Lower Bounds on Implementing Mediators in Asynchronous Systems with Rational and Malicious Agents

IVAN GEFFNER and JOSEPH Y. HALPERN, Cornell University

Abraham, Dolev, Geffner, and Halpern [1] proved that, in asynchronous systems, a (k, t)-*robust equilibrium* for n players and a trusted mediator can be implemented without the mediator as long as $n > 4(k + t)$, where an equilibrium is (k, t)-robust if, roughly speaking, no coalition of t players can decrease the payoff of any of the other players, and no coalition of k players can increase their payoff by deviating. We prove that this bound is tight, in the sense that if $n \leq 4(k+t)$ there exist (k, t)-robust equilibria with a mediator that cannot be implemented by the players alone. Even though implementing (k, t)-robust mediators seems closely related to implementing asynchronous multiparty $(k + t)$-secure computation [6], to the best of our knowledge there is no known straightforward reduction from one problem to another. Nevertheless, we show that there is a non-trivial reduction from a slightly weaker notion of $(k + t)$-secure computation, which we call $(k + t)$-*strict secure computation*, to implementing (k, t)-robust mediators. We prove the desired lower bound by showing that there are functions on n variables that cannot be $(k + t)$-strictly securely computed if $n \leq 4(k + t)$. This also provides a simple alternative proof for the well-known lower bound of $4t + 1$ on asynchronous secure computation in the presence of up to t malicious agents [4, 8, 10].

CCS Concepts: • **Theory of computation** → **Distributed algorithms**; **Solution concepts in game theory**; *Algorithmic mechanism design*; • **Security and privacy** → Mathematical foundations of cryptography;

Additional Key Words and Phrases: Lower bounds, games with communication, multiparty secure computation

ACM Reference format:
Ivan Geffner and Joseph Y. Halpern. 2023. Lower Bounds on Implementing Mediators in Asynchronous Systems with Rational and Malicious Agents. *J. ACM* 70, 2, Article 13 (March 2023), 21 pages.
https://doi.org/10.1145/3578579

1 INTRODUCTION

Ben-Or, Goldwasser, and Wigderson [7] (BGW from now on) showed that given a finite domain D, a function $f : D^n \rightarrow D$ can be t-*securely computed* by n agents in a synchronous network with private authenticated channels as long as $n > 3t$, where t is a bound on the number of malicious agents. Roughly speaking, "t-securely computed" means that all honest agents correctly compute the output of f, while a group of up to t malicious agents can learn nothing about the agents' inputs beyond what can be learned the output of f. Ben-Or, Canetti, and Goldreich [6] (BCG from

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.
Authors' addresses: I. Geffner and J. Y. Halpern, Bill and Melinda Gates Hall. 107 Hoy Rd, Cornell University, Ithaca, NY 14853; emails: ieg8@cornell.edu, halpern@cs.cornell.edu.

0004-5411/2023/03-ART13 $15.00
https://doi.org/10.1145/3578579

now on) later provided analogous results for the asynchronous case: a function $f : D^n \to D$ can be t-securely computed by n agents if $n > 4t$.

Secure function computation is often viewed as an interaction with a trusted third party, or *mediator*. Roughly speaking, we want the outcome to be the same as if the agents had sent their input values $\vec{x}$ to the mediator, who then sends back $f(\vec{x})$. However, the problem of implementing such an interaction while tolerating faulty behavior is approached differently by the distributed computing community and the game theory community. In the distributed computing literature, as in BGW and BCG, the agents controlled by the adversary are malicious and may deviate in any way they want in order to subvert the computation. However, in the game-theory literature, there are no malicious agents; all participating agents are *rational*, which means that they deviate if and only if they can increase their expected payoff by doing so. In the game-theoretic approach, there exists an underlying game Γ that determines the payoff of the agents depending on the actions they play. Two other games that extend Γ are also considered: Γ_d and Γ_{ACT}. In Γ_d, agents can communicate with a trusted mediator before playing an action in Γ; in Γ_{ACT}, they can communicate only among themselves. The goal is to show when certain types of equilibria in Γ_d can be implemented in Γ_{ACT}, where *implemented* means that the resulting distributions over action profiles are identical in both games. Forges [11] and Barany [5] showed that any Nash equilibrium in Γ_d can be implemented by a Nash equilibrium in Γ_{ACT} if $n \geq 4$, assuming that the communication in both Γ_d and Γ_{ACT} is synchronous.

Abraham, Dolev, Gonen, and Halpern [2] combined both points of view by considering (k, t)-*robust* equilibria. Intuitively, a (k, t)-robust equilibrium is a strategy profile (i.e., a strategy for each agent) in which no coalition of t malicious agents can decrease the payoff of anyone else and no coalition of k rational agents can increase their payoff by deviating, even when colluding with the other t malicious agents. Thus, a (k, t)-robust equilibrium deals with both Byzantine and rational behavior, as long as there are at most t Byzantine players and at most k rational players that deviate. Abraham et al. [2] generalized Forges and Barany's results by showing that any (k, t)-robust equilibrium in Γ_d can be implemented in Γ_{ACT} if $n > 3(k + t)$ (note that a $(1, 0)$-robust equilibrium is just a Nash equilibrium). They also proved a matching lower bound [3].

Abraham, Dolev, Geffner, and Halpern [1] (ADGH from now on) extended this result to the asynchronous setting. They showed that if $n > 4(k+t)$ and there exists a (k, t)-robust equilibrium $\vec{\sigma} + \sigma_d$ in the mediator game Γ_d (where $\vec{\sigma} + \sigma_d$ means that player i plays σ_i while the mediator plays σ_d), then there exists a (k, t)-robust equilibrium $\vec{\sigma}_{ACT}$ in Γ_{ACT} in an asynchronous setting such that, for all inputs, $\vec{\sigma}_{ACT}$ and $\vec{\sigma} + \sigma_d$ produce the same set of possible distributions over outputs (note that agents have no control over how long the messages take to be delivered, and this can affect the output).

Our goal in this article is to prove a lower bound that matches the upper bounds of ADGH. To do so, we would like to reduce implementing asynchronous $(k + t)$-secure computation to implementing (k, t)-robust mediators. If such a reduction were possible, the $n > 4(k + t)$ lower bound for implementing (k, t)-robust mediators would follow immediately from the same lower bound for secure computation [4, 8, 10]. Unfortunately, the existence of such a reduction is still an open problem. However, we show that there exists a nontrivial reduction from a slightly weaker notion of $(k + t)$-secure computation, which we call $(k + t)$-*strict secure computation*, to implementing (k, t)-robust mediators. This suffices to prove the desired lower bound. We thus start by providing a careful proof of the lower bound for $(k + t)$-strict secure computation in the asynchronous setting. In the process, we also give a simple alternative proof for the lower bound on asynchronous secure computation.[1]

[1]As Ran Canetti [private communication] agreed, there is a nontrivial problem with the proof given in his thesis [10]; a different technique is needed. We thank him for his comments.

Intuitively, a protocol t-strictly securely computes a function f if it satisfies the properties of secure computation, but only for adversaries consisting of exactly t malicious agents. It might seem that t-secure computation should be equivalent to t-strict secure computation. After all, if a function can be securely computed with adversaries of maximal size, surely it can be securely computed with smaller adversaries! As we show by example in Section 2.3, this is not the case. While investigating these issues, we noted an ambiguity in the definition of t-secure computation in BCG, which led us to consider yet another notion that we call *t-weak secure computation*. As the name suggests, it is weaker than t-secure computation; we show that it is actually equivalent to t-strict secure computation. By considering these variants of secure computation, we gain a deeper understanding of its subtleties.

Another contribution of this article is to show that BCG's result follows from that of ADGH. In ADGH, the authors claim that their result generalizes secure computation by showing that there exist games Γ_d with a (k, t)-robust strategies $\vec{\sigma}_d$ that cannot be $(k + t)$-securely computed (for instance, when the mediator takes into account the order in which it receives the messages). However, as we show in this article, it is not so obvious that all instances of secure computation can be captured by a (k, t)-robust strategy in some game. We show that, for all functions f on n inputs and all t such that $2t < n$, there exists a game $\Gamma^{f,t}$ and a $(t, 0)$-robust strategy $\vec{\sigma}_d$ for $\Gamma^{f,t}$ such that t-securely computing f reduces to implementing $\vec{\sigma}_d$ in Γ_{ACT}. This formally proves ADGH's claim.

2 BASIC DEFINITIONS

2.1 The Asynchronous Model

We assume a network where there is a reliable, authenticated, and the asynchronous channel between all pairs of players. This means that all messages sent by player i to player j are guaranteed to be delivered eventually, and that j can identify that these messages were sent by i. However, messages may be delayed arbitrarily. The order in which messages are received and the order in which the players are scheduled is decided by an entity called the *scheduler*.

We define the *view* h_i of player i to be the ordered sequence of local computations (including random coin tosses), messages sent and received (including senders and recipients), in addition to all the times in between in which i has been scheduled. Similarly, we define the view h_T of a subset T of players as the collection of views h_i with $i \in T$. The scheduler's view H_e consists of all the times in which each player has been scheduled in addition to all messages sent and received. The scheduler cannot access the contents of messages; thus, in H_e, messages are listed without their content. Note that in the distributed computing literature, it is often assumed that players are scheduled immediately after receiving a message. However, we allow the scheduler to decide separately when messages are delivered and when players are scheduled. This means that when a player moves, that player may have received no messages since its last move, or it may have received more than one (as opposed to exactly one). It is straightforward to check that all of our results also hold if we use the more standard model; we have separated message delivery from when players are scheduled only for ease of exposition.

2.2 Secure Computation

For the main definitions in this section, we need the following notation, largely taken from BCG. Given a finite domain D, let $\vec{x}$ be a vector in D^n. Given a set $C \subseteq [n]$, denote by $\vec{x}_C$ the vector obtained by projecting $\vec{x}$ onto the indices of C. Given a vector $\vec{z} \in D^{|C|}$, let $\vec{x}/_{(C,\vec{z})}$ be the vector obtained by replacing the entries of $\vec{x}$ indexed by C with the corresponding entries of $\vec{z}$. To simplify notation, given a function $f : D^n \to D$, we write $f_C(\vec{x})$ rather than $f(\vec{x}/_{(\overline{C},\vec{x}_0)})$ to denote the output of evaluating f on $\vec{x}$ with the entries in $\vec{x}$ not indexed by an element of C replaced by some default value $x_0 \in D$.

