocol with the mediator also occur in the protocol without the mediator.

Property (b) is typically not required in security papers. It plays a critical role in our work on implementing mediators [1], but we believe it of independent interest. Requiring only (a) may result in protocols where outcomes that may be likely in the mediator setting do not arise at all. This is especially relevant in asynchronous systems, since by requiring only (a) we are implicitly assuming that the adversary has total control over the scheduler. However, it may be the case that the scheduler acts randomly or that is even influenced by honest agents. For instance, suppose that a group of $n$ agents wants to check who has the fastest internet connection. To do this, each agent pings the server and waits for the server's response. The server (who we are viewing as the mediator) waits until the first ping arrives, then sends a message to each agent saying which agent's ping was received first. In this example, the scheduler determines the lag in the system. If we wanted to simulate this interaction without the mediator but requiring only property (a), even with no malicious agents, a protocol profile in which every honest agent does nothing and outputs 1 would suffice. But, intuitively, this implementation does not capture the behavior of the server. Similar examples exist even in the case of function evaluation. Suppose a group of $n$ congressmen vote remotely (by sending a vote to a trusted third party) to either pass or not a bill that requires support from at least 90% of them. We can view this as a multiparty computation of a function $f$ in which each agent has input 0 (vote against) or 1 (vote for), and the output is either 0 (reject the bill) or 1 (pass the bill) depending on how many agents had input 1 (agents that do not submit input count as 0). In this case, a protocol in which every agent does nothing and outputs 0 securely computes $f$ while tolerating up to $n/4$ malicious agents. To see this, note that regardless of the adversary, the scheduler can delay $n/4$ of the players until everyone else has finished the computation. This is indistinguishable from $n/4$ agents deviating from the protocol and submitting no input. However, again, this protocol does not capture the intended behavior of the voting process. By way of contrast, a protocol that bisimulates $f$ would come closer to capturing the intended behavior of the voting process.

Clearly, the results of BCG and BKR are special cases of our result. However, in general, our results do not follow from those of BCG/BKR, as is shown in Section 3.2. Specifically, the results of BCG/BKR do not give us property (b), since the outcome can depend on the behavior of the scheduler. For example, consider protocols for two agents and a mediator $m$ in which each agent sends its input to the mediator, the mediator $m$ sends to each agent the first message it receives, and each agent outputs whatever they receive from the mediator. Let $\sigma_e^i$ be the scheduler that delivers the message from agent $i$ first, for $i = 1, 2$. It is easy to check that if the agents have inputs 0 and 1, respectively, and play with mediator $\sigma_e^1$, then they both output 0, while if they play with $\sigma_e^2$, then they both output 1. This means that, unlike secure function computation, even if all the agents are honest, the distribution over the agents' outputs can depend on the scheduler's protocol, not just the agents' inputs.

Even though our results do not follow from those of BCG/BKR, our proofs very much follow the lines of those of BCG/BKR. However, there are some new subtleties that arise in our setting. In particular, as the example above shows, when we try to implement the setting with the mediator, the agents must somehow keep track of the scheduler's possible behaviors. Doing this adds nontrivial complexity to our argument. We also show that our construction satisfies an analogue of (a) and (b) in the *universal composability* framework [7], which intuitively means that if a set of agents runs a distributed protocol that requires calls to a subroutine that can be implemented with a mediator, then the agents can implement that subroutine using our construction instead (with no need of a mediator), and the resulting protocol would preserve its original security properties.

Besides the main result, we also show that our protocol without the mediator has two additional security properties, which may be of independent interest. Specifically, we show that the following two properties hold for coalitions of malicious agents of size at most $t < n/3$.

(P1) The only way malicious agents can disrupt the computation is by preventing honest agents from terminating; if an honest agent terminates, then its output is correct.

(P2) If $2t + 1$ or more honest agents terminate, then all honest agents terminate. That is, either all the honest agents terminate or a nontrivial number of honest agents (more than $n - 2t$) do not terminate.

If we allow an $\epsilon$ probability of error, we get analogous results if we have $n > 2t$ rather than $n > 3t$. We remark that these two properties are in fact also satisfied by BCG's and BKR's implementations, but they do not prove this (or even state the properties explicitly).

Our interest in these properties stems in part from a game-theoretic variant of the problem that is considered by Abraham et al. [1], where agents get utility for various outcomes, and, in addition to honest and malicious agents, there are *rational agents*, who will deviate from a protocol if (and only if) it is in their interest to do so. We also assume that honest agents can leave "wills", so that if sufficiently many honest agents do not terminate, the remaining agents will be punished. Property P2 guarantees that either all the honest agents terminate, or sufficiently many of them do not terminate so as to guarantee that rational agents will not try to prevent honest agents from terminating (due to the threat of punishment). Property P1 guarantees that if all the honest agents terminate, their output will be correct. Thus, using these results allows us to obtain results stronger than those of this paper in the game-theoretic setting.

The focus of this paper is on upper bounds. Since our algorithms have the same upper bounds as those of BCG and BKR, despite the results of BCG and BKR being special cases of our results, and BCG and BKR prove lower bounds that match their upper bonds on the number of malicious agents that can be tolerated, we immediately get lower bounds that match our upper bounds from the results of BCG and BKR.

## 2 The Model

The model used throughout this paper is that of an asynchronous network in which every pair of agents can communicate through a private and reliable communication channel. For most of our results, we assume that all messages sent through any of these channels are eventually received, but they can be delayed arbitrarily. The order in which these messages are received is determined by the *environment* (also called the *scheduler*), which is an adversarial entity. The scheduler also chooses the order in which the agents are scheduled. For some of the results of this paper, we drop the condition that all messages must be eventually delivered. We call these more general schedulers *relaxed schedulers.*

Whenever an agent is scheduled, it reads all the messages that it has received since the last time it was scheduled, sends a (possibly empty) sequence of messages, and then performs some internal actions. We assume that the scheduler does not deliver any message or schedule other agents during an agent's turn. Thus, although agent $i$ does not send all its messages simultaneously when it is scheduled, they are sent atomically, in the sense that no other agent is scheduled while $i$ is scheduled, nor are any messages delivered while $i$ is scheduled. Note that the atomicity assumption is really a constraint on the scheduler's protocol.

More precisely, consider the following types of *events*:

- $sch(i)$: Agent $i$ gets scheduled.
- $snd(\mu, j, i)$: Agent $i$ sends a message $\mu$ to agent $j$.
- $rec(\mu, j, i)$: Message $\mu$ sent by $j$ is received by $i$. The message $\mu$ must be one sent at an earlier time to $i$ that was not already received.
- $comp(v, i)$: Agent $i$ locally computes value $v$.
- $out(s, i)$: Agent $i$ outputs string $s$.
- $done(i)$: $i$ is done sending messages and performing computations (for now).

For simplicity, we assume that agents can output only strings in $\{0,1\}^*$. Note that all countable sets can be encoded by such strings, and thus we can freely talk about agents being able to output any element of any countable set (for instance, elements of a finite field $\mathbb{F}_q$) by assuming that they are actually outputting an encoding of these elements. We also assume that at most one event occurs at each time step. Let $h(m)$ denote a *global view* up to time $m$: a sequence that starts with an input profile $\boldsymbol{x}$, followed by the ordered sequence of events that have occurred up to and including time $m$. We assume that the only events between events of the form $sch(i)$ and $done(i)$ are ones of the form $snd(\mu, j, i)$ and $comp(v, i)$. This captures our atomicity assumption. We do not include explicit events that correspond to reading messages. (Nothing would change if we included them; they would simply clutter the notation.) Message delivery (which is assumed to be under the control of the scheduler) occurs at times between when agents are scheduled. We can also consider the subsequence involving agent $i$, namely, $i$'s initial state, followed by events of the form $sch(i)$, $snd(\cdot,\cdot,i)$, $comp(\cdot,i)$, $rec(\cdot,\cdot,i)$, and $done(i)$. This subsequence is called $i$'s *local view* . We drop the argument $m$ if it can be deduced from context or if it is not relevant (for instance, when we consider the local view of an agent after a particular event).

Agent $i$ moves only after a $sch(i)$ event. What it does (in particular, the order in which $i$ sends messages) is determined by $i$'s protocol, which is a function of $i$'s local view. The scheduler moves after an action of the form $done(i)$ or $rec(\cdot,\cdot,i)$. It is convenient to assume that the scheduler is also running a protocol, which is also a function of its local view. Since the scheduler does not see the contents of messages, we can take its view to be identical to $h(m)$, except that *comp* events and the contents of the messages in *snd* and *rec* events are removed, although we do track the index of the messages delivered; that is, we replace events of the form $snd(\mu, i, j)$ and $rec(\mu, i, j)$ by $snd(i, j)$ and $rec(i, j, \ell)$, where $\ell$ is the index of the message sent by $i$ to $j$ in $h(m)$. For instance, $rec(i, j, 2)$ means that the second message sent by $i$ to $j$ was delivered to $j$. Note that the scheduler does see events of the form $done(i)$; indeed, these are signals to the scheduler that it can move, since $i$'s turn is over. Since we view the agents (and the mediator) as sending messages atomically, in the sequel, we talk about an agent's (or the mediator's) *turn.* An agent's $k$th turn takes place the $k$th time it is scheduled. During its turn, the agent sends a block of messages and performs some local computation.

It is more standard in the literature to assume that agents perform at most one action when they are scheduled. We can view this as a constraint on agents' protocols. A *single-action* protocol for agent $i$ is one where agent $i$ sends at most one message before performing the $done(i)$ action. As we show in Appendix 3.6, we could have restricted to single-action protocols with no loss of generality as far as our results go; allowing agents to perform a sequence of actions atomically just makes the exposition easier.

Even though it might appear that malicious agents and the scheduler act independently, it is shown by Abraham et al. [1, Section A.1] that we can assume without loss of generality that they can coordinate their actions, even when there is no direct communication channel between them. In fact, we can assume without loss of generality that they are all under the control of a single entity that is aware of all their local views at all times. We call this entity the *adversary.*

**Definition 1.** *An* adversary *is a triple* $(T, \boldsymbol{\sigma}_T, \tau_e)$*, consisting of a set* $T$ *of malicious agents, the protocol* $\boldsymbol{\tau}_T$ *used by the agents in* $T$*, and a protocol* $\sigma_e$ *for the scheduler. An adversary where the scheduler is relaxed is a* relaxed adversary*.*

In this paper, we consider protocols that involve a *mediator*, typically denoted $d$, using a protocol denoted $\pi_d$. In protocols that involve a mediator, we assume that honest agents' protocols are always such that the honest agents communicate only with the mediator and not with each other, as opposed to malicious agents that can do both. As far as the scheduler is concerned, the mediator is like any other agent, so the scheduler (and the mediator's protocol) determine when the mediator sends and receives messages. However, the mediator is never malicious, and thus never deviates from its announced protocol.

We deal only with bounded protocols, where there is a bound $N$ on the number of messages that an honest agent sends. Of course, there is nothing to prevent malicious agents from spamming the mediator and sending an arbitrary number of messages. We assume that the mediator reads at most $N$ messages from each agent $i$, ignoring any further messages sent by $i$.

For our results involving termination, specifically, (P2), it is critical that agents know when the mediator stops sending messages. For these results, we restrict the honest agents and the mediator to using protocols that have the following *canonical form*: Using a canonical protocol, each honest agent tags its $\ell$th message with label $\ell$ and all honest agents are guaranteed to send at most $N$ messages regardless of their inputs or the random bits they use. Whenever the mediator receives a message from an agent $i$, it checks its tag $\ell$; if $\ell > N$ or if the mediator has already received a message from $i$ with tag $\ell$, it ignores the message. The mediator is guaranteed to eventually terminate. Whenever this happens, it sends a special "STOP" message to all agents and halts. Whenever an honest agent receives a "STOP" message, it terminates.