Suppose that a group of n agents wants to compute the output of a function $f : D^n \to D$, but the ith input x_i is known only by agent i. A protocol securely computes f if (a) all agents correctly compute f, regardless of the deviations of malicious players, and (b) malicious agents do not learn anything about the input of honest agents beyond what can be deduced from the output of f. Before going on, we need to make precise what it means to *correctly compute* f, since a malicious agent can lie about its input or not participate in the computation at all. Roughly speaking, the idea is to accept as correct any output of f that can be obtained from an input profile that differs from the actual input profile in at most t coordinates (intuitively, these coordinates are ones corresponding to inputs of malicious agents who did not submit a value or lied about their actual input). More precisely, we have the following definition:

Definition 2.1. A protocol $\vec{\pi}$ *t-securely computes f* in synchronous systems, if for every coalition T of at most t malicious agents and every protocol $\vec{\tau}_T$ for agents in T, there exist functions $h : D^{|T|} \to D^{|T|}$ and $O : D^{|T|} \times D \times T \to \{0,1\}^*$ such that, for each input vector $\vec{x}$,

(a) each agent $i \notin T$ outputs $f(\vec{x}/_{(T,h(\vec{x}_T))})$;
(b) each agent $i \in T$ outputs $O(\vec{x}_T, f(\vec{x}/_{(T,h(\vec{x}_T))}), i)$.

Note that h and O encode how malicious agents might lie about their inputs (if a malicious agent does not participate in the computation, its input is assumed to be the default value $x_0 \in D$) and what they output, respectively. We thus consider an output to be correct if only the inputs of agents in T used in the computation of f differ from their actual inputs, and if the output of malicious agents is just a function of the output of f and their own inputs. Note that this last requirement captures the fact that malicious agents do not learn anything besides the (honest agents') output of the secure computation protocol, since otherwise, they could use this extra information to generate outputs that cannot be written as such a function O. Since malicious agents can randomize, we assume that both h and O have an extra input r, a bitstring chosen uniformly at random from $\{0,1\}^{\omega}$ (the set of all finite bitstrings), and that agent i's output is distributed identically to $f(\vec{x}/_{(T,h(\vec{x}_T))})$ or $O(\vec{x}_T, f(\vec{x}/_{(T,h(\vec{x}_T))}), i)$, depending on whether i is honest. (See Definition 2.3 for the more standard formalization of this property.) BGW proved the following result:

THEOREM 2.2 ([7]). *If D is a finite domain, $n > 3t$, and $f : D^n \to D$, then there exists a protocol $\vec{\pi}$ that t-securely computes f in synchronous systems.*

Subtleties introduced by asynchrony make the definition of secure computation slightly more involved in asynchronous systems. In asynchronous systems, as is standard, we assume that there is a scheduler with its own protocol σ_e that decides the order in which agents act and how long it takes for a message to be delivered. As pointed out by ADGH, malicious agents can effectively communicate with the scheduler, so we can assume that the adversary and malicious agents are all controlled by a single entity. We call this entity the *adversary*; we define it as a tuple $A = (T, \vec{\tau}_T, \sigma_e)$ consisting of the set T of malicious agents, their joint protocol $\vec{\tau}_T$, and the scheduler's protocol σ_e. With such adversaries, there are deviations that are possible in asynchronous systems that are not possible in synchronous systems; specifically, the scheduler can delay a subset of agents until the other agents terminate the protocol. If the number of agents delayed is less than the number of malicious agents that the protocol tolerates, delayed honest agents are indistinguishable from malicious agents that never engage in the communication, and thus the remaining agents must be able to terminate regardless of the delay. Since the inputs of delayed honest agents are not taken into consideration, the adversary can choose a set $C \subseteq [n]$ of size at least $n - t$ and force the computation to ignore the inputs of agents not in C.

To define asynchronous secure computation, BCG introduced an *ideal adversary*, defined as a quadruple $A = (T, h, c, O)$ where

- T is the set of malicious agents;
- $h : D^{|T|} \times \{0,1\}^{\omega} \rightarrow D^{|T|}$ is the input substitution function;
- $c : D^{|T|} \times \{0,1\}^{\omega} \rightarrow \{C \subseteq [n] \mid |C| \geq n - t\}$ chooses a subset of agents (intuitively, the ones whose inputs are taken into consideration);
- $O : D^{|T|} \times \{0,1\}^{\omega} \times D \times T \rightarrow \{0,1\}^*$ is the output function for the malicious agents.

In the sequel, we use "ideal adversary" to refer to such a tuple (T, h, c, O), and reserve the term "adversary" for a tuple of the form $(T, \vec{\tau}_T, \sigma_e)$, as defined earlier.

Given a function $f : D^n \rightarrow D$, an ideal adversary $A = (T, h, c, O)$, and an input vector $\vec{x}$, let $C = c(\vec{x}_T, r)$ and $\vec{y} = \vec{x}/_{(T, h(\vec{x}_T, r))}$. Intuitively, C is the set of agents whose inputs are considered and $\vec{y}$ is the input profile obtained by replacing the actual inputs of agents in T with the output of h. The output of f with ideal adversary A and input $\vec{x}$ is an n-vector of random variables $\text{IDEAL}_{f,A}(\vec{x})$ whose ith component satisfies

$$\text{IDEAL}_{f,A}(\vec{x})_i = \begin{cases} (C, f_C(\vec{y})) & \text{if } i \notin T, \\ O(\vec{x}_B, r, f_C(\vec{y}), i) & \text{if } i \in T. \end{cases}$$

Note that the outputs of f with ideal adversaries are analogous to the outputs of secure computation in the synchronous case, except that here we must take into account the subset C of agents that provide their inputs. In asynchronous systems, secure computation is defined as follows:

Definition 2.3 (Secure Computation). Let $f : D^n \rightarrow D$ be a function of n variables over some finite domain D. The protocol $\vec{\pi}$ *t-securely computes f in an asynchronous setting* if the following hold for all (standard) adversaries $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| \leq t$:

- on all inputs, agents not in T terminate the protocol with probability 1;
- there exists an ideal adversary $A^{id} = (T, h, c, O)$ such that, for all inputs $\vec{x} \in D^n$, we have $\text{EXEC}_{\vec{\pi},A}(\vec{x}) \sim \text{IDEAL}_{f,A^{id}}(\vec{x})$ (i.e., $\text{EXEC}_{\vec{\pi},A}(\vec{x})$ and $\text{IDEAL}_{f,A^{id}}(\vec{x})$ are identically distributed), where $\text{EXEC}_{\vec{\pi},A}(\vec{x})$ is the output distribution that arises from running $\vec{\pi}$ with adversary A and input profile $\vec{x}$.

In other words, a protocol $\vec{\pi}$ t-securely computes some function f if, for all adversaries, it terminates with probability 1 and there exists an ideal adversary that, for all inputs, gives the same distribution over outputs.

Theorem 2.4 ([6]). *If D is a finite domain, $n > 4t$, and $f : D^n \rightarrow D$, then there exists a protocol $\vec{\pi}$ that t-securely computes f in asynchronous systems.*

2.3 Weaker Notions of Secure Computation

Note that the T in the second condition of Definition 2.3, that is, the T in the ideal adversary (T, h, c, O) is the same as the T in the adversary. This is also true in the BGW definition of t-secure computation. While we believe that this was also the intention of BCG, their definition simply says that there exists an ideal adversary, without specifying the set T (of malicious agents) that satisfies the second bullet of Definition 2.3. Taking this definition seriously leads to a slightly weaker notion of secure computation that we call *t-weak secure computation*, which is defined just as t-secure computation except that the ideal adversary A^{id} may involve any set T' of malicious agents such that $|T'| = t$ and $T' \supseteq T$, as opposed to consisting of the same set T of malicious agents as A.

We show that t-weak secure computation is strictly weaker than the standard notion of secure computation. To do so, we first introduce an intermediate notion of secure computation called

t-strict secure computation; it is defined just as t-secure computation, except that we require only that the properties are satisfied for adversaries of size exactly t (i.e., for $|T| = t$). As we mentioned in the introduction, somewhat surprisingly, t-strict secure computation is strictly weaker than t-secure computation, but, as we show next, it is actually equivalent to t-weak secure computation.

Theorem 2.5.

(a) *If a protocol $\vec{\pi}$ t-securely computes a function f, it also t-strictly securely computes f.*
(b) *A protocol $\vec{\pi}$ t-strictly securely computes a function f iff it t-weakly securely computes f.*
(c) *If $t > 1$ and $n > 4t$, there exists a function f on n variables and a protocol $\vec{\pi}$ such that $\vec{\pi}$ t-strictly securely computes f but does not t-securely computes f.*

The analogue of part (c) of Theorem 2.5 is easy to show if $n \le 2t$. For instance, we can easily check that a protocol where each agent does nothing and outputs $f(\vec{0})$ t-strictly securely computes any f for all t such that $n \le 2t$ (note that, in this case, for an ideal adversary consisting of t malicious agents, all agents can pretend to have input 0 and choose a set C arbitrarily), but it is not necessarily a t-secure computation of f if $t < n \le 2t$. What is perhaps surprising is that this result holds even when $n > 4t$.

Proof. Part (a) follows immediately from the definition of secure computation and strict secure computation. For part (b), first suppose that a protocol $\vec{\pi}$ t-strictly securely computes $f : D^n \rightarrow D^n$. Given an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| \le t$, consider an adversary of the form $A' = (T \cup T', \vec{\tau}_T + \vec{\pi}_{T'}, \sigma_e)$ such that $T \cap T' = \emptyset$ and $|T \cup T'| = t$. Note that agents in T' play $\vec{\pi}$, and thus they in fact do not deviate. Since $\vec{\pi}$ t-strictly securely computes f, there exists an ideal adversary $A^{id} = (T \cup T', h, c, O)$ such that $\mathrm{EXEC}_{\vec{\pi},A'}(\vec{x}) \sim \mathrm{IDEAL}_{f,A^{id}}(\vec{x})$ for all inputs. By construction, $\mathrm{EXEC}_{\vec{\pi},A'}(\vec{x}) \sim \mathrm{EXEC}_{\vec{\pi},A}(\vec{x})$, since the additional malicious agents in A' do not deviate from the protocol. Therefore, $\mathrm{EXEC}_{\vec{\pi},A}(\vec{x}) \sim \mathrm{IDEAL}_{f,A^{id}}(\vec{x})$, so $\vec{\pi}$ t-weakly securely computes f.

The converse is almost immediate from the definitions. Suppose that protocol $\vec{\pi}$ t-weakly securely computes $f : D^n \rightarrow D^n$ for some t. Given an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| = t$, then, by assumption, there exists an ideal adversary $A^{id} = (T, h, c, O)$ such that $\mathrm{EXEC}_{\vec{\pi},A}(\vec{x}) \sim \mathrm{IDEAL}_{f,A^{id}}(\vec{x})$.

For part (c), consider the following setup. Let $\mathbb{F}_2$ be the field with domain $\{0, 1\}$. Given n and t such that $t > 1$ and $n > 4t$, consider a function $f : (\mathbb{F}_2)^{n^3} \rightarrow (\mathbb{F}_2)^{n^2}$ that does the following. Given the input $(x^i, c^i, y^i, z^i) \in (\mathbb{F}_2)^{n^2}$ of each agent i, where $x^i \in \mathbb{F}_2$, $c^i \in (\mathbb{F}_2)^n$, $y^i \in (\mathbb{F}_2)^{t-1}$, and z^i consists of the remaining $n^2 - n - t$ coordinates (which do not affect the function f; they are needed because in Definition 2.3, the input space of each agent must be the same as the output space of the function), let $p_i \in \mathbb{F}_2[X]$ be the unique polynomial of degree $t-1$ such that $p_i(0) = x^i$ and $p_i(j) = y^i_j$ for all $j = 1, 2, \ldots, t-1$. The output of f is then $\{p_i(j) + c^j_i\}_{i,j \in [n]}$. In other words, f encodes the first coordinate of each agent's input using Shamir's agent secret sharing scheme [13]. The polynomial p_i that each agent i uses to do the encoding and the one-time pads c^j_i added by i to each of the shares are part of i's input, and not known by the other agents. However, a coalition T of t malicious agents can reconstruct the values $p_i(j)$ for all $i \in [n]$ and $j \in T$, and thus is able to reconstruct each x_i as well, since the agents in T know t points on each polynomial p_i, although no coalition of size strictly smaller than t knows those values.

Consider a protocol $\vec{\pi}$ that consists of the following: each agent i performs its part of BCG's t-secure computation protocol to compute f and then, if i is included in the core set of the output, i broadcasts the first bit of its input. By the earlier argument, if the adversary is of size exactly t, it can reconstruct the first coordinate of the inputs of the agents in the core-set from the output of f and its own inputs, which means that the values broadcast after BCG's secure computation protocol do not give any extra information about the inputs of honest agents to the adversary. However, this is not true for smaller adversaries. Thus, $\vec{\pi}$ t-strictly securely computes f, but does not t-securely compute f. □

2.4 Implementing Mediators

In this section, we formalize the definition of Γ_d and Γ_{ACT} and the notion of (k,t)-robust equilibrium.

2.4.1 Normal-form and Bayesian Games. A *normal-form* game Γ is a tuple (P, A, U), where $P = \{1, \dots, n\}$ is the set of *players*, $A = A_1 \times \cdots A_n$, where A_i is the set of possible actions for player $i \in P$, and $U = (u, \dots, u_n)$ is an n-tuple of *utility* functions $u_i : A \to \mathbb{R}$, again, one for each player $i \in P$. A *pure strategy profile* $\vec{a}$ is an n-tuple of actions $(a_1, \dots, a_n)$, with $a_i \in A_i$. A *mixed strategy* for player i is an element of $\Delta(A_i)$, the set of probability distributions on A_i. We extend u_i to mixed strategy profiles $\sigma = (\sigma_1, \dots, \sigma_n)$ by defining $u_i(\sigma) = \sum_{(a_1,\dots,a_n)\in A} \sigma_1(a_1)\dots\sigma_n(a_n)u_i(a_1, \dots, a_n)$: that is, the sum over all pure strategy profiles $\vec{a}$ of the probability of playing $\vec{a}$ according to σ times the utility of $\vec{a}$ to i. As is standard, we use $\vec{\sigma} := (\sigma_1, \dots, \sigma_n)$ to denote a strategy profile for n players in which each player i plays σ_i, and use (σ_{-T}, τ_T) to denote the strategy where each player $i \notin T$ uses the strategy σ_i while $j \in T$ uses the strategy τ_j.

Bayesian games extend normal-form games by assuming that each player $i \in P$ has a *type* $t_i \in T_i$. A player's type can be thought of as private information that the player has, such as whether he is lazy or industrious. In our applications, an agent's type will be its input. Types are assumed to be sampled from a distribution $q \in \Delta(T)$, where $T = T_1 \times \cdots \times T_n$. The utility u_i of a player i is a function of not only the action profile played, but also of of the type profile $(t_1, \dots, t_n)$. Formally, a Bayesian game is a tuple (P, T, q, A, U), where, as in normal-form games, P, A, and U are the set of players, their actions, and their utility functions, respectively; T is the set of possible type profiles, and q is a distribution in $\Delta(T)$.

A *strategy* in a Bayesian game for player i is a map $\mu_i : T_i \to \Delta(A_i)$. Intuitively, a strategy in a Bayesian game tells player i how to choose its action given its type. Since the distribution q is common knowledge, given a strategy profile $\vec{\mu} = (\mu_1, \dots, \mu_n)$ in Γ, the expected utility of a member i of a coalition K is

$$u_i(\vec{\mu}) = \sum_{\vec{t}_K \in T_K} q(\vec{t}_K) \sum_{\vec{t}} q(\vec{t} \mid \vec{t}_K) u_i(\vec{\mu}(\vec{t})),$$

where $u_i(\vec{\mu}(\vec{t}))$ denotes the expected utility of player i when an action profile is chosen according to $\mu(\vec{t})$. Intuitively, we are assuming that players in K can share their types, which is why we condition on $\vec{t}_K$.

2.4.2 Defining Γ_d and Γ_{ACT}. Given a Bayesian game Γ, we consider two extensions Γ_d and Γ_{ACT} with the same set of players. In Γ_d, players can communicate with a trusted mediator d, who is a non-strategic player (i.e., there is no utility function for the mediator) and uses a commonly-known strategy σ_d. We call Γ_d the *mediator game*, and denote by $\vec{\sigma} + \sigma_d$ the strategy in which players play strategy profile $\vec{\sigma}$ and the mediator plays σ_d. In the *communication game* Γ_{ACT}, there is no trusted mediator but players can communicate with each other. In both Γ_d and Γ_{ACT}, players can decide at any time to play an action a in the underlying game Γ; however, each player can play an action in Γ at most once. The resulting payoff or utility of each player is determined by the action profile played and the type profile $\vec{t}$, using the utility function in Γ. Strategies in Γ_d and Γ_{ACT} describe how each player communicates with the mediator and the other players, and when each player plays an action in the underlying game Γ.

In this article, we assume that the communication in Γ_d and Γ_{ACT} is asynchronous, as described in Section 2.1. This means that there a player i may end up deadlocked, waiting for other players' messages, and never gets to play an action in the underlying game. If this happens, we assume that the action played by i is a default action $\perp \in A_i$.[2]

Given these definitions, we can define what it means for a strategy profile to *implement* another in asynchronous systems:

Definition 2.6. Strategy profile $\vec{\sigma}$ in Γ_{ACT} *implements* strategy profile $\vec{\tau}$ in Γ_d if, for all type profiles $\vec{t}$ and all schedulers σ_e, there exists a scheduler σ'_e such that the distribution over output profiles induced by $\vec{\sigma}$ with input profile $\vec{t}$ and scheduler σ_e is identical to the distribution over output profiles induced by $\vec{\tau}$ with input profile $\vec{t}$ and scheduler σ'_e.

It is important to note that strategy profiles in Γ_d and Γ_{ACT} are essentially just protocols in which the type profiles t_i are the players' inputs and in which players have the option to play an action in the underlying game. Thus, in Γ_d and Γ_{ACT}, the terms *strategy* and *protocol* are equivalent (we use *strategy profile* and *joint protocol* to refer to tuples of strategies or protocols), as are *type* and *input*. For the sake of simplicity, we will stick with the distributed computing notation, except when referring specifically to purely game-theoretic concepts (e.g., normal-form or Bayesian games). We assume that all games considered from here on are Bayesian games, unless explicitly stated otherwise.

2.4.3 (k,t)-robustness. In large systems, there will almost surely be players who act in apparently arbitrary ways. This may not necessarily happen because they are malicious; it could be that they don't understand how the system works or that their utilities are not what the system designer is expecting. In order to tolerate these players' behavior, it is necessary to take into account the worst-case scenario, where they might be malicious. Thus, Abraham, Dolev, Gonen, and Halpern [2] introduced the notion of a (k,t)-robust equilibrium, a solution concept that is appropriate in systems in which at most k players are rational and at most t players are malicious.

Definition 2.7. Given a game Γ, a strategy profile $\vec{\sigma}$ is *t-immune* if, for all subsets T of size at most t and all strategies $\vec{\tau}_T$ for players in T, $u_i(\vec{\sigma}_{-T}, \vec{\tau}_T) \geq u_i(\vec{\sigma})$ for all $i \notin T$, where $u_i(\vec{\sigma})$ is the payoff of player i when players play $\vec{\sigma}$.

Intuitively, a strategy profile is a t-immune equilibrium if no subset of at most t players can decrease the payoff of other players by deviating,

Definition 2.8. A strategy profile $\vec{\sigma}$ is a *(k,t)-resilient* (respectively, *strongly (k,t)-resilient*) *equilibrium* of a game Γ if, for all disjoint subsets K and T of sizes at most k and t, respectively, and all strategy profiles $\vec{\tau}_{K\cup T}$ for players in $K \cup T$, $u_i(\vec{\sigma}_{-(K\cup T)}, \vec{\tau}_{K\cup T}) \leq u_i(\vec{\sigma}_{-T}, \vec{\tau}_T)$ for some (respectively, for all) $i \in K$.

Intuitively, a strategy protocol is a (k,t)-robust equilibrium if no subset of at most k players can all increase their payoffs, even if they can collude with up to t malicious players. It is a strong (k,t)-robust equilibrium if not even one player in the set can increase its payoff.

Definition 2.9. A strategy profile is a *(k,t)-robust* (respectively, *strongly (k,t)-robust*) *equilibrium* in a game Γ if it is t-immune and a (k,t)-resilient (respectively, strongly (k,t)-resilient) equilibrium.

[2]There exist other approaches for how to choose which action player i is taken to play if i is deadlocked; we chose this one for definiteness. The same results hold for the other approaches that have been taken. See for [1] for further discussion of this subject.

Abraham, Dolev, Gonen, and Halpern [2] proved the following result:

THEOREM 2.10 ([2]). *If $\vec{\sigma} + \sigma_d$ is a (k, t)-robust equilibrium of a synchronous game Γ_d that extends some game Γ and $n > 3(k + t)$, then there exists a (k, t)-robust equilibrium $\vec{\sigma}_{ACT}$ of Γ_{ACT} that implements $\vec{\sigma} + \sigma_d$.*

ADGH proved an analogous result for asynchronous systems.