Even though canonical protocols have a bound $N$ on the number of messages that honest agents and the mediator can send, the mediator's local view in a canonical protocol can be arbitrarily long, since it can be scheduled an arbitrary number of times. We conjecture that, in general, since the message space is finite, the expected number of messages required to simulate the mediator is unbounded. However, we can do better if the mediator's protocol satisfies two additional properties. Roughly speaking, the first property says that the mediator can send messages only either at its first turn or in response to an agent's message; the second property says that the mediator ignores *empty turns*, that is, turns where it does not receive or send messages. More precisely, the first property says that whenever the mediator $\pi_d$ is scheduled with view $h_d$, then if $h_d \neq (\,)$ (i.e., if $h_d$ is not the initial view) or if the mediator has not received any messages in $h_d$ since the last time it was scheduled, then $\pi_d(h_d) = done(d)$. The second property says that $\pi_d(h_d) = \pi_d(h'_d)$, where $h'_d$ is the result of removing consecutive $(\mathit{done}(d), \mathit{sch}(d))$ pairs in $h_d$ (e.g., if $h_d = (\mathit{sch}(d), \mathit{snd}(\mu, j, d), \mathit{done}(d), \mathit{sch}(d), \mathit{done}(d), \mathit{rec}(\mu', i, d), \mathit{sch}(d), \mathit{done}(d), \mathit{sch}(d))$, then $h'_d = (\mathit{sch}(d), \mathit{snd}(\mu, j, d), \mathit{done}(d), \mathit{rec}(\mu', i, d), \mathit{sch}(d)))$. A protocol for the mediator that satisfies these two properties is called *responsive.* In the full paper [**?**, Section 4.4], we show that if the mediator uses a responsive protocol $\pi_d$ that can be represented using a circuit with $c$ gates, then we can simulate all protocol profiles $\boldsymbol{\pi} + \pi_d$ in such a way that the expected number of messages sent by honest agents during the simulation is polynomial in $n$ and $N$ and linear in $c$.

## 3 Secure Computation in Interactive Settings

In this section, we present the main results of this paper and show how they extend and generalize other well-known results.

### 3.1 The BGW/BCG notion of secure computation

*Secure computation* is concerned with jointly computing a function $f$ on $n$ variables, where the $i$th input is known only to agent $i$. For instance, if we want to compute the average salary of the people from the state of New York, then $n$ would be New York's population, the input $x_i$ is $i$'s salary, and $f(x_1, \ldots, x_n) = \frac{\sum_{i=1}^n x_i}{\sum_{x_i \neq 0} 1}$. (For the denominator we count only people who are actually working.) Ideally, a secure computation protocol that computes $f$ would be a protocol in which each agent $i$ outputs $f(x_1, \ldots, x_n)$ and gains no information about the inputs $x_j$ for $j \neq i$. In our example, this amounts to not learning other people's salaries.

Typically, we are interested in performing secure computation in a setting where some of the agents might be malicious and not follow the protocol. In particular, they might not give any information about their input or might just pretend that they have a different input (for instance, they can lie about their salary). What output do we want the secure computation of $f$ to produce in this case? To make precise what we want, we use notation introduced by BGW and BCG.

Let $\boldsymbol{x}$ be a vector of $n$ components; let $C$ be a subset of $[n]$ (where we use the notation $[n]$ to denote the set $\{1, \ldots, n\}$, as is standard); let $\boldsymbol{x}_C$ denote the vector obtained by projecting $\boldsymbol{x}$ onto the indices of $C$; and if $\boldsymbol{z}$ is a vector of length $|C|$, let $\boldsymbol{x}/_{(C,\boldsymbol{z})}$ denote the vector obtained by replacing the entries of $x$ indexed by $C$ with $\boldsymbol{z}$. Given a set $C$ of indices, a default value, which we take here to be 0, and a function $f$, we take $f_C$ to be the function results from applying $f$, but taking the inputs of the agents not in $C$ to be 0; that is, $f_C(\boldsymbol{x}) = f(\boldsymbol{x}/_{(\overline{C},\mathbf{0})})$. Roughly speaking, if only the agents in $C$ provide inputs, we want the output of the secure computation to be $f_c(\boldsymbol{x})$.

What about agents who lie about their inputs? A malicious agent $i$ who lies about his input $x_i$ and pretends to have some other input $y_i$ is indistinguishable from an honest agent who has $y_i$ as his actual input. We can capture this lie using a function $L : D^{|T|} \to D^{|T|}$, where $D$ is the domain of the inputs and $T$ is the set of malicious agents. The function $L$ encodes the inputs malicious agents pretend to have given their actual inputs. BCG require that all the honest agent output the same value and that the output has the form $(C, f_C(\boldsymbol{y}))$, where $\boldsymbol{y} = \boldsymbol{x}/_{(T,L(\boldsymbol{x}_T))}$. They allow $C$ to depend on $\boldsymbol{x}_T$, since malicious agents can influence the choice of $C$. They also allow the choice of $C$ and the function $L$ to be randomized. Since the choice of $L$ and $C$ can be correlated, $L$ and $C$ are assumed to take as input a common random value $r \in \mathcal{R}$, where $\mathcal{R}$ denotes the domain of random inputs. That is, $C = c(\boldsymbol{x}_T, r)$ for some function $c$, and the malicious agents with actual input $\boldsymbol{x}_T$ pretend that their input is $L(\boldsymbol{x}_T, r)$.

BCG place no requirements on the output of malicious agents, but they do want the inputs of honest agents to remain as secret as possible. Hence, in an ideal scenario, the outputs of malicious agents can depend only on $\boldsymbol{x}_T$, $f_C(\boldsymbol{y})$, and possibly some randomization. Taking $O_i$ to denote the output function of a malicious agent $i$, we can now give BCG's definitions.

**Definition 2.** *An* ideal $t$-adversary $A$ *is a tuple* $(T, c, L, \boldsymbol{O})$ *consisting of a set* $T \subseteq [n]$ *of malicious agents with* $|T| \leq t$ *and three randomized functions* $c : D^{|T|} \times \mathcal{R} \to \mathcal{P}([n])$ *with* $|c(\boldsymbol{z}, r)| \geq n - t$ *for all input profiles* $\boldsymbol{z}$ *and* $r$, $L : D^{|T|} \times \mathcal{R} \to D^{|T|}$ *and* $\boldsymbol{O} : D^{|T|} \times D \times \mathcal{R} \to (\{0,1\}^*)^{|T|}$. *The* ideal *output* $\boldsymbol{\rho}$ *of* $A$ *given function* $f$, *input profile* $\boldsymbol{x}$, *and a value* $r \in \mathcal{R}$ *is*

$$\rho_i(\boldsymbol{x}, A, r; f) = \begin{cases} (c(\boldsymbol{x}_T, r), f_{c(\boldsymbol{x}_T,r)}(\boldsymbol{x}/_{(T,L(\boldsymbol{x}_T,r))})) & \text{if } i \notin T \\ O_i(\boldsymbol{x}_T, f_{c(\boldsymbol{x}_T,r)}(\boldsymbol{x}/_{(T,L(\boldsymbol{x}_T,r))}), r) & \text{if } i \in T. \end{cases}$$

Note that an ideal $t$-adversary is somewhat different from the adversary as defined in Definition 1, although they are related, as we show in Section 3.2. We use variants of $A$ to denote both types of adversary.

Let $\boldsymbol{\rho}(\boldsymbol{x}, A; f)$ denote the distribution induced over outputs by the protocol profile $\boldsymbol{\rho}$ on input $x$ given the ideal $t$-adversary $A$. We can now give the BCG definition of secure computation. Let $\boldsymbol{\pi}(\boldsymbol{x}, A)$ be the distribution of outputs when running protocol $\boldsymbol{\pi}$ on input $\boldsymbol{x}$ with adversary $A = (T, \boldsymbol{\tau}_T, \sigma_e)$.

**Definition 3 (Secure computation).** *Let* $f : \mathcal{D}^n \to \mathcal{D}$ *be a function on* $n$ *variables and* $\boldsymbol{\pi}$ *a protocol for* $n$ *agents. Protocol* $\boldsymbol{\pi}$ $t$*-securely computes* $f$ *if, for every adversary* $A = (T, \boldsymbol{\tau}_T, \sigma_e)$, *the following properties hold:*

*SC1. For all input profiles* $\boldsymbol{x}$, *all honest agents terminate with probability 1.*
*SC2. There exists an ideal* $t$*-adversary* $A' = (T, c, L, \boldsymbol{O})$ *such that, for all input profiles* $\boldsymbol{x}$, $\boldsymbol{\rho}(\boldsymbol{x}, A'; f)$ *and* $\boldsymbol{\pi}(\boldsymbol{x}, A)$ *are identically distributed.*

Note that BCG just require that *some* ideal $t$-adversary $A$ gives the same distribution over the outputs of $\boldsymbol{\pi}$. This captures the idea that all ways that malicious agents can deviate are modeled by adversaries. Also note that SC1 follows from SC2 if we view non-termination as a special kind of output.

BCG prove the following result:

**Theorem 1 (BCG).** *Given* $n$ *and* $t$ *such that* $n > 4t$ *and a function* $f : D^n \to D$, *there exists a protocol* $\boldsymbol{\pi}^f$ *that* $t$*-securely computes* $f$.

The construction of $\boldsymbol{\pi}^f$ is sketched in [2] and [**?**, Section 3.2.7]; most of the primitives used in this construction are also used in ours.

## 3.2 Secure computation and mediators

Even though it is not explicitly proven by BCG, their construction of $\boldsymbol{\pi}^f$ satisfies an additional property that we call SC3, which is essentially a converse of SC2.

SC3. For all ideal $t$-adversaries $A = (T, c, L, \boldsymbol{O})$, there exists an adversary $A' = (T, \boldsymbol{\tau}_T, \sigma_e)$ such that, for all input profiles $\boldsymbol{x}$, $\boldsymbol{\rho}(\boldsymbol{x}, A; f)$ and $\boldsymbol{\pi}(\boldsymbol{x}, A')$ are identically distributed.

**Lemma 1.** *Given a function $f : D^n \to D$, protocol $\boldsymbol{\pi}^f$ satisfies SC3.*

*Proof (Proof (sketch).).* Given a trusted-party adversary $A = \{T, c, L, \boldsymbol{O}\}$ and an input profile $\boldsymbol{x}_T$, the adversary $A'$ runs $\boldsymbol{\pi}^f_T$ with input $L(\boldsymbol{x}_T)$, except that if a malicious agent $i$ would output a tuple of the form $(S, z)$ (note that all outputs of honest players have this form), it outputs $O_i(\boldsymbol{x}_T, z)$ instead. Meanwhile, the scheduler delays all messages from agents not in $c(\boldsymbol{x}_T)$ until all honest players finish their part of the computation. We can show that the outputs of $A$ and $A'$ are identically distributed. Since the full proof requires the actual implementation of $\boldsymbol{\pi}^f$, the details are given in [**?**, Section 3.3]

We next show how secure computation relates to simulating a mediator. Consider the following protocol $\boldsymbol{\tau}^f + \tau^f_d$ for $n$ agents and a mediator: Agents send their inputs to the mediator the first time that they are scheduled. The mediator waits until it has received a valid input from all agents in a subset $C \subseteq [n]$ with $|C| \geq n - t$. The mediator then computes $y = f_C(\boldsymbol{x})$ and sends each agent the pair $(C, y)$. When the agents receive a message from the mediator, they output that message and terminate.

Clearly $\boldsymbol{\tau}^f + \tau^f_d$ satisfies SC1. It is easy to see that it also satisfies SC2: Given a set $T$ of malicious agents, a deterministic protocol profile $\boldsymbol{\tau}_T$ for the malicious agents, and a deterministic scheduler $\sigma_e$, define $L(\boldsymbol{x}, r)$ to be whatever the malicious agents send to the mediator with input $\boldsymbol{x}$, let $c(\boldsymbol{x})$ be the set of agents from whom the mediator has received a message the first time it is scheduled after having received a message from a least $n - t$ agents (given $\sigma_e$, $\boldsymbol{\tau}_T$, and input $\boldsymbol{x}$), and let $O(\boldsymbol{x})$ be the output function that malicious agents use in $\boldsymbol{\tau}^f + \tau^f_d$ (note that they receive a single message with the output of the computation, so their output depends only on $\boldsymbol{x}$, $\boldsymbol{\tau}_T$, and $\sigma_e$). Clearly SC2 holds with this choice of $t$-ideal adversary. Randomized functions $\boldsymbol{\tau}_T$ and $\sigma_e$ can be viewed as resulting from sampling random bits $r$ according to some distribution and then running deterministically; the protocols $c$, $h$, and $O$ can sample $r$ from the same distribution and then proceed as above with respect to the deterministic $\boldsymbol{\tau}_T(r)$ and $\sigma_e(r)$.