THEOREM 2.11 ([1]). *If $\vec{\sigma} + \sigma_d$ is a (k, t)-robust equilibrium of a game Γ_d that extends some game Γ and $n > 4(k+t)$, then there exists a (k, t)-robust equilibrium $\vec{\sigma}_{ACT}$ of Γ_{ACT} that implements $\vec{\sigma} + \sigma_d$.*

It is easy to t-securely compute a function f with the help of a mediator: Each player sends its input to the mediator, the mediator waits until it receives an input from at least $n - t$ agents (in synchronous systems it just waits one round), then it computes the output of f given the input of the players, and sends it to all players. However, despite the fact that we think of t-secure computation in terms of mediators, it is not obvious that Theorem 2.4 follows from Theorem 2.11, due to the differences between the definitions of (k, t)-robustness and secure computation. In Section 5, we sketch how to reduce t-secure computation to implementing strongly $(t, 0)$-robust equilibria in mediator games.

To conclude this section, note that if a strategy profile is a (k, t)-robust equilibrium (respectively, strongly (k, t)-robust equilibrium), it is also a $(k+k', t-k')$-robust equilibrium (respectively, strongly $(k+k', t-k')$-robust equilibrium), for $0 \le k' \le t$. However, the converse is not necessarily true, since rational players with known utilities are far more restricted than malicious players. In fact, ADGH [1] showed that if there is a way to punish rational players, then any (k, t)-robust equilibrium with a mediator can be implemented without a mediator if $n > 3k + 4t$. Intuitively this means that, if a punishment strategy exists, each rational player needs three additional honest players to counteract its behavior, while malicious players need four.

2.5 Weak Consensus

Some of the results in this article involve reductions from a well-known problem in the distributed computing literature called *Weak Consensus*.

Definition 2.12 ([12]). A protocol $\vec{\sigma}$ for n agents is a *t-resilient implementation of weak consensus* if the following holds for all adversaries $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| \le t$ and all histories:

(a) All agents not in T output the same value.
(b) If all agents are honest and have the same input x, all agents output x.

As the name suggests Weak Consensus is weaker than the more standard *Consensus* problem. In Weak Consensus, honest players must output x only when all players are honest and they all have input x; in standard Consensus, they must output x even if only the honest players have input x, no matter what input the malicious players have. Clearly, if $\vec{\sigma}$ is a t-resilient implementation of Consensus, it is also a t-resilient implementation of Weak Consensus. Lamport [12] showed that both problems have the same lower bound:

THEOREM 2.13 ([12]). *If $n \le 3t$, there is no t-resilient implementation of weak consensus.*

3 MAIN RESULTS

In this article, we show that the bound in Theorem 2.11 is tight. In this section, we briefly outline the structure of the proof, leaving the details of the arguments to later sections.

THEOREM 3.1. *If $k + t + 1 < n \le 4k + 4t$, there exists a (k, t)-robust (respectively, strongly (k, t)-robust) equilibrium, $\vec{\sigma} + \sigma_d$ for n agents and a mediator such that there is no (k, t)-robust (respectively, strongly (k, t)-robust) equilibrium $\sigma_{\vec{ACT}}$ that implements $\vec{\sigma} + \sigma_d$.*

We break up the proof of Theorem 3.1 into two cases:

3.1 Case 1: $3k + 3t \leq n \leq 4k + 4t$

For this case, we show that $(k+t)$-strictly securely computing a function f reduces to implementing a (k, t)-robust joint protocol $\vec{\sigma} + \sigma_d$ for some game $\Gamma_d^{f,k,t}$ with a mediator. To make this precise, we need the following definition:

Definition 3.2. If $g : A \rightarrow B$ and σ is a protocol that plays actions in A, then $g(\sigma)$ is the protocol that is identical to σ except that playing each action $a \in A$ is replaced by outputting $g(a) \in B$. If $\vec{\sigma}$ is a joint protocol where each player i plays actions in A, then $g(\vec{\sigma}) = (g(\sigma_1), \ldots, g(\sigma_n))$.

Theorem 3.3. *If $f : D^n \rightarrow D$, D is a finite domain, and $2(k+t) < n$, then there exists a game $\Gamma_d^{f,k,t}$ in which all players have the same set A of possible actions, a function $g : A \rightarrow D$, and a (k, t)-robust (respectively, strongly (k, t)-robust) equilibrium $\vec{\sigma} + \sigma_d$ for n agents and the mediator in $\Gamma_d^{f,k,t}$ such that if $\vec{\sigma}_{ACT}$ is a (k, t)-robust (respectively, strongly (k, t)-robust) equilibrium that implements $\vec{\sigma} + \sigma_d$, then $g(\vec{\sigma}_{ACT})$ $(k + t)$-strictly securely computes f.*

Theorem 3.3 shows that if we could implement all (k, t)-robust equilibria in the mediator game using only communication among the n players, then all functions on n variables would be $(k + t)$-strictly securely computable. The proof of Theorem 3.1 for $3(k + t) \leq n \leq (k + t)$ follows from the fact that if $3(k + t) \leq n \leq 4(k + t)$, then there exist functions that cannot be $(k + t)$-weakly securely computed (and hence, cannot be $(k + t)$-strictly securely computed).

Theorem 3.4.

(a) *If $n > 4t$ or $n \leq 2t$, then every function $f : D^n \rightarrow D$ can be t-weakly securely computed in asynchronous systems.*
(b) *If $3t \leq n \leq 4t$, there exists a domain D and a function $f : D^n \rightarrow D$ that cannot be t-weakly securely computed in asynchronous systems.*

The proof of Theorem 3.4 is given in Section 6. A slight variation of it provides a simple proof for the well-known lower bound on secure computation in asynchronous systems:

Theorem 3.5 ([4, 8, 10]).

(a) *If $n > 4t$ or $n \leq t$, then for all domains D, every function $f : D^n \rightarrow D$ can be t-securely computed in asynchronous systems.*
(b) *If $t < n \leq 4t$ there exists a domain D and a function $f : D^n \rightarrow D$ that cannot be t-securely computed in asynchronous systems.*

Note that the range of n and t for which a function can be t-securely computed and t-weakly securely computed are different, and that Theorem 3.4 does not state whether a function can always be t-weakly securely computed if $2t < n < 3t$. We leave that as an open problem. While it might seem that there should be an easy reduction from implementing mediators to secure computation, the techniques needed to prove Theorem 3.5 do not seem suffice to prove Theorem 3.1; we seem to need a number of new ideas (see Section 5).

3.2 Case 2: $k + t + 1 < n \leq 3k + 3t$

If $k + t + 1 < n \leq 3k + 3t$, we show that implementing $(k + t)$-resilient weak consensus with n players can be reduced to implementing (k, t)-robust mediators:

Theorem 3.6. *If $n > k + t + 1$, then there exists a game $\Gamma_d^{k,t}$ with a mediator in which all agents have the same set A of possible actions, a function $g : A \rightarrow \{0, 1\}$, and a (k, t)-robust (respectively,* *strongly (k, t)-robust) equilibrium $\vec{\sigma} + \sigma_d$ for n players and the mediator in $\Gamma_d^{k,t}$ such that if $\vec{\sigma}_{ACT}$ is a (k, t)-robust (respectively, strongly (k, t)-robust) equilibrium that implements $\vec{\sigma} + \sigma_d$, then $g(\vec{\sigma}_{ACT})$ is a $(k + t)$-resilient implementation of weak consensus.*

The proof of Theorem 3.1 for $k + t + 1 \leq n \leq 3(k + t)$ follows easily from Theorem 3.6 and Theorem 2.13.

3.3 Approximate Implementation

ADGH [1] showed that if we allow an arbitrarily small probability of error, then any (k, t)-robust equilibrium with a mediator can be implemented without a mediator if $n > 3k + 3t$ (as opposed to $n > 4k + 4t$ for error-free implementation). As shown in [3, Theorem 4], we can do no better, even in the synchronous case. *A fortiori*, this lower bound also holds in the asynchronous case.

4 PROOF OF THEOREM 3.3

We prove Theorem 3.3 only for the case of (k, t)-robustness; the proof in the case of strong (k, t)-robustness is identical. The proof has two parts. We first give the necessary intuition and provide the necessary constructions for Theorem 3.3, and then prove formally that these constructions do indeed satisfy the conditions of the theorem.

A naive construction of Γ_d and $\vec{\sigma} + \sigma_d$ proceeds as follows. The set of actions of each agent consists of all possible outputs of a secure computation of f in addition to their type (more precisely, actions have the form (C, z, Q) with $C \subseteq [n]$, $z \in D$, and $Q \in \{H, R, M\}$, where H stands for honest, R stands for rational, and M stands for malicious). If there is no subset S of at least $n - k - t$ honest agents such that agents in S securely compute f, that is, every subset S of $n - k - t$ agents either do not all output the same value or they all output a value that is not a possible output of a secure computation of f, then rational agents get a higher payoff and/or the honest agents get a lower payoff. In $\vec{\sigma} + \sigma_d$, each agent sends its input to the mediator when it is scheduled for the first time. The mediator waits until it receives the input x_i from a set C of agents with $|C| \geq n - k - t$, then computes $z := f_C(\vec{x})$ and sends (C, z, H) to all agents. Agents play (C, z, H) when they receive the message.

It would seem that any (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_d$ also $(k + t)$-strictly securely computes f. In fact, any $(k + t)$-secure computation of f is also a (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_d$, but the converse does not necessarily hold. Consider a protocol $\vec{\sigma}_{ACT}$ in which agents run BCG's $(k + t)$-secure computation protocol and then broadcast their inputs immediately afterwards. It is easy to check that $\vec{\sigma}_{ACT}$ is a (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_d$ if $n > 4(k + t)$, but that $\vec{\sigma}_{ACT}$ does not $(k + t)$-securely compute f, since it leaks the honest agents' inputs to all other agents.

This shows that it is necessary to somehow encode all information that malicious agents can learn into the set of actions of Γ_d in such a way that they can increase their payoff if they manage to learn anything about the other agents' inputs besides what can be learned from the output of the computation. The idea for doing this is that, besides the output, the action of each agent should include a *guess* as to what the input profile $\vec{x}$ is (they can also guess $\perp$ if they have no guess). If an agent i guesses correctly it receives an additional positive payoff q_i, while if it guesses wrong, its payoff decreses by 1. The value of q_i should be chosen in such a way that (a) it is never worthwhile deviating if i is not able to learn anything besides the output, and (b) it is always worthwhile deviating if i is able to learn something (otherwise, $\vec{\sigma}_{ACT}$ may not $(k + t)$-strictly securely compute f even if it is a (k, t)-robust equilibrium, as in the example above). Given the set C of agents whose inputs are included in the computation, the output z of f, and the input profile $\vec{x}$, let p_i be the probability that an agent i guesses $\vec{x}$ conditional on its own input x_i, C, and z. Conditions (a) and (b) imply that $p_i q_i + (1 - p_i)(-1) \leq 0$ and $p_i q_i + (1 - p_i)(-1) \geq 0$ respectively, which means that $p_i q_i + p_i - 1 = 0$ and thus that $q_i = p_i^{-1} - 1$.

This approach cannot be generalized easily to a situation where a coalition of agents may deviate. In this case, an agent in the coalition will know the values of all agents in the coalition, not just its own input. Moreover, if an agent in the coalition plays just like an honest agent except that it tells the other coalition members its input, then this is completely indistinguishable (by the honest agents) from the scenario in which that agent is honest and the other members of the coalition were just lucky guessing its input. In addition, an agent can lie about its input if it is easier to guess the input profile with a different input than its own. For instance, suppose that $D = \mathbb{F}_2$ and that $f(\vec{x}) = \prod_{i=1}^{n}(1 - x_i)$. If i has input 1 and plays honestly, then it learns absolutely nothing about the other agents' inputs, since the output will be 0 no matter what. However, if i pretends to have input 0, i will learn more information: if the output is 1, then all agents have input 0; otherwise, at least one agent has input 1. In this case, it would always be worthwhile for agent i to act as if it has input 0, regardless of its actual input. This shows that to compute the probability that the adversary guesses the inputs correctly, it is critical to know who is malicious and what inputs the malicious agents are pretending to use in the computation.

To deal with the fact that we may not be able to tell which agents are deviating, we require that exactly $k + t$ agents must try to guess a non-$\bot$ value in order to get an additional (positive or negative) payoff. Moreover, their guesses must be identical. If honest agents always guess $\bot$, this suffices to identify the coalition of $k + t$ deviating agents given their action profile. Note that this is why we require strict secure computation in Theorem 3.3. If we required only (standard) secure computation, smaller adversaries wouldn't be able to get a better payoff, even if they managed to guess the inputs of everyone else (so the protocol would not satisfy condition (b)). To deal with agents lying about their inputs, we require that the action profile of the agents encode the inputs used by the agents for the computation (even though these inputs may differ from their actual inputs). The probability of guessing the input profile is based on the inputs used, not agents' actual inputs. Note that these values must be encoded into the action profile without any coalition of $k+t$ agents learning anything about them. This can be done as follows: each agent i, in addition to the set C, the output z, and their guess b_i, also outputs n values $s_{i1}, \dots, s_{in}$ such that the values $s_{i,j}$ for a fixed j encode the value used by j for the computation (using Shamir's secret-sharing scheme).

There is one final issue. The definition of $(0, t)$-robustness is equivalent to that of t-immunity, which means that no coalition of t agents can decrease the payoff of other agents by deviating. In this case, the effect of a coalition of t agents being able to learn something about the inputs should be to decrease the payoffs of the remaining agents, rather than increasing their own payoff. To deal with this, we require agents to declare wither they are G (good), R (rational), or M (malicious). If a coalition of $(k + t)$ agents tries to guess the inputs of everyone else, if they all declare R, then they get an additional payoff as described above. Otherwise, everyone gets the negative of that value.

We now formalize these ideas. Given f and integers k and t such that $n > 2k + 2t$, consider the game $\Gamma^{f,k,t}$ defined as follows. The input profile of the agents is chosen uniformly at random from D^n. The set of actions of each agent in $\Gamma^{f,k,t}$ is $\{G, R, M\} \times 2^{[n]} \times D \times D^n \times (D^n \cup \{\bot\})$, so an action of agent i has the form $a_i = (Q_i, C_i, z_i, \vec{s}_i, b_i)$, where $Q_i \in \{G, R, M\}$, $C_i \subseteq [n]$, $z_i \in D$, $x_i \in D$, and $b_i \in D^n \cup \{\bot\}$. Intuitively, Q_i denotes if i is good (G), rational (R), or malicious (M), (C_i, z_i) is i's output in the secure computation of f; s_{ij} is i's share of j's input (this will be made clearer below), and b_i is i's guess of the (supposedly secret) input, where $b_i = \bot$ if i has no guess.

We next define the utility function. We take $u_i = u_i^1 + u_i^2$, where, intuitively, u_i^1 is the utility that i gets if honest agents either output different values or some honest agent outputs a value that is not a possible output of a secure computation of f and u_i^2 is the utility that i gets from guessing the correct input of the other agents. To define u_i^1, we first define what it means for an action profile $\vec{a}$

to be *secure for an input profile* $\vec{x}$. This is the case if there exist subsets $C, T \subseteq [n]$ with $|C| \geq n-t-k$ and $|T| = k+t$, a vector $\vec{v} \in D^{k+t}$, and n polynomials $p_1, \dots, p_n$ of degree $k+t$ (where, intuitively, p_j encodes j's input, so $p_j(i)$ is i's share of j's input) such that, for each agent $j \notin T$, the action $a_j = (Q_j, C_j, z_j, \vec{s}_j, b_j)$ of agent j satisfies (1) $Q_j = G$, (2) $C_j = C$, (3) $b_j = \perp$, (4) $z_j = f(\vec{y})$, where $\vec{y} = (\vec{x}/_{(T,\vec{v})})/_{(\overline{C},\vec{0})}$ (5) $p_{j'}(j) = s_{jj'}$ for all $j' \in [n]$, and (6) $p_j(0) = y_j$. We say that $\vec{a}$ *is* $(T, \vec{x})$*-secure* if these properties hold for the set T. Intuitively, $\vec{a}$ is $(T, \vec{x})$-secure if it could be the output of a $(k+t)$-secure computation of f with input $\vec{x}$, and the inputs used for the computation—which may differ from the the actual input profile due to deviating agents lying about their inputs—were shared correctly between the agents. If $\vec{a}$ is secure for $\vec{x}$, then $u_i^1(\vec{a}, \vec{x}) = 0$ for all $i \in [n]$; if $\vec{a}$ is not secure for $\vec{x}$ and at least one agent i played R (i.e., played an action with $Q_i = R$), then $u_j^1(\vec{a}, \vec{x}) = 1$ for all agents j; otherwise, $u_j^1(\vec{a}, \vec{x}) = -1$ for all agents j.

If $\vec{a}$ is not secure for $\vec{x}$, then $u_i^2(\vec{a}, \vec{x}) = 0$. If $\vec{a}$ is secure for $\vec{x}$, let K be the subset of agents that did not play G. Note that if $\vec{a}$ is secure for $\vec{x}$, then $|K| \leq k+t$. If $|K| < k+t$ or not all agents in K guess the same value b (i.e., not all agents in K have the same value b as the last component of their action), then $u_i^2(\vec{a}, \vec{x}) = 0$ for all i. Otherwise, let b be the common guess of agents in K and let p be the probability that a vector $\vec{w}$ sampled uniformly from D^n is equal to $\vec{x}$, conditional on $\vec{w}_K = \vec{y}_K$ and $f_C(\vec{w}) = z$. Note that if $\vec{a}$ is $(T, \vec{x})$-secure for some T, then C is uniquely determined by $\vec{a}$, and if, in addition, $n > 2(t+k)$, then $\vec{y}$ is also uniquely determined by the shares $\vec{s}_i$ of agents $i \notin T$. If $b = \vec{\perp}$, then $u_i^2(\vec{a}, \vec{x}) = 0$ for all i. If at least one agent $i \in K$ played R in its action, then, if $b = \vec{x}$, $u_i^2(\vec{a}, \vec{x}) = p^{-1} - 1$ for all $i \in K$; otherwise, $u_i^2(\vec{a}, \vec{x}) = -1$ for all $i \in K$. On the other hand, if no agent $i \in K$ played R in its action, then, if $b = \vec{x}$, $u_i^2(\vec{a}, \vec{x}) = 1 - p^{-1}$ for all $i \notin K$; otherwise, $u_i^2(\vec{a}, \vec{x}) = 1$ for all $i \notin K$. Note that since $p(p^{-1}-1)-(1-p) = 0$, the payoffs u_i^2 are designed in such a way that the adversary can, in expectation, either increase its payoff (if there are any rational agents) or decrease the payoff of everyone else (if there are no rational agents) if it can guess the inputs of honest agents with higher probability than p (which is the probability of guessing the honest agents' inputs if the adversary knew nothing but the output of the function and its own protocol and inputs).

Consider the following joint protocol $\vec{\sigma} + \sigma_d$ for $\Gamma_d^{f,k,t}$. According to σ_i, agent i sends its input to the mediator at the beginning of the game. If i receives a message *msg* from the mediator, it plays *msg* in the underlying game. According to σ_d, the mediator waits until there exists a set $C \subseteq [n]$ with $|C| \geq n-t-k$ such that it has received exactly one message from each agent $i \in C$ and each of these messages consists of a value $y_i \in D$. The mediator computes n polynomials $p_1, \dots, p_n \in D[X]$ of degree $k+t$ whose non-constant coefficients are chosen uniformly at random such that $p_i(0) = y_i$ if $i \in C$ and $p_i(0) = 0$ otherwise. It then computes $z := f(p_1(0), \dots, p_n(0))$ and sends $(G, C, z, p_1(i), \dots, p_n(i), \perp)$ to each agent i.

Proposition 4.1. *$\vec{\sigma} + \sigma_d$ is a (k,t)-robust equilibrium and the equilibrium payoff is* 0.

Proof. Let $u_i(\vec{\sigma}, A, x_T)$ be the expected payoff of agent i when running $\vec{\sigma}$ with an adversary A that has input x_T. It follows by construction that $u_i^1(\vec{\sigma} + \sigma_d, A, \vec{x}_T) = 0$ for all adversaries $A = (T, \vec{\tau}_T, \sigma_e)$ of size at most $k+t$ and all inputs since, no matter what T or $\vec{x}_T$ are, the output profile $\vec{a}$ is $(T, \vec{x})$-secure for all input profiles $\vec{x}$.

Thus, $\vec{\sigma} + \sigma_d$ is not a (k,t)-robust equilibrium if and only if there exists an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| \leq k+t$ and an input profile $\vec{x}_T$ such that, in expectation, (a) $u_i^2(\vec{\sigma} + \sigma_d, A, \vec{x}_T) > 0$ for some $i \in T$, or (b) $u_i^2(\vec{\sigma} + \sigma_d, A, \vec{x}_T) < 0$ for all $i \notin T$. The definition of u_i^2 guarantees that, in both cases, the adversary must consist of exactly $k+t$ agents and these agents must all play a non-G action. Moreover, these agents must guess the input of honest agents with a probability higher than they could guess it by just knowing the output of the function, their protocol, and their inputs.

However, the construction of $\vec{\sigma} + \sigma_d$ guarantees that they don't have any extra information (note that C depends only on the adversary, and that the adversary does not get any information about the input of the honest agents besides the value of z, since the polynomials p_i are all of degree $k + t$). □

We next show that if there exists a (k, t)-robust equilibrium $\vec{\sigma}'$ that implements $\vec{\sigma} + \sigma_d$, then $\vec{\sigma}'$ also $(k + t)$-securely computes f. We first need the following lemma.