The protocol $\boldsymbol{\tau}^f + \tau^f_d$ satisfies SC3 as well. Given $A = (T, c, L, O)$, the definition of $\boldsymbol{\tau}_T$ and $\sigma_e$ is straightforward: the agents in $T$ choose a random input $r \in \mathcal{R}$ and then each agent $i \in T$ sends $L(x_i, r)$ to the mediator. The scheduler $\sigma_e$ delivers all messages from the agents in $c(\boldsymbol{x}_T, r)$ first, and then schedules the mediator. It then delivers all the other messages.

Since both $\boldsymbol{\tau}^f + \tau_d^f$ and $\boldsymbol{\pi}^f$ satisfy SC2 and SC3, for all adversaries $A$, there exists an adversary $A'$ (resp., for all adversaries $A'$ there exists an adversary $A$) such that $(\boldsymbol{\tau}^f + \tau_d^f)(\boldsymbol{x}, A)$ and $\boldsymbol{\pi}^f(\boldsymbol{x}, A')$ are identically distributed.

Unfortunately, given a protocol $\boldsymbol{\pi}_d$ for the mediator, there might not exist a function $f$ such that SC2 and SC3 hold, as the example given in the introduction shows (where the mediator sends to the agents the first message it receives). Note that, in this example, the output of the agents is not a function of their input profile; thus, there is no function $f$ for which SC2 and SC3 hold. Nevertheless, we are still interested in securely computing the output of the protocol with the mediator. That is, we are interested in getting analogues to SC2 and SC3 for arbitrary interactive protocols. This is captured by the following definition:

**Definition 4.** *Protocol $\boldsymbol{\pi}'$ $t$-*bisimulates *$\boldsymbol{\pi}$ if the following two properties hold:*

*(a) For all adversaries $A = (T, \boldsymbol{\tau}_T, \sigma_e)$ with $|T| \leq t$, there exists an adversary $A' = (T, \boldsymbol{\tau}'_T, \sigma'_e)$ such that for all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, A)$ and $\boldsymbol{\pi}'(\boldsymbol{x}, A')$ are identically distributed.*

*(b) For all adversaries $A' = (T, \boldsymbol{\tau}'_T, \sigma'_e)$ with $|T| \leq t$, there exists an adversary $A = (T, \boldsymbol{\tau}_T, \sigma_e)$ such that all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, A)$ and $\boldsymbol{\pi}'(\boldsymbol{x}, A')$ are identically distributed.*

Note that the first clause is analogous to SC2, while the second clause is analogous to SC3. There is no clause analogous to SC1 since we allow agents not to terminate. In any case, since we can view non-termination as a special type of output (i.e., we can view an agent that does not terminate as outputting $\perp$), so SC2 already guarantees that non-termination happens with the same probability in $\boldsymbol{\pi}'$ and $\boldsymbol{\pi}$ (In the setting of BGW, since all functions terminate, with this viewpoint, SC2 implies SC1, a point already made by Canetti [6].)

The following proposition follows from Theorem 1 and Lemma 1.

**Proposition 1.** *$\boldsymbol{\pi}^f$ $t$-bisimulates $\boldsymbol{\tau}^f + \tau_d^f$ if $n > 4t$.*

## 3.3 Beyond secure computation

Although BCG make claims for their protocol only if $n > 4t$, variants of some of the properties that they are interested in continue to hold even if $n < 4t$. The first of these properties is that if $n > 3t$, then the only way that the adversary can affect $\boldsymbol{\pi}^f$ is by preventing some honest agents from terminating. We can capture this notion as follows.

**Definition 5.** *A scheduler is* relaxed *if it can decide not to deliver some of the messages. A protocol $\boldsymbol{\pi}'$ $(t, t')$-*bisimulates *$\boldsymbol{\pi}$ if it $t$-bisimulates $\boldsymbol{\pi}$ but the schedulers $\sigma'_e$ and $\sigma_e$ of the first and second clause of Definition 4 respectively may be relaxed for $t \geq |T| > t'$.*

**Proposition 2.** *$\boldsymbol{\pi}^f$ $(t, t')$-bisimulates $\boldsymbol{\tau}^f + \tau_d$ and $t \geq t'$.*

This means that if $3t + t' < n$, then adversaries of size between $t'$ and $t$ have the same power to affect the outcome with $\boldsymbol{\pi}^f$ as with $\boldsymbol{\tau}^f + \tau_d$ as long as schedulers are allowed to discard messages, so that they never reach their recipient. In particular, this means that the adversary cannot influence the outcome in any other way than by preventing some honest players from terminating. However, we can show that the BCG protocol has the property that if at least $2t+1$ honest agents terminate, then all the remaining honest agents terminate. This observation motivates the following definition:

**Definition 6.** *A protocol $\boldsymbol{\pi}$ $(t,k)$-*coterminates *if, all adversaries $A = (T, \boldsymbol{\tau}_T, \sigma_e)$ with $|T| \le t$ and all input profiles $\boldsymbol{x}$, in all executions of $\boldsymbol{\pi}$ with adversary $A$ and input $\boldsymbol{x}$, either all the agents not in $T$ terminate or strictly fewer than $k$ agents not in $T$ do.*

**Proposition 3.** $\boldsymbol{\pi}^f$ *$(t, 2t+1)$-coterminates.*

We do not prove Proposition 2 or 3 here, since we prove a generalization of them below (see Theorem 2).