Lemma 4.2. *If $\vec{\sigma}_{ACT}$ is a (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_d$, then for all adversaries $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| = k + t$, all inputs $\vec{x}$, and all views H of $\vec{\sigma}_{ACT}$ with adversary A and input $\vec{x}$, the action profile $\vec{a}$ played in H is $(T, \vec{x})$-secure.*

Proof. Suppose that $k > 0$. If there exists an input $\vec{x}$ and an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| = k + t$ such that, for some view H, the action profile $\vec{a}$ played in H is not $(T, \vec{x})$-secure, consider the adversary $A' = (T, \vec{\tau}'_T, \sigma_e)$ where $\vec{\tau}'_T$ is identical to $\vec{\tau}_T$, except that if a agent $i \in T$ plays an action a with τ_i, then i instead plays $(R, \emptyset, 0, 0, \perp)$ with τ'_i. Thus, if an action profile $\vec{a}'$ played in some view H' when $\vec{\sigma}_{ACT}$ is run with adversary A' is $(T', \vec{x})$-secure, then $T \subseteq T'$ (note that for an action profile $\vec{a}$ to be $(T', \vec{x})$-secure, we require that all agents not in T' play G in the first component, and none of the agents in T plays G) and, since $|T| = k + t$, $T = T'$. Since the views generated by playing with adversaries A and A' are indistinguishable by honest agents, if there exists a view H with adversary A and input $\vec{x}$ such that the action $\vec{a}$ played in H is not $(T, \vec{x})$-secure, then the resulting action profile $\vec{a}'$ of playing $\vec{\sigma}_{ACT}$ with adversary A' and input $\vec{x}$ in which all agents use the same randomization as in H is not $(T, \vec{x})$-secure, and all agents in T would get a payoff of 1 rather than 0. It follows that $\vec{\sigma}_{ACT}$ is not a (k, t)-robust equilibrium. If $k = 0$, the argument is analogous, except that agents in T play $(M, \emptyset, 0, 0, \perp)$ rather than $(R, \emptyset, 0, 0, \perp)$ □

To complete the proof of Theorem 3.3, we must show that if $\vec{\sigma}_{ACT}$ is a (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_d$, the output of an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| = k + t$ is just a (randomized) function of its input $\vec{x}_T$ and the output v of the function. To do this, we need the following lemma:

Lemma 4.3. *Consider two random variables X and Y that take values on countable spaces S_1 and S_2 respectively. Then,* $\Pr[X = x \mid Y = y] = \Pr[X = x \mid Y = y']$ *for all $x \in S_1$ and $y, y' \in S_2$ if and only if X and Y are independent.*

Proof. If $\Pr[X = x \mid Y = y]$ does not depend on y, there exists a constant λ_x such that $\Pr[X = x \mid Y = y] = \lambda_x$ for all $y \in S$. Then, since $\Pr[X = x \mid Y = y] = \frac{\Pr[X=x, Y=y]}{\Pr[Y=y]}$, it follows that $\Pr[X = x, Y = y] = \lambda_x \Pr[Y = y]$. Therefore, $\sum_{y \in S_2} \Pr[X = x, Y = y] = \sum_{y \in S_2} \Pr[Y = y]$, which gives that $\lambda_x = \Pr[X = x]$, as desired. The converse is straightforward. □

We can now complete the proof of Theorem 3.3. Suppose that $k > 0$. Recall that if all agents $i \in T$ set b_i to some input $\vec{x}$, they get a payoff of $p_{\vec{x}}^{-1} - 1$, where $p_x \in [0, 1]$ if the input profile is indeed $\vec{x}$, and otherwise they get -1. Given a view H_T in which the adversary has input $\vec{x}_T$ and honest agents output v, let $p_v^{H_T}(\vec{x})$ be the probability that the input profile is $\vec{x}$ conditional on v and H_T. If $p_v^{H_T}(\vec{x}) > p_{\vec{x}}$, then $p_v^{H_T}(\vec{x})(p_{\vec{x}}^{-1} - 1) + (-1)(1 - p_v^{H_T}(\vec{x})) > 0$, which means that taking $b_i = \vec{x}$ is strictly better than taking $b_i = \perp$ for each of the agents in T, contradicting the assumption that $\vec{\sigma}_{ACT}$ is a (k, t)-robust equilibrium. Thus, $p_v^{H_T}(\vec{x}) \leq p_{\vec{x}}$ for all $\vec{x}$. Since both $\sum_{\vec{x}} p_v^{H_T}(\vec{x})$ and $\sum_{\vec{x}} p_{\vec{x}}$ are 1, it must be the case that $p_v^{H_T}(\vec{x}) = p_{\vec{x}}$ for all $\vec{x}$. This shows that for every view $\vec{h}_T$ of the adversary, the distribution of possible inputs of honest agents conditional on $\vec{h}_T$ depends only on their inputs and what honest agents output. By Lemma 4.3, this implies that the input of honest agents and the view of the adversary are independent (given the input $\vec{x}_T$ of the adversary and the output v of honest agents), and thus, again by Lemma 4.3, it follows that the distribution of possible views of the adversary depends only on $\vec{x}_T$ and v. This shows that every possible output function of the adversary can be simplified to a function that has as inputs only $\vec{x}_T$ and v, as desired. The argument for $k = 0$ is analogous, except that in this case, if the distribution of possible views of the adversary is not independent of the inputs of the honest agents, the adversary decreases the payoffs of the honest agents, rather than increasing the payoffs of the deviating agents. This completes the proof of Theorem 3.3.

5 EXTENDING THEOREM 3.3

In the construction used for Theorem 3.3, a (k, t)-robust equilibrium that implements $\vec{\sigma} + \sigma_{ACT}$ may not necessarily (non-strictly) $(k + t)$-securely compute f, since if the adversary consists of fewer than $k + t$ malicious agents, the malicious agents might be able to deduce information about the honest agents' inputs without being able to take advantage of it (recall that for $u_i^2(\vec{a}, \vec{x})$ to be non-zero, a subset K with $|K| = k + t$ must all guess the same value for $u_i^2(\vec{a}, \vec{x})$). However, a small variation in the construction of u_i^2 in $\Gamma^{f,k,t}$ gives a game $\Gamma^{f,k}$ such that any strongly $(k, 0)$-robust equilibrium that implements the $(k, 0)$-robust equilibrium of Proposition 4.1 also k-securely computes f, so secure computation can be reduced to implementing strongly $(k, 0)$-robust equilibria for certain mediator games. The idea is that instead of requiring the subset K of agents who do not play G to have size exactly k, we require only that it have size at most k. This modification of u_i^2 leads to some of the problems discussed in Section 4, namely, that if some rational agents act like honest agents except that they share their inputs with other rational agents, the latter agents might be able to guess the input profile and get a strictly positive expected payoff. This scenario is indistinguishable from one in which the agents who shared their input are actually honest and rational agents are just lucky. To deal with this issue, we further modify the payoffs in $\Gamma^{f,k}$ so that if the agents in some subset K with $|K| \leq k$ guess the inputs correctly, then everyone else gets a huge negative payoff (rather than 0, as in the original construction). We can show that if this payoff is sufficiently negative (e.g., $-n$ times the winnings) and there exists a deviation from the agents in K in the $(k, 0)$-robust equilibrium $\vec{\sigma} + \sigma_d$ given in Proposition 4.1 in which rational agents get a positive payoff from u^2, then there exists a deviation from the agents in K in $\vec{\sigma} + \sigma_d$ in which rational agents get a positive payoff from u^2 and they all guess the same value in every possible view (if the negative payoffs are small enough, rational agents not guessing any value gives a negative total payoff for rational agents, even if some of them guess the correct value). (Note that this modification works only for strong (k, t)-robustness, since if we require only (k, t)-robustness, a rational agent may decrease its own payoff if that helps other rational agents, even if the total gain of the rational agents from doing so is negative.) With this, an analogous argument to the one given in the proof of Proposition 4.1 shows that the strategy used in Proposition 4.1 is strongly $(k, 0)$-robust with these payoffs. The rest of the proof is identical to that of Theorem 3.3.

6 PROOF OF THEOREMS 3.4 AND 3.5

For Theorem 3.5(a), note that if $n > 4t$, Theorem 2.4 shows that every function $f : D^n \to D$ can be t-securely computed, and thus t-weak securely computed as well. If $n \leq t$, let $\perp$ be the input assigned to the agents that did not submit an input. It can be easily shown that the protocol where each agent sends no messages and outputs $(\emptyset, f(\perp^n))$ t-securely computes f. Similarly, for Theorem 3.4(a), it can be easily checked that if $n \leq 2t$, the protocol where each agent sends nothing and outputs $([n - t], f(\perp^n))$ t-weak securely computes f.

It remains to show Theorems 3.4(b) and 3.5(b). Our proofs are similar to that of Canetti [10] at a high level: We construct a function f with four inputs, the scheduler schedules the agents so that the fourth agent never gets to participate in the computation, and one of the three remaining agents is malicious and manages to trick the other two participating agents into outputting something inappropriate. Canetti then claims that *conversations* between agents (where a conversation is just the collection of messages sent by two given agents) must be independent of the inputs of the agents, and that agent 3 can send messages to agents 1 and 2 in such a way that agents 1 and 2 believe they should output different values. However, there are two significant problems with this approach:

(a) First, the conversations between the agents might not be totally independent of their inputs, since they can depend on the output of the computation, and this ultimately does depend on the inputs. For example, agents can run Bracha's [9] consensus protocol (which tolerates t malicious agents if $n > 3t$) after terminating the secure computation protocol to decide the output. This would guarantee that all honest agents output the same value at the end of the computation, so their conversations are certainly not independent.
(b) Second, there is a more subtle issue when trying to simultaneously trick gents 1 and 2 into outputting some given values a and b, respectively. Even though Canetti proves that for the function f that he uses and a particular input $\vec{x}$, for each conversation $h_{1,2}$ between 1 and 2, there is a protocol for player 3 that results in a conversation $h_{1,3}$ between 1 and 3 such that 1 outputs a, and that for each conversation $h_{1,2}$ between 1 and 2 there exists a protocol for player 3 that results in a conversation $h_{2,3}$ between 2 and 3 such that 2 outputs b, there might not exist a protocol for agent 3 that results in 1 and 2 having conversation $h_{1,3}$ and agents 2 and 3 having conversation $h_{2,3}$ simultaneously. In fact, if a and b are different and agents run a consensus protocol as in (a), there is not.

Roughly speaking, we deal with these issues as follows. We prove that for our function f, a malicious agent can make all honest agents output the same incorrect value, and we show that in our case there does exist a conversation $h_{1,2}$ between 1 and 2 such that agent 3 can trick both of them simultaneously, as desired (see Lemma 6.1). Some of these techniques can also be applied to prove lower bounds for weak secure computation.