## 3.4 Simulating arbitrary protocols

The goal of this paper is to show that we can securely implement any interaction with a mediator, and do so in a way that ensure the two properties discussed in Section 3.3. This is summarized in the following theorem:

**Theorem 2.** *For every protocol $\boldsymbol{\pi} + \pi_d$ for $n$ agents and a mediator, there exists a protocol $\boldsymbol{\pi}'$ for $n$ agents such that $\boldsymbol{\pi}'$*

*(a) $(t,t')$-bisimulates $\boldsymbol{\pi}$ if $n > 3t + t'$ and $t \ge t'$, and*
*(b) $(t, 2t+1)$-coterminates if $n > 3t$ and $\boldsymbol{\pi} + \pi_d$ is in canonical form.*

*Moreover, if $\pi_d$ is responsive, the expected number of messages sent in an execution of $\boldsymbol{\pi}'$ is polynomial in $n$ and $N$, and linear in $c$, where $N$ is the expected number of messages sent when running $\boldsymbol{\pi} + \pi_d$ and $c$ is the number of gates in an arithmetic circuit that implements the mediator's protocol.*

The construction of $\boldsymbol{\pi}'$ is sketched in Section 4 and given in full detail in given in the full paper [**?**, Section 4.2] and, not surprisingly, uses many of the techniques used by BCG. And, like BKR, if we allow an $\epsilon$ probability of error we get stronger results. We define $\epsilon$-$t$-bisimulation just like $t$-bisimulation (Definition 4), except that, in both clauses, the distance between $(\boldsymbol{\pi} + \pi_d)(\boldsymbol{x}, A)$ and $\boldsymbol{\pi}'(\boldsymbol{x}, A')$ is less than $\epsilon$, where the distance $d$ between probability measures $\nu$ and $\nu'$ on some finite space $S$ is defined as $d(\nu, \nu') = \sum_{s \in S} |\nu(s) - \nu'(s)|$. The definition of $\epsilon$-$t$-bisimulation and $\epsilon$-$(t,t')$-bisimulation are analogous. A protocol $\epsilon$-$(t,k)$-*coterminates* if it $(t,k)$-coterminates with probability $1-\epsilon$.

**Theorem 3.** *For every protocol $\boldsymbol{\pi} + \pi_d$ for $n$ agents and a mediator and all $\epsilon > 0$, there exists a protocol $\boldsymbol{\pi}'$ for $n$ agents such that $\boldsymbol{\pi}'$*

*(a)* $\epsilon$-$(t,t')$*-bisimulates* $\boldsymbol{\pi}+\pi_d$ *if* $n>2t+t'$ *and* $t\geq t'$*, and*
*(b)* $\epsilon$-$(t,t+1)$*-coterminates if* $n>2t$ *and* $\boldsymbol{\pi}+\pi_d$ *is in canonical form.*

*Moreover, if* $\pi_d$ *is responsive,* $\boldsymbol{\pi}'$ *can be implemented in such a way that the expected number of messages when running* $\boldsymbol{\pi}'+\pi_d$ *is polynomial in* $n$ *and* $N$*, and linear in* $c$*, where* $N$ *is the expected number of messages sent when running* $\boldsymbol{\pi}+\pi_d$*.*

## 3.5 Universally composable security

Both the definition of secure computation (Definition 3) and of bisimulation (Definition 4) capture only the intended security properties in the *stand-alone* model, where only a single execution of a given protocol is run. However, in many cases, it is important that these properties are satisfied even when a protocol is run several times in succession, or even when these executions are performed concurrently. For this purpose, the standard approach is to prove that the given protocol is secure in the *universal composability model* [7]. Kushilevitz, Lindell and Rabin [9] showed that every protocol that is perfectly secure in the stand-alone model and has a *straight-line black-box simulator* is also secure in the UC model. Having a black-box straight-line simulator means that the adversary is able to simulate what its view would be when running the protocol without the mediator (resp., with the mediator), given its view in the protocol with the mediator (resp., without the mediator), and that it is able to do so without having to *rewind*, which means to go back to a previous state and interact with the other agents in a different way. This is exactly the approach we take when showing that the protocol presented in [**?**, Section 4.2] satisfies the properties of Theorem 2 (see [**?**, Section 4.3] for details). Therefore, Theorem 2 holds with perfect universally composable security as well; that is, if a protocol uses the mediator as a subroutine, and we replace the subroutine with our implementation, then all the desired properties would still hold, even if the protocol is ran concurrently with another protocol.

## 3.6 Variant models

In this section, we show that the choices made in our formal model are essentially being made without loss of generality. We start by considering our assumption that agents perform a sequence of actions atomically when they are scheduled. We next show that we would get theorems equivalent to the ones that we are claiming if we had instead assumed that agents perform just a single action when they are scheduled. To prove this, we first need the following notion:

**Definition 7.** *A protocol* $\boldsymbol{\pi}$ *is* $N$-message bounded *if, for all inputs, no agent ever sends more than* $N$ *messages in a single turn. A protocol is* message bounded *if it is* $N$*-message bounded for some* $N$*.*

**Proposition 4.** *There exist a function* $H$ *from message-bounded protocols to single-action protocols such that for all protocols* $\boldsymbol{\pi}$*, the following holds:*

*(a) For all schedulers (resp., relaxed schedulers) $\sigma_e$ there exists a scheduler (resp., relaxed scheduler) $\sigma_e'$ such that, for all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed, where $H(\boldsymbol{\pi}) := (H(\pi_1), \ldots, H(\pi_n))$ and we view $\sigma_e$ and $\sigma_e'$, respectively, as the adversaries (i.e., we take $T = \emptyset$).*

*(b) For all schedulers (resp., relaxed schedulers) $\sigma_e'$ there exists a scheduler (resp., relaxed scheduler) $\sigma_e$ such that, for all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.*

The converse of Proposition 4 is trivial, since single-action protocols are protocols. It follows from Proposition 4 that Theorem 2 holds even if we restrict agents to using single-action protocols (note that canonical protocols are message bounded).

*Proof.* Intuitively, $H(\pi_i)$ is identical to $\pi_i$, except that rather than sending a sequence of messages when it is scheduled, $i$ sends the messages one at a time. The scheduler $\sigma_e'$ is then chosen to ensure that $i$ is scheduled so that it sends all of its messages as if they were sent atomically. In addition to keeping track of the messages it has sent and received, $i$ uses the variable $U_i$ whose value is a sequence of messages (intuitively, the ones that $i$ would have sent at this point in the simulation of $\pi_i$ that it has not yet sent), initially set to the empty sequence, and a binary variable *next*, originally set to 1. When $i$ is scheduled by $\sigma_e'$, $H(\pi_i)$ proceeds as follows: If $\mathit{next} = 1$, then $i$ sets $U_i$ to the sequence of messages that it would send with $\pi_i$ given its current view. (If $\pi_i$ randomizes, then $H(\pi_i)$ does the same randomization. If $U_i$ is the empty sequence (so $\pi_i$ would not send any messages at that point), $i$ performs the action $\mathit{done}(i)$, and outputs whatever it does with $\pi$; otherwise, $i$ sets *next* to 0, sends the first message in $U_i$ to its intended recipient, and removes this message from $U_i$. If $\mathit{next} = 0$, then if $U_i$ is empty, $i$ sets *next* to 1, sends $\mathit{done}(i)$, and outputs whatever it does with $\pi$; otherwise, $i$ sends the first message in $U_i$ to its intended recipient and removes it from $U_i$.

Since $\boldsymbol{\pi}$ is message bounded, there exists an $N$ such that $\boldsymbol{\pi}$ is $N$-message bounded. For part (a), given $\sigma_e$, we construct $\sigma_e'$ so that it simulates $\sigma_e$, except that if $\sigma_e$ schedules $i$, $\sigma_e'$ schedules $i$ repeatedly until either it observes $\mathit{done}(i)$ or until $i$ sends messages in $N + 1$ consecutive turns. Since $\boldsymbol{\pi}$ is $N$-message bounded, it is clear that $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed. Note that it is necessary for $\boldsymbol{\pi}$ to be $N$-message bounded, since if the scheduler schedules each agent $i$ repeatedly until it stops sending a message during its turn, an agent that keeps sending messages would be scheduled indefinitely, and so would prevent other agents from being scheduled.

For part (b), given $\sigma_e'$, we construct $\sigma_e$ so that it simulates $\sigma_e$. There is one issue that we have to deal with. Whereas with $\sigma_e$, an agent $i$ can send $k$ messages each time it is scheduled, with $\sigma_e'$, it can send only one message when it is scheduled. The scheduler $\sigma_e'$ constructed from $\sigma_e$ in part (a) scheduled $i$ repeatedly until it sent all the messages it did with $\sigma_e$. But we cannot assume that the scheduler $\sigma_e'$ that we are given for part (b) does this. Thus, $\sigma_e$ must keep track of how many of the messages that each agent $i$ was supposed to send the last time it was scheduled by $\sigma_e$ have been sent so far. To do this, $\sigma_e$ uses variables $mes_i$, one for each agent $i$, initially set to 0, such that $mes_i$ keeps track of how many of the messages that agent $i$ sent with $\sigma_e$ still need to be sent by $\sigma_e'$. As we observed above, given a local view $h$ of the scheduler where the agents use $\boldsymbol{\pi}$ and the scheduler uses $\sigma_e$, there is a corresponding local view $h'$ of the scheduler where the agents use $\boldsymbol{\pi}'$ and the scheduler uses $\sigma_e'$. If, given $h'$, $\sigma_e'$ schedules agent $i$ with probability $\alpha_i$, then with the same probability $\alpha_i$, $\sigma_e$ proceeds as follows: if $mes_i = 0$ (which means that all the messages that $i$ sent the last time it was scheduled have been delivered in $h'$), then $\sigma_e$ schedules $i$, sees how many messages $i$ delivers according $\pi_i$, and sets $mes_i$ to this number; if $mes_i \neq 0$, then $mes_i$ is decremented by 1 but no agent is scheduled. Again, it is clear that that $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.

BCG put further constraints on the scheduler. Specifically, they assume that, except possibly for the first time that agent $i$ is scheduled, $i$ is scheduled immediately after receiving a message and only then. That is, in our terminology, BCG assume that a $rec(\cdot,\cdot,i)$ event must be followed by a $sch(i)$ event, and all $sch(i)$ events except possibly the first one occur after a $rec(\cdot,\cdot,i)$ event. We call the schedulers that satisfy this constraint *BCG schedulers*.

We now prove a result analogous to Proposition 4, from which it follows that we could have obtained our results using a BCG scheduler.

**Proposition 5.** *There exist a function $H$ from protocols to protocols such that for all protocols $\boldsymbol{\pi}$ the following holds:*

*(a) For all schedulers (resp., relaxed schedulers) $\sigma_e$ there exists a BCG scheduler (resp., relaxed BCG scheduler) $\sigma_e'$ such that, for all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.*

*(b) For all BCG schedulers (resp., relaxed schedulers) $\sigma_e'$ there exists a scheduler (resp., relaxed scheduler) $\sigma_e$ such that, for all input profiles $\boldsymbol{x}$, $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.*

*Proof.* As in Proposition 4, the idea is that $\sigma_e'$ simulates $\sigma_e$, but since $\sigma_e$ can schedule an agent only when it delivers a message, we have each agent $i$ send itself special messages, denoted $proceed_i$, to ensure that there are always enough messages in the system. In more detail, $H(\pi_i)$ works as follows. When it is first scheduled, agent $i$ sends itself a $proceed_i$ message. Since we are considering BCG schedulers, agent $i$ is scheduled subsequently only when it receives a message. If it receives a message other than $proceed_i$, it does nothing (although the message is added to its view). If it receives a $proceed_i$ message, then it does whatever it would do with $\pi_i$ given its current view with the $proceed_i$ messages and the $sch(i)$ events not preceded by a $proceed_i$ message removed, and sends itself another $proceed_i$ message.

For part (a), given $\sigma_e$, $\sigma_e'$ first schedules each agent once (in some arbitrary order), to ensure that that each of them has sent a $proceed_i$ message that is available to be delivered. Given a view $h'$, $\sigma_e'$ considers what $\sigma_e$ would do in the view $h$ that results from $h'$ by removing the initial $sch(i)$ event for each agent $i$, the last message that each agent $i$ sends when it is scheduled if it sends a message at all, and the receipt of these messages. If $h'$ is a view that results where the agents are running $H(\boldsymbol{\pi})$, then the send and receive events removed are precisely those that involve $\mathit{proceed}_i$. If $\sigma_e$ delivers a message with some probability, then $\sigma_e'$ delivers the corresponding message with the same probability; if $\sigma_e$ schedules an agent $i$ with some probability, $\sigma_e'$ delivers the last $\mathit{proceed}_i$ that $i$ sent and schedules agent $i$ with the same probability. If there is no $\mathit{proceed}_i$ message to deliver, then $\sigma_e'$ does nothing, but our construction of $H(\pi_i)$ guarantees that if $h'$ is a view that results from running $H(\boldsymbol{\pi})$, then there will be such a message that can be delivered. Again, it is clear that $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.