6.1 Proof of Theorem 3.4(b)

Consider the function $f^n : \{0, 1, \perp\}^n \to \{0, 1, \perp\}$ that essentially takes majority between 0 and 1: it outputs 1 if the number of agents with input 1 is greater or equal to the number of agents with input 0, otherwise it outputs 0. Players who do not submit an input are assumed to have input $\perp$. We start by showing that f^4 cannot be 1-weakly securely computed by four agents.

Suppose that f^4 can be 1-weakly securely computed using a protocol $\vec{\sigma}$. Let σ_e^N be the scheduler that schedules agents 1, 2, and 3 cyclically, and right before scheduling an agent, it delivers the messages that were sent by the other agents the last time they were scheduled. After scheduling each of the first three agents N times, it schedules agent 4 as well, adding it to the cyclic order.

Given a view H of the protocol, let $\vec{x}_H$ denote the input profile of agents in H, let H_i denote agent i's view in H, let H_e denote the scheduler's view in H, and let $H_{(i,j)}$ denote the *conversation* between agents i and j (i.e., the messages sent and received between i and j, in addition to the relative times at which i and j were scheduled). We can now prove essentially what BCG claimed to prove (although, as we said, this claim does not hold for the BCG construction).

Lemma 6.1. *There exist N and two (finite) views H and H' of $\vec{\sigma}$ where the scheduler uses σ_e^N, $\vec{x}_H = (1, 0, 1, 1)$, $\vec{x}_{H'} = (0, 1, 1, 1)$, agents 1, 2, and 3 all output 1 in H, agent 4 is never scheduled in either H or H', $H_{1,2} = H'_{1,2}$, and $H_e = H'_e$.*

To prove Lemma 6.1, we first need to prove that if all agents are honest, at most t agents have input 0, and $n \geq 3t$, then the output of a 1-weakly secure computation of f^n will be 1. While this seems obvious (and is true), it is not quite so trivial. For example, it is not true if $n = 3t - 1$. In this case, if we consider an ideal adversary $A = (T, c, h, O)$, in which $|T| = t$, h replaces all inputs of malicious agents with 0, and c chooses all t malicious agents and $t - 1$ additional honest agents, it is easy to check that the output of honest players is 0.

Lemma 6.2. *If $n \geq 3t$ and $\vec{\sigma}$ is a protocol that t-weakly securely computes f^n, then for all schedulers, in all possible views of $\vec{\sigma}$ in which all agents are honest and at most t agents have input* 0*, all agents output* 1*.*

Proof. Let S be the subset of agents that have input 0. Given a scheduler σ_e, consider an adversary $A = (T, \vec{\tau}_T, \sigma_e)$ such that $T \supseteq S$, $|T| = t$, and $\vec{\tau}_T = \vec{\sigma}_T$ (so all the malicious agents follow protocol $\vec{\sigma}$). By definition of t-weak secure computation, the output of honest agents with adversary A should be one that is possible with an ideal adversary of the form $A' = (T, c, h, O)$. However, no matter what the output C of c is, since $|C| \geq n - t$, there will be at least $n - 2t$ honest agents in C, all of them with input 1. Since $n \geq 3t$, this suffices to guarantee that all agents not in T output 1. Since malicious agents play their part of $\vec{\sigma}$, they are indistinguishable from honest agents. Thus, if all agents are honest, all agents not in T output 1. To see that agents in T also output 1 if all agents are honest, consider an adversary $A'' = (T', \vec{\tau}_{T'}, \sigma_e)$ such that $T' \cap T = \emptyset$, $|T'| = t$ (such a set T' always exists since $n \geq 3t$), and $\vec{\tau}_{T'} = \vec{\sigma}_{T'}$. Since honest agents not in $T \cup T'$ (note that $[n] \setminus (T \cup T') \neq \emptyset$) have the same views with A and A'', they must output the same value with both adversaries, and so must output 1 with adversary A''. By definition of t-weak secure computation, since $|T'| = t$, all agents not in T' must output the same value. Thus, since $T \cap T' = \emptyset$, all agents in T also output 1 with adversary A''. Again, since agents in T' are indistinguishable from honest agents, this implies that agents in T also output 1 if all agents are honest. □

Proof of Lemma 6.1. By Lemma 6.2, there exists an integer N such that if agents 1, 2, and 3 are honest, with nonzero probability, they will output 1 with scheduler σ_e^N at or before the Nth time they are scheduled. Let H be a view where the agents use $\vec{\sigma}$, the scheduler uses σ_e^N, the input is $(1, 0, 1, 1)$, agents 1, 2, and 3 are honest and have been scheduled at most N times and all three have outputted 1. By the properties of secure computation, in particular, the secrecy of the inputs, there must exist a view H'' such that $\vec{x}_{H''} = (1, 1, 0, 1)$, $H''_1 = H_1$, and $H''_e = H_e$. (Note that this means that we can assume, without loss of generality, that the scheduler uses protocol σ_e^N.) If this were not the case and agent 1 were malicious in H'', then it would know that the input profile can't be $(1, 1, 0, 1)$ given histories H_1 and H_e. (Recall that we can assume without loss of generality that the malicious agents can communicate with the scheduler.) Similarly, there exists a view H' with $\vec{x}_{H'} = (0, 1, 1, 1)$ such that $H'_2 = H''_2$ and $H'_e = H''_e$. The fact that the scheduler has the same view in H, H', and H'' and that $H_1 = H''_1$ and $H''_2 = H'_2$ implies that $H_{1,2} = H'_{1,2} (= H''_{1,2})$, as desired. In more detail, since $H'_2 = H''_2$, agent 2 sends the same messages to and receives the same messages from agent 1 in H' and H'', so 1 receives the same messages from and sends the same messages to 2 in both H' and H''. Thus, $H'_{1,2} = H''_{1,2}$. A similar argument shows that $H_{1,2} = H''_{1,2}$. □

Now suppose that agents have input profile $\vec{x} = (0, 0, 0, 0)$. We show that there exists a protocol τ_3 for agent 3 such that if all other agents play $\vec{\sigma}$ and the scheduler plays $\vec{\sigma}_e^N$, then with non-zero probability, agents 1 and 2 output 1. This suffices to show that f^4 cannot be 1-weakly securely computed, since honest agents should output 0 when playing with any trusted-party adversary with at most one malicious agent.

Lemma 6.3. *If the agents have input profile* $(0, 0, 0, 0)$*, then there exists a protocol τ_3 for agent 3 such that if all other agents run $\vec{\sigma}$ and the scheduler runs σ_e^N, then with non-zero probability, agents 1 and 2 output 1.*

Proof. Let H and H' be the two views guaranteed to exist by Lemma 6.1. The protocol τ_3 for agent 3 consists of sending agent 1 the messages that agent 3 sends to agent 1 in H' while sending agent 2 the messages that agent 3 sends to agent 2 in H. Suppose that agent 1 has the same random bits as in H', while agent 2 has the same random bits as in H. An easy induction now shows that, in the resulting history, agent 1 will have history H'_1 and agent 2 will have history H_2 after each having been scheduled at most N times, using the fact that, as shown in Lemma 6.1, $H_{12} = H'_{12}$. Thus, by Lemma 6.1, agents 1 and 2 output 1 in this case. This contradicts the fact that $\vec{\sigma}$ 1-weakly securely computes f^4, since Lemma 6.2 shows that, with input profile $(0, 0, 0, 0)$, all honest players output 0. □

It is straightforward to extend this argument to all n and t such that $3t \leq n \leq 4t$. Given n and t such that $3t \leq n \leq 4t$, we divide the agents into four disjoint sets S_1, S_2, S_3, and S_4 such that $0 < |S_i| \leq t$ for all $i \in \{1, 2, 3\}$ and $0 \leq |S_4| \leq t$. Consider a scheduler σ_e^N that schedules agents in S_1, S_2 and S_3 cyclically and, right before scheduling an agent, it delivers the messages that were sent by the other agents the last time they were scheduled. After scheduling each of the agents in $S_1 \cup S_2 \cup S_3$ N times, it schedules the agents in S_4 as well. Suppose that $\vec{\sigma}$ is a protocol for n agents that t-weakly securely computes f^n.

Lemma 6.4. *There exist N and two (finite) views H and H' of $\vec{\sigma}$ where the scheduler uses σ_e^N, $\vec{x}_H = (1_{S_1}, 0_{S_2}, 1_{S_3}, 1_{S_4})$, $\vec{x}_{H'} = (\vec{0}_{S_1}, \vec{1}_{S_2}, \vec{1}_{S_3}, \vec{1}_{S_4})$, agents in $S_1 \cup S_2 \cup S_3$ output 1 in H, agents in S_4 are never scheduled in either H or H', $H_{S_1,S_2} = H'_{S_1,S_2}$ (which is the conversation between the agents in S_1 and the agents in S_2), and $H_e = H'_e$.*

Proof. The proof is analogous to the proof of Lemma 6.1; the subsets S_1, S_2, S_3, and S_4 play the roles of agents 1, 2, 3, and 4, respectively. □

We now have the tools we need to prove Theorem 3.4(b). Given H and H' from Lemma 6.4, consider a protocol $\vec{\tau}_{S_3}$ for agents in S_3 that consists of sending agents in S_1 and S_2 exactly the same messages they would send in H' and H respectively. Again, if agents have input $\vec{0}$, a reasoning analogous to that of Lemma 6.3 shows that, with non-zero probability, agents in S_2 will eventually have view H_{S_2}, and thus will output 1, contradicting the assumption that $\vec{\sigma}$ t-weakly securely computes f^n. This completes the proof of Theorem 3.4(b).

The proof of Theorem 3.5(b) is similar to that of Theorem 3.4(b), and is given in Section 6.2.

6.2 Proof of Theorem 3.5(b)

The proof of Theorem 3.5(b) is similar to that of Theorem 3.4(b). We start with an analogue of Lemma 6.2 which holds for a larger range of values of n:

Lemma 6.5. *Let $n \geq t + 2$ and $\vec{\sigma}$ be a protocol that t-securely computes f^n. Then, in all views of $\vec{\sigma}$ in which all agents are honest and at most $(n - t)/2$ agents have input 0, all agents output 1.*

Proof. Given any scheduler σ_e, if all agents are honest, their output should be one that is possible with a trusted-party adversary of the form $A = (\emptyset, c, h, O)$. No matter what the output C of c is, at most $(n - t)/2$ agents in C have input 0. Since $|C| \geq n - t$, at least half of the agents in C have input 1, and thus all honest agents output 1. □

We also need the following technical result:

Lemma 6.6. *If $t + 2 \leq n \leq 4t$ then*

(a) $n \geq 3\lceil \frac{n-t}{3} \rceil$;
(b) $\lceil \frac{n-t}{3} \rceil \leq \frac{n-t}{2}$.