For part (b), given $\sigma_e'$, the construction of $\sigma_e$ is similar to that of Proposition 4. Again, given a local view $h$ of $\sigma_e$ where the agents use $\boldsymbol{\pi}$, there is a corresponding view $h'$ of $\sigma_e'$ where the agents use $H(\boldsymbol{\pi})$. If, given input $h'$, $\sigma_e'$ delivers a message with some probability $p$ and the messages is not a $\mathit{proceed}_i$ message, then $\sigma_e$ delivers the corresponding message with probability $p$. If the message is a $\mathit{proceed}_i$ message, then $\sigma_e$ also schedules agent $i$. If $\sigma_e'$ schedules an agent $i$ with probability $p$, and in $h'$ this is the first time that $i$ is scheduled, then $\sigma_e$ schedules $i$ with probability $p$ and otherwise does nothing with probability $p$. Yet again, it is straightforward to show that $\boldsymbol{\pi}(\boldsymbol{x}, \sigma_e)$ and $H(\boldsymbol{\pi})(\boldsymbol{x}, \sigma_e')$ are identically distributed.

## 4 Proof of Theorem 2

In this section we sketch the construction of a protocol $\boldsymbol{\pi}'$ that satisfies Theorem 2. We provide the full construction and the proof of correctness in the full paper [**?**, Sections 4.3 and 4.5].

The construction uses a number of primitives, some of which were defined by BCG and some of which go back much further. Among other, it uses *secret-sharing*, which goes back to Shamir [10], a broadcast protocol due to Bracha [5], and a *circuit computation* protocol. Recall that with secret sharing, a sender can distribute a secret $s$ (which is just an element of the field $\mathbb{F}_p$) among $n$ agents in such a way that no subset of $t$ agents can guess the value of $s$ with better probability than $1/p$ (the cardinality of $\mathbb{F}_p$), but such that any subset of $t+1$ agents can compute $s$ with no probability of error. This is done by having the sender choose a polynomial $p_s \in \mathbb{F}_p[X]$ of degree $t$ uniformly at random such that $p_s(0) = s$ and sending each agent $i$ $i$'s *share* of $s$, which is $s_i := p_s(i)$. Bracha's broadcast protocol allows a sender to broadcast a message to a group in an asynchronous system so that, if $n > 3t$, all honest players will eventually get it.

The circuit computation protocol [2] allows agents to compute their shares of the output of a circuit $f$ from their shares of the inputs of $f$ without learning anything about the shares of other players.

The construction proceeds as follows. Intuitively, agents simulate $\boldsymbol{\pi} + \pi_d$ by jointly computing the mediator's state, which messages it receives, and which messages it would send to each agent. To keep the mediator's computation secret, instead of computing the mediator's state directly, agents just compute their share of it. Each agent $i$ sends $j$ its share of each of the messages that $i$ sends in the protocol with the mediator; each agent $j$ then uses these shares to update the state of the mediator. Agents use the verifiable secret sharing and circuit computation protocols provided by BCG in order to tolerate malicious behavior when distributing the shares of their messages and when computing the shares of the mediator's state every time it is updated. They also use Bracha's consensus protocol to agree on what messages the mediator receives every time it is scheduled and in which order it does so. The properties of all of these primitives are given in [5], [2], and [**?**, Section 3.2]

While this is the outline of the protocol, there are still a number of subtleties that have to be dealt with. For example, we must decide when each agent and the mediator is scheduled in the simulation of $\boldsymbol{\pi} + \pi_d$, or how agents update the mediator's state. All of these details are provided in the full version [**?**, Section 4.2].

## 5 Conclusion

We have shown how to simulate arbitrary protocols securely in an asynchronous setting in a "bidirectional" way (as formalized by our notion of bisimulation). This bidirectionality plays a key role in the application of these results in [1]; we believe that it might turn out to be useful in other settings as well. While this property holds for BCG's secure computation implementation, proving that we can simulate arbitrary protocols so that it holds seems to be nontrivial.

Our construction may not be message-efficient in the general case. However, for responsive mediators, a small modification (see [**?**, Section 4.4]) allows us to bound the expected number of messages by a function that is polynomial in the number of agents $n$ and the maximum number of messages $N$ sent in the setting with the mediator, and linear in $c$, the number of gates in a circuit that implements the mediator's protocol. It is still an open problem whether all protocols $\boldsymbol{\pi} + \pi_d$ can be implemented in a way that the expected number of messages sent by honest agents is bounded by some function of $n$, $N$, and $c$.

## References

1. Abraham, I., Dolev, D., Geffner, I., Halpern, J.Y.: Implementing mediators with asynchronous cheap talk. In: Proc. 38th ACM Symposium on Principles of Distributed Computing. pp. 501–510 (2019)
2. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: STOC '93: Proceedings of the 25 Annual ACM Symposium on Theory of Computing. pp. 52–61 (1993)
3. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation. In: Proc. 20th ACM Symposium on Theory of Computing. pp. 1–10 (1988)

4. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: Proc. 13th ACM Symp. Principles of Distributed Computing. pp. 183–192 (1994)
5. Bracha, G.: An asynchronous $[(n-1)/3]$-resilient consensus protocol. In: Proc. 3rd ACM Symposium on Principles of Distributed Computing. pp. 154–162 (1984)
6. Canetti, R.: Studies in Secure Multiparty Computation and Applications. Ph.D. thesis, Technion (1996), citeseer.nj.nec.com/canetti95studies.html
7. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proc. 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145 (2001)
8. Chaum, D., Crépeau, C., Damgård, I.: Multi-party unconditionally secure protocols. In: Proc. 20th ACM Symposium on Theory of Computing. pp. 11–19 (1988)
9. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. SIAM Journal on Computing **39**(5), 2090–2112 (2010)
10. Shamir, A.: How to share a secret. Commun. ACM **22**, 612–613 (1979)