Proof. If $t + 2 \leq 4t$ then $t > 0$. To prove part (a), note that if $t = 1$, then n can be only 3 or 4. In both cases, the inequality is satisfied. If $t \geq 2$ then $\lceil \frac{n-t}{3} \rceil \leq \lceil \frac{n-2}{3} \rceil \leq \frac{n}{3}$, from which the desired result immediately follows. To prove part (b), let a and b be the two positive integers such that $n - t = 3a + b$ with $1 \leq b \leq 3$. Then $\lceil \frac{n-t}{3} \rceil = a + 1$ and $\frac{n-t}{2} = \frac{3a+b}{2} = a + \frac{a+b}{2}$. Since $n - t \geq 2$, then either $a > 0$ or $b > 1$. Since $b \geq 1$, in both cases, $a + 1 \leq a + \frac{a+b}{2}$. □

Given n and t such that $t + 2 \leq n \leq 4t$, we divide the agents into four disjoint sets S_1, S_2, S_3, S_4 such that $|S_i| = \lceil \frac{n-t}{3} \rceil$ for $i \leq 3$ and $|S_4| \leq t$ (which is always possible, by Lemma 6.5(a)). If $n \geq t + 2$, then by Lemma 6.6(b), $\lceil \frac{n-t}{3} \rceil \leq \frac{n-t}{2}$, and thus by Lemma 6.5, in all views in which all agents are honest and have inputs $(\vec{0}_{S_1}, \vec{1}_{S_2}, \vec{1}_{S_3}, \vec{1}_{S_4})$, $(\vec{1}_{S_1}, \vec{0}_{S_2}, \vec{1}_{S_3}, \vec{1}_{S_4})$ or $(\vec{1}_{S_1}, \vec{1}_{S_2}, \vec{0}_{S_3}, \vec{1}_{S_4})$, all the agents output 1. Reasoning analogous to that used in the proof of Theorem 3.4(b) then shows that f^n cannot be t-securely computed for $t + 2 \leq n \leq 4t$.

It remains to deal with the case where $n = t + 1$. To show that there exist functions that cannot be t-securely computed if $n = t + 1$, we reduce *t-resilient weak consensus* to t-secure computation. The reduction proceeds as follows: Consider a function $g^n : \{0, 1, \perp\} \rightarrow \{0, 1, \perp\}$ such that $g^n(\perp, \dots, \perp) = \perp$, and $g^n(x_1, \dots, x_n) = x_i$ if $x_i \neq \perp$ and $x_j = \perp$ for all $j < i$; that is, g^n outputs the first non-$\perp$ value if there is one, and otherwise outputs $\perp$. Suppose, by way of contradiction, that $\vec{\sigma}$ t-securely computes g^n. Let $\vec{\tau}$ be a protocol identical to $\vec{\sigma}$ except that, whenever agent i would have output (C, v) with σ_i, it outputs v instead if $v \neq \perp$, and otherwise it outputs 0. By the properties of t-secure computation, all honest agents output the same value when running $\vec{\tau}$. Moreover, if all honest agents have input 0 or all of them have input 1, if $n > t$, then the output of the secure computation has the form $(C, 0)$ or $(C, 1)$, respectively. Thus, if there exists a protocol that t-securely computes g^n for $n = t + 1$, then there also is a t-resilient implementation of weak consensus for $t + 1$ agents, contradicting Theorem 2.13. This proves Theorem 3.5(b).

7 PROOF OF THEOREM 3.6

Consider the game $\Gamma^{k,t}$ in which the set of actions of each agent is $\{G, R\} \times \{0, 1\}$. Given an action profile $\vec{a}$, in which each agent i plays $a_i = (Q_i, y_i)$ with $Q_i \in \{G, R\}$ and $y_i \in \{0, 1\}$, let T be the subset of agents i such that $Q_i = R$. If $|T| > k + t$, if $k = 0$ all agents get a payoff of -1, otherwise all agents get a payoff of 1. If $|T| = t + k$ and there exist two agents $i, j \notin T$ such that $y_i \neq y_j$, if $k = 0$ all agents get a payoff of -1, otherwise all agents get a payoff of 1. In all remaining cases, all agents get a payoff of 0. Let g be the function such that $g(Q, y) = y$.

Consider the following protocol $\vec{\sigma} + \sigma_d$ for n agents and a mediator. With σ_i, each agent i sends the mediator its input x_i the first time it is scheduled. The mediator waits until receiving a message containing either 0 or 1, and sends that value y to all agents. The agents play (G, y) whenever they receive y from the mediator. Clearly, this give a (k, t)-robust(respectively, strongly (k, t)-robust) equilibrium, since the only way that agents get a payoff other than 0 with an adversary of size at most $k+t$ is if two honest agents output different values, which cannot happen since they all receive the same value from the mediator. Suppose a protocol $\vec{\sigma}_{ACT}$ is a (k, t)-robust (respectively, strongly (k, t)-robust) implementation of $\vec{\sigma} + \sigma_d$. We show next that (a) for all adversaries $A = (T, \vec{\tau}_T, \sigma_e)$ with $|T| \leq k + t$, all honest agents play the same value y_i, and (b) if all agents are honest and have the same input x, then they output x.

Property (b) follows trivially from the fact that $\vec{\sigma}_{ACT}$ implements $\vec{\sigma} + \sigma_d$: if all agents are honest and have the same input x, the value received by the mediator in $\vec{\sigma} + \sigma_d$ is guaranteed to be x, and thus, in $\vec{\sigma} + \sigma_d$, all honest agents play (G, x).

To prove (a), suppose that there exists an adversary $A = (T, \vec{\tau}, \sigma_e)$ with $|T| \leq k + t$ such that, in some view H of $\vec{\sigma}_{ACT}$ with A, there exist two agents $i, j \notin T$ that play (Q_i, y_i) and (Q_j, y_j), respectively, with $y_i \neq y_j$, $Q_i = R$, or $Q_j = R$. Consider an adversary $A' = (T', \vec{\tau}_{T'}, \sigma_e)$ such that $|T'| = k + t$, $T \subseteq T'$, and $i, j \notin T'$ (we know that such a subset T' exists, since $n > t + k + 1$), and such that agents in T act as in $\vec{\tau}_T$ and agents in $T' - T$ act like honest agents, except that all of them play $(R, 0)$. Since views generated by playing with A and A' are indistinguishable by honest agents, there exists a view H' in $\vec{\sigma}_{ACT}$ with adversary A' in which all honest agents send and receive the same messages, and perform the same actions. If $Q_i = R$ in H, then there are $k + t + 1$ agents that play R in H': the $k + t$ agents in T' and i. Thus, all agents get a payoff of 1 if $k > 0$, contradicting the assumption that $\vec{\sigma}_{ACT}$ is (k, t)-resilient, or all agents get a payoff of -1 if $k = 0$, contradicting the assumption that $\vec{\sigma}_{ACT}$ is t-immune. The same argument shows that $Q_i = G$ in H and H' and, indeed, that all honest agents must play G in H and H'. Now if $q_i \neq q_j$ in H, then $q_i \neq q_j$ in H', so (since all honest agents play G, so exactly $k + t$ agents in H' play R), again, all agents in H' get a payoff of 1 if $k > 0$ and a payoff of -1 if $k = 0$, so we again get the same contradiction as before.

8 CONCLUSION

We have shown that both $(k + t)$-secure computation and the problem of implementing a (k, t)-robust equilibrium with a mediator have a lower bound of $n > 4k + 4t$. Moreover, we have shown that this is also a lower bound for weaker notions of secure computation such as $(k + t)$-strict secure computation and $(k + t)$-weak secure computation. Finally, by considering a number of variants of the definition of secure computation, we also highlighted some of the subtleties in the definition.

ADGH showed that protocols can tolerate more malicious behavior if honest agents can punish rational agents if they are caught deviating. Honest players can perform this punishment by playing an action profile that results in all agents getting an expected payoff that is worse than their payoff in equilibrium. Not all games have such a punishment profile, but ADGH showed that for games that do, every (k, t)-robust protocol with a mediator can be implemented if $n > 3k + 4t$. Finding a matching lower bound for this case remains an open problem.

REFERENCES

[1] I. Abraham, D. Dolev, I. Geffner, and J. Y. Halpern. 2019. Implementing mediators with asynchronous cheap talk. In *Proceedings of the 38th ACM Symposium on Principles of Distributed Computing*. 501–510.
[2] I. Abraham, D. Dolev, R. Gonen, and J. Y. Halpern. 2006. Distributed computing meets game theory: Robust mechanisms for rational secret sharing and multiparty computation. In *Proceedings of the 25th ACM Symposium on Principles of Distributed Computing*. 53–62.
[3] I. Abraham, D. Dolev, and J. Y. Halpern. 2008. Lower bounds on implementing robust and resilient mediators. In *Proceedings of the 5th Theory of Cryptography Conference*. 302–319.
[4] I. Abraham, D. Dolev, and G. Stern. 2020. Revisiting asynchronous fault tolerant computation with optimal resilience. In *Proceedings of the 39th ACM Symposium on Principles of Distributed Computing*. 139–148.
[5] I. Barany. 1992. Fair distribution protocols or how the players replace fortune. *Mathematics of Operations Research* 17, 2 (1992), 327–340.
[6] M. Ben-Or, R. Canetti, and O. Goldreich. 1993. Asynchronous secure computation. In *STOC'93: Proceedings of the 25 Annual ACM Symposium on Theory of Computing*. ACM, New York, NY, 52–61. DOI: https://doi.org/10.1145/167088.167109
[7] M. Ben-Or, S. Goldwasser, and A. Wigderson. 1988. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proceedings of the 20th ACM Symp. Theory of Computing*. 1–10. DOI: https://doi.org/10.1145/62212.62213
[8] M. Ben-Or, B. Kelmer, and T. Rabin. 1994. Asynchronous secure computations with optimal resilience (extended abstract). In *Proceedings of the 13th ACM Symp. Principles of Distributed Computing*. 183–192.
[9] G. Bracha. 1984. An asynchronous $[(n-1)/3]$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*. 154–162.

[10] R. Canetti. 1996. *Studies in Secure Multiparty Computation and Applications*. Ph.D. Dissertation. Technion. citeseer.nj.nec.com/canetti95studies.html.
[11] F. Forges. 1990. Universal mechanisms. *Econometrica* 58, 6 (1990), 1341–64.
[12] L. Lamport. 1983. The weak Byzantine generals problem. *Journal of the ACM* 30, 3 (1983), 668–676. DOI: https://doi.org/10.1145/2402.322398
[13] A. Shamir. 1979. How to share a secret. *Communications of the ACM* 22, 11 (1979), 612–613.

Received 31 July 2021; revised 15 July 2022; accepted 4 December 2